

CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA

CELSO SUCKOW DA FONSECA – CEFET/RJ

**Interface e comunicação com inversores de
frequência**

Átila Luiz Matos dos Santos

José Vilson Alvarenga Júnior

Prof. Orientador: Dr. Luciano Mendes Camillo

Rio de Janeiro

Dezembro de 2014

CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA

CELSO SUCKOW DA FONSECA – CEFET/RJ

Interface e comunicação com inversores de frequência

Átila Luiz Matos dos Santos

José Vilson Alvarenga Júnior

Projeto final apresentado em cumprimento às
normas do Departamento de Educação Superior
do CEFET/RJ, como parte dos requisitos para obtenção
do título de Bacharel em Engenharia Eletrônica

Prof. Orientador: Dr. Luciano Mendes Camillo

Rio de Janeiro

Dezembro de 2014

Ficha catalográfica elaborada pela Biblioteca Central do CEFET/RJ

S237 Santos, Átila Luiz Matos dos
Interface e comunicação com inversores de frequência / Átila
Luiz Matos dos Santos [e] José Vilson Alvarenga Júnior.—2014.
xvii, 90f. : il.color. , grafs. ; enc.

Projeto Final (Graduação) Centro Federal de Educação
Tecnológica Celso Suckow da Fonseca, 2014.

Bibliografia : f. 89-90

Orientador : Luciano Mendes Camillo

Inclui apêndice e anexo.

1. Inversores elétricos. 2. Motores elétricos de indução. 3.
Sistemas embarcados (Computadores). 4. Controladores
programáveis. 5. Raspberry Pi (Computador). I. Alvarenga Júnior,
José Vilson. II. Camillo, Luciano Mendes (Orient.). III. Título.

CDD 621.31

DEDICATÓRIA

Dedicamos este trabalho às nossas famílias, aos nossos amigos, às pessoas com quem trabalhamos, aos nossos professores e a todas as outras pessoas que também contribuíram para nossa formação pessoal e profissional.

AGRADECIMENTOS

Átila

Agradeço aos meus pais, André Santos e Sandra Matos, por toda dedicação para contribuir para a minha formação como homem e todo apoio e incentivo para contribuir com a minha formação profissional.

Quero agradecer também a todas as pessoas com quem tive oportunidade de trabalhar e que foram essenciais para a formação do meu conhecimento técnico, em especial, agradeço ao senhor Erenilton Masiero por estar presente, me apoiar e me incentivar em toda caminhada durante a minha graduação e ainda, por ceder incondicionalmente toda estrutura e material necessários para o desenvolvimento deste projeto.

Agradeço ainda meu antigo professor engenheiro Ricardo Baldner por ser fundamental na minha escolha profissional, desde a época em que eu fazia curso técnico.

Ainda assim, quero agradecer o senhor Edilson Braga por toda sua contribuição e incentivo para que eu pudesse iniciar minha trajetória da área técnica.

Ao professor orientador Luciano Camillo por nos ajudar e nos orientar com todas as dúvidas para o desenvolvimento do presente projeto.

Por fim, agradeço a todas as outras pessoas, amigos e familiares que contribuíram e contribuem para que eu possa sempre alcançar objetivos maiores.

José

Agradeço primeiramente a meus pais, José Vilson Alvarenga e Gilcimara de Souza Martins, por me incentivar e apoiar durante a graduação, assegurando que essa fosse minha única obrigação durante esses cinco anos. Agradeço também a todos os meus familiares que serviram de suporte e me ajudaram como estudante e contribuíram para minha formação como cidadão.

Agradeço a todos os amigos que fiz durante este tempo no CEFET/RJ, que sempre estiveram junto comigo durante os momentos de alegria, e mais importante no momento que precisava de ajuda na realização de algum trabalho ou nas horas de estudo conjunto.

Agradeço aos profissionais que trabalharam comigo durante o período de estágio, que puderam me passar uma experiência diferente da obtida através das salas de aula.

Agradeço aos docentes que participaram de todo o meu processo de aprendizado, em especial ao professor orientador Luciano Camillo que nos ajudou durante esse último ano na elaboração do projeto final de curso e tirou as dúvidas que surgiam ao longo da confecção do mesmo.

Por fim gostaria de agradecer a todos que não se incluem em um dos parágrafos acima, mas estiveram ao meu lado e contribuíram para que eu pudesse estar me formando.

RESUMO

A tendência por usar motores elétricos de indução substituindo os motores de corrente contínua tem como causa a melhoria em seu controle devido ao uso dos inversores de frequência. Além disso, a economia gerada pelo controle mais otimizado compensa o investimento neste equipamento. Os processos que utilizam motores de indução passam a depender cada vez mais dos inversores de frequência e com isso, a automatização de sensores e atuadores pode ser realizada cada vez mais com o uso das entradas e saídas digitais e analógicas dos inversores de frequência. Contudo, a demanda por aplicações com processamento de informações favorece a implementação da comunicação serial prevista nos equipamentos atuais. Ainda assim, as comunicações, em geral, são realizadas por CLPs (Controladores Lógicos Programáveis) ou até mesmo por placas microcontroladas dedicadas a uma aplicação. A solução adotada no presente trabalho consiste em utilizar uma placa microprocessada com componentes, que podem ser utilizados em diversas aplicações, de baixo custo e com grande facilidade comercial. Neste contexto, o sistema de interface e comunicação com inversores de frequência proposto neste projeto, a partir da placa microprocessada Raspberry Pi modelo B+, buscou construir um código-fonte robusto, para servir como base para processar e executar funções no inversor de frequência, através do protocolo de comunicação Modbus.

Palavras-chave: Inversor de frequência, Motor de indução, Raspberry Pi e Modbus.

ABSTRACT

The modern trend to use electrical induction motors instead of using the direct current motors happens due to the improvement in the control of power electronics. The Variable Frequency Drives (VFD) is the most used and developed equipment to control the induction motors. The safety and the economy provided by this equipment pay the investment. The industrial processes that use induction motors depend more and more of the VFDs and because of that, the automation systems with these elements are using digital and analogs, inputs and outputs to process sensors and actuators. However, the applications that use data processing encourage the implementation of serial communication expected in the current equipment. Nevertheless, the communications are realized by PLC (Programmable Logic Control) or even by microcontrollers embedded boards. The adopted solution in this project is to use a microcontroller embedded board classified as SBC (Single Board Computer), because of the capacity to use it in a lot of applications, its lower costs and the commercial availability. In this context, the interface and communication system with VFDs proposed by this work, using a Raspberry Model B+ board, aimed to build a robustness source code to process and execute functions in the driver, through the Modbus communication protocol.

Keywords: Variable Frequency Drives, Induction motors, Raspberry Pi e Modbus.

SUMÁRIO

1. Introdução	1
1.1 Motivação	1
1.2 Justificativa	1
1.3 Objetivos	2
1.4 Metodologia e trabalho realizado.....	2
1.5 Organização do trabalho	3
2. Fundamentação teórica	4
2.1 Motores elétricos.....	4
2.1.1 Introdução aos motores.....	4
2.1.2 Fundamentos.....	6
2.1.2.1 Indução eletromagnética	6
2.1.2.2 Força eletromagnética	6
2.1.3 Aspectos construtivos	7
2.1.3.1 Carcaça.....	7
2.1.3.2 Estator	7
2.1.3.3 Rotor	7
2.1.4 Princípio de funcionamento.....	8
2.1.5 Características de torque e velocidade	10
2.1.6 Classes	13
2.1.7 Variação de velocidade.....	13
2.2 Inversores de frequência	15
2.2.1 Definição	15
2.2.2 Princípio de funcionamento.....	15
2.2.2.1 Retificação	16
2.2.2.2 Circuito de pré-carga	16
2.2.2.3 Filtragem	17

2.2.2.4	Circuito de frenagem	17
2.2.2.5	Circuito de chaveamento	17
2.2.2.6	Circuito de controle	18
2.2.2.7	Operador	18
2.2.3	Curva V/F	19
2.2.4	Comunicação serial.....	21
2.2.5	Aplicações	21
2.3	Redes industriais	22
2.3.1	Modelo OSI	22
2.3.2	Camada física	24
2.3.3	Camada de enlace	25
2.3.4	Rede MODBUS.....	25
2.3.4.1	Comunicação	26
2.3.4.2	Formato da mensagem	26
2.3.4.3	Erros de comunicação	27
2.3.5	Outras redes utilizadas em ambientes industriais	28
2.3.5.1	Profibus	29
2.3.5.2	ASi	29
2.3.5.3	DeviceNet	29
2.3.5.4	Foundation Fieldbus	30
3.	Sistemas embarcados.....	31
3.1	Histórico.....	31
3.2	Definição.....	31
3.3	Computadores de uma placa (SBC).....	32
3.4	Tipos de SBCs	35
4.	Sistema de interface e comunicação com o inversor de frequência	36
4.1	Inicialização e parametrização do inversor de frequência	36
4.1.1	Seleção.....	36

4.1.2	Instalação e conexão	37
4.1.3	Parametrização.....	40
4.2	Rede para comunicação	44
4.2.1	Camada Física	44
4.2.2	Camada de Enlace	46
4.3	Raspberry Pi.....	48
4.3.1	Definição	48
4.3.2	Características.....	48
4.3.3	Alimentação e conexão.....	50
4.3.4	Instalação do Sistema Operacional.....	51
4.3.5	Modo embarcado	52
4.4	QT	53
4.4.1	Características.....	53
4.4.2	Instalação em modo nativo	54
4.4.3	Bibliotecas e Programação	54
4.5	Interface do usuário	56
4.5.1	Entrada de Dados.....	56
4.5.2	Tela	57
5.	Código-fonte.....	59
5.1	Visão geral	59
5.2	Entrada de dados	60
5.3	Menu	60
5.4	Leitura e escrita no inversor de frequência	61
5.5	Dados do programa	62
5.6	Envio da mensagem	62
5.7	Recebimento da mensagem	62
6.	Campo de aplicação.....	63
7.	Conclusão	64

APÊNDICE A: Código-fonte – mainwindow.cpp	66
ANEXO A: Exemplos de comandos MEMOBUS/Modbus.....	87

LISTA DE FIGURAS

Figura 1: Motor de indução do tipo rotor bobinado [2].....	5
Figura 2: Rotor de um motor de indução do tipo gaiola de esquilo [2].....	8
Figura 3: Motor de indução do tipo rotor bobinado [2].....	8
Figura 4: Densidade de campo magnético no rotor x velocidade síncrona [2].	11
Figura 5: Densidade de campo magnético resultante x velocidade síncrona [2].....	12
Figura 6: Fator de potência x velocidade síncrona [2].	12
Figura 7: Torque induzido x velocidade síncrona [2].....	12
Figura 8: Classes de motores de indução [2].....	13
Figura 9: Diagrama de potência de um inversor de frequência [4].	16
Figura 10: Operador digital [4].....	19
Figura 11: Curvas de tensão de saída e torque em relação a frequência no inversor [1].	20
Figura 12: Modelo OSI [5].	23
Figura 13: Arquitetura, serviços e protocolos dentro do modelo OSI [6].	24
Figura 14: Top 10 SBCs – Pesquisa realizada em 05/2014 [13].....	33
Figura 15: Características mais importantes de SBCs – Pesquisa realizada em 05/2014 [13].	33
Figura 16: Tipos de projetos com SBCs – Pesquisa realizada em 05/2014 [13].....	34
Figura 17: Aplicações com SBCs – Pesquisa realizada em 05/2014 [13].....	34
Figura 18: Fotografia do inversor de frequência utilizado no projeto [4].	36
Figura 19: Diagrama completo de conexões elétricos do inversor de frequência [4].	38
Figura 20: Desenho da localização dos bornes de potência do inversor [4].....	39
Figura 21: Desenho da localização dos bornes de controle do inversor [4].	40
Figura 22: Modos do operador digital [4].	41
Figura 23: Conversor USB/TTL/RS232/RS485 [18].	44
Figura 24: Diagrama do conversor USB/TTL/RS232/RS485 [18].	45
Figura 25: Fotografia da placa Raspberry Pi modelo B+.	49
Figura 26: Desenho mecânico da placa Raspberry Pi modelo B+ [14].....	50
Figura 27: Diagrama da barra de pinos da placa Raspberry Pi modelo B+ [14].....	50
Figura 28: Imagem da tela inicial do Sistema Operacional Raspbian.....	52
Figura 29: Fotografia do teclado numérico utilizado no projeto.	56
Figura 30: Imagem da tela do sistema de interface e comunicação.	57
Figura 31: Máquina de estados do menu do sistema.	61

LISTA DE TABELAS

Tabela 1: Formato padrão de uma mensagem que utiliza o Modbus [4].	26
Tabela 2: Códigos de erro do protocolo MEMOBUS/MODBUS [4].	28
Tabela 3: Grupos de parâmetros do inversor de frequência [4].	42

LISTA DE ABREVIATURAS E SIGLAS

ARM - Acorn RISC Machine

ASCII - American Standard Code for Information Interchange

ASi - Actuator Sensor Interface

ATO - ASi Trade Organization

BMFT - BundesMinisterium für Forschung und Technologie

CAN - Controller Area Network

CEO - Chief Executive Officer

CLP - Controlador Lógico Programável

CPU - Central Processing Unit

CR - Carriage Return

CRC - Cyclic Redundancy Check

EIA - Electronic Industries Alliance

FIP - Factory Instrumentation Protocol

GCC - GNU Compiler Collection

GPIO - General-Purpose Input/Output

HART - Highway Addressable Remote Transducer Protocol

HDMI - High-Definition Multimedia Interface

HSE - High-Speed Ethernet

I2C - Inter-Integrated Circuit

IEC - International Electrotechnical Commissio

IGBT - Insulated Gate Bipolar Transistor

IHM - Interface Homem-Máquina

IP - Internet Protocol

ISA - International Society of Automation

ISO - International Organization for Standardization

LED - Light Emitting Diode

LF - Line Feed

OSI - Open Systems Interconnection

ProfiBus - Process Field Bus

PWM - Pulse Width Modulation

QML - Qt Meta Language

RAM - Random Access Memory

RS232 - Recommended Standard 232

RS485 - Recommended Standard 485

RTS - Request-To-Send

RTU - Remote Terminal Unit

SBC - Single Board Computer

SD - Secure Digital

SPI - Serial Peripheral Interface

TCP - Transmission Control Protocol

TTL – Transistor-Transistor Logic

UART - Universal Asynchronous Receiver/Transmitter

USB - Universal Serial Bus

VGA - Video Graphics Array

XML - Extensible Markup Language

LISTA DE SÍMBOLOS

N - Número de espiras
 Φ - Fluxo magnético
 E - Tensão elétrica.
 F - Força magnética;
 B - Densidade de ampo magnético;
 L - Comprimento do condutor;
 I - Corrente elétrica;
 \emptyset - Ângulo entre a direção do fio e as linhas de campo magnético.
 n_s - Velocidade síncrona
 f_e - Frequência de entrada
 P - Número de polos
 e_{ind} - Tensão induzida
 v - Velocidade da barra em relação ao campo magnético
 B - Vetor de densidade de fluxo magnético
 l - Comprimento do condutor em um campo magnético
 τ_{ind} - Torque induzido
 k - Constante
 B_r - Densidade de fluxo magnético no rotor
 B_s - Densidade de fluxo magnético no estator
 n_{esc} - Velocidade de escorregamento
 n_s - Velocidade síncrona
 n_m - Velocidade mecânica
 B_{res} - Densidade de campo magnético resultante no estator.
 δ - Ângulo entre o B_r e B_{res} .
 V - Volt
 f - Frequência
 psi - Libra por polegada quadrada
 mA - Miliampère
 Gbps - Giga bits por segundo
 bps - Bits por segundo
 Mbps - Mega bits por segundo

kbps - Quilo bits por segundo

km - Quilômetro

m - Metro

mV - Milivolt

Capítulo 1

Introdução

1.1 Motivação

Cerca de 25% da energia elétrica no Brasil é consumida pelo sistema motriz, segundo o PROCEL (2009). Os motores elétricos trifásicos de indução ocupam hoje, grande parte do mercado de motores. A ampliação do uso desse tipo de motor está sem dúvida relacionada ao controle de grandezas que antes não era possível. O controle de torque e posição através de inversores de frequência possibilita sistemas mais complexos, mais econômicos e mais robustos.

As aplicações de inversores de frequência junto aos motores de indução estão em praticamente todo o tipo de indústria, como a de açúcar e álcool, metalúrgicas, refrigeração, têxtil, entre tantas outras. As máquinas que utilizam o conjunto inversor e motor são, por exemplo, centrífugas, elevadores, pontes rolantes, ventiladores, extrusoras e muitas outras.

1.2 Justificativa

As aplicações que envolvem os motores de indução e os inversores de frequência estão presentes em todas as áreas da indústria e no transporte de cargas. Por isso, o desenvolvimento deste projeto garante que seu protótipo possa ser utilizado para futuras aplicações comerciais.

Além disso, os temas que serão abordados são tendências para equipamentos e programas eletrônicos, o que faz o projeto bem moderno para o mercado da eletrônica e da computação.

A experiência adquirida com a criação de sistemas eletrônicos utilizando processamentos gráficos, processamento de sinais e comunicação seriais, além da disponibilidade do inversor de frequência e do motor para utilização neste projeto, tornam a oportunidade única de desenvolver um protótipo com essas características.

1.3 Objetivos

Desenvolver uma solução para aplicação industrial que possa servir também de protótipo para outras soluções futuras. Esta solução contará com o uso de uma Placa Raspberry Pi que será utilizada como interface e comunicação com um inversor de frequência, onde através de um monitor e um teclado será possível acionar e parar um motor, além de programar e monitorar funções do inversor de frequência. Por se tratar de uma proposta básica de controle de um sistema, através da implementação de uma configuração de hardware de controle e comunicação, e do conhecimento adquirido sobre motores e inversores de frequência, permitirá que o desenvolvimento de futuros projetos semelhantes torne-se mais rápido e facilitado.

1.4 Metodologia e trabalho realizado

Visando a tecnologia empregada nos inversores de frequência que permite não só o controle do motor como também o uso de outras variáveis como entradas e saídas, digitais e analógicas, a proposta de trabalho do presente projeto é criar um sistema eletrônico que permite monitorar ou alterar parâmetros em inversores de frequência utilizando a comunicação serial. O sistema possibilitará a criação de automatizações e interfaces gráficas para facilitar a operação das aplicações que envolvam motores.

O sistema será composto por um SBC (*Single Board Computer*), acrônimo em inglês para computador de placa única, com um software embarcado que será desenvolvido a partir de uma ferramenta de programação com licença gratuita (freeware), conhecido como Qt, e uma interface gráfica que possibilita a operação dos inversores de forma simples. Esse sistema foi projetado para possuir as menores dimensões possíveis, assim as adaptações necessárias nos locais onde possa ser instalado sejam mínimas.

Na primeira etapa do projeto será realizado um levantamento de material e estudo sobre os temas relevantes para o desenvolvimento do trabalho. Logo após, será criado um sistema mecânico de teste contendo o inversor de frequência, o motor e a placa Raspberry Pi. A partir deste ponto, a placa será programada e por fim, serão realizados os testes para verificar o desempenho do projeto.

1.5 Organização do trabalho

O trabalho está organizado de forma que o capítulo 2 apresenta uma breve introdução teórica sobre todos os elementos envolvidos no projeto, ou seja, motores trifásicos de indução, inversores de frequência e redes industriais.

No capítulo 3 são abordados os sistemas embarcados, apresentando um histórico, como foi o desenvolvimento da área e quais as suas tendências. Será apresentado também o conceito de *Single Board Computer* (SBC) e o seu papel no cenário de sistema embarcado hoje, além de uma análise das repercussões do surgimento dessa nova tecnologia no mercado, tanto industrial, quanto doméstico.

No capítulo 4 são apresentados e detalhados os processos de desenvolvimento da interface de comunicação e o de seleção dos equipamentos e ferramentas a serem utilizadas. Neste capítulo cada etapa da comunicação será explicada, assim como a maneira pela qual se interconectam. Além disso, serão abordados todos os detalhes da criação da interface gráfica, bem como as ferramentas utilizadas para sua criação.

Ao fim será explicado como foi montado o código fonte do programa responsável pelo controle e exibição de informações do sistema (o código completo do programa está no apêndice A). E também é apresentada uma breve dissertação sobre as áreas de aplicações de um sistema como esse projetado.

Capítulo 2

Fundamentação teórica

2.1 Motores elétricos

2.1.1 Introdução aos motores

Os motores elétricos são máquinas de conversão de energia utilizadas em todos os setores da indústria. Sua capacidade de transformar energia elétrica em energia mecânica linear ou rotativa, bem como sua flexibilidade de utilização para diversos tipos de carga popularizam o seu uso.

O estudo das técnicas de construção dos motores elétricos está ligado aos estudos sobre a eletricidade e o magnetismo. A primeira relação entre a corrente elétrica e um campo magnético foi descoberta pelo físico dinamarquês Hans Oersted, em 1820. Percorrendo uma corrente elétrica em um fio condutor, verificou-se uma que o ponteiro de uma bússola se deslocava quando próxima do fio. A partir daí, construiu-se o galvanômetro.

Na época da descoberta de Oersted, muitos cientistas evoluíam seus estudos nesta linha de pesquisa. Em 1831, o britânico Michael Faraday demonstrou a fenômeno da indução eletromagnética onde segundo seus estudos, ao variar um campo magnético no interior de um enrolamento de fio condutor, este fio produzia uma tensão elétrica em seus terminais.

Em 1832, o cientista italiano S. Dal Negro construiu a primeira máquina de corrente alternada com um movimento pendular.

Posteriormente, muitos cientistas, como Nikola Tesla, Galileo Ferraris e Michael Dolivo-Dobrovolski desenvolveram estudos no sentido de relacionar as grandezas elétricas às grandezas magnéticas. Estes estudos produziram conteúdo científico que possibilitaram o desenvolvimento dos motores elétricos. Atualmente, classificam-se os motores da seguinte maneira:

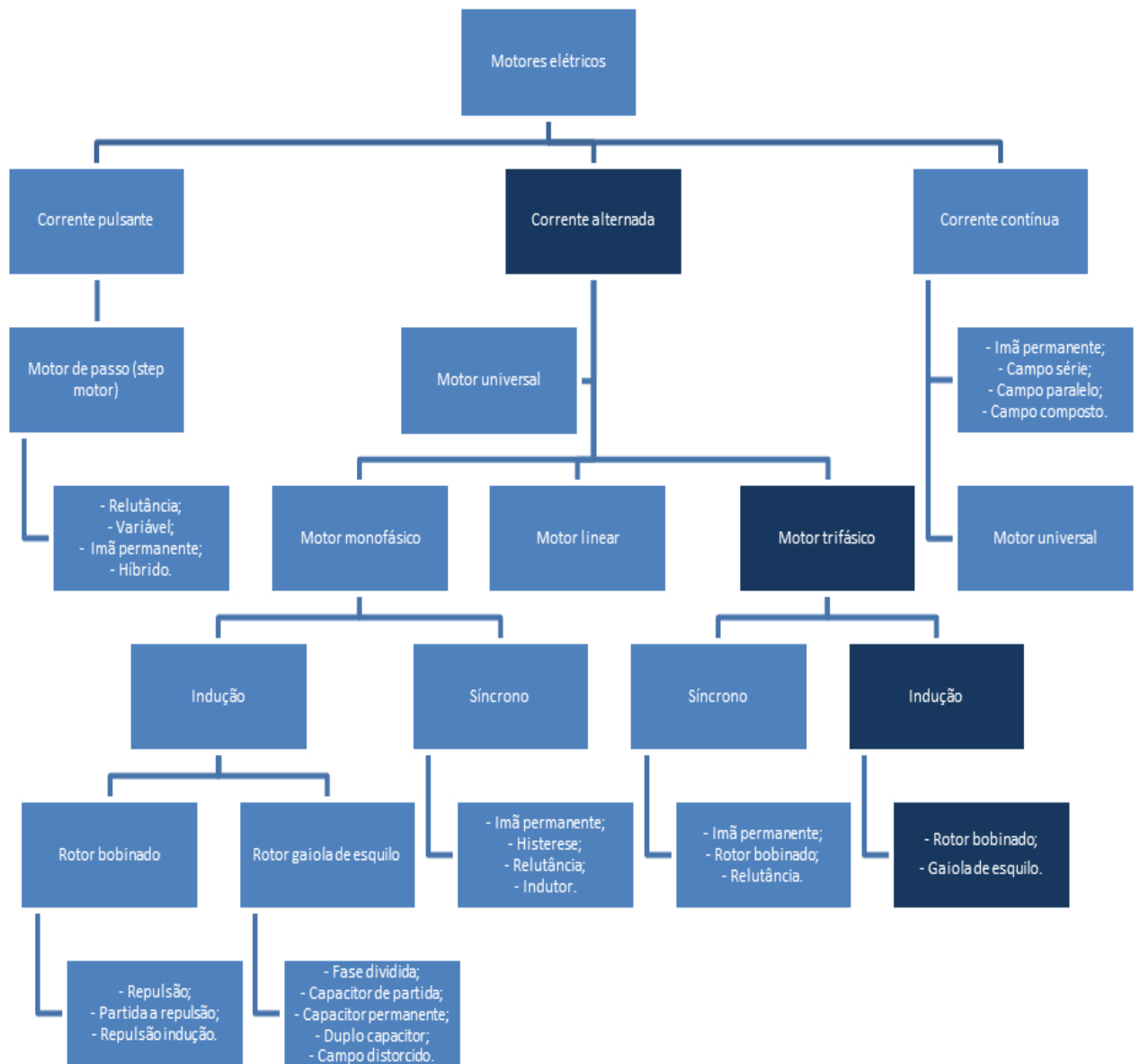


Figura 1: Motor de indução do tipo rotor bobinado [2].

O motor de indução trifásico em destaque na figura acima é escopo do presente trabalho. Este tipo de motor pode ser classificado em rotor bobina e gaiola de esquilo.

2.1.2 Fundamentos

2.1.2.1 Indução eletromagnética

O funcionamento dos motores elétricos de indução é explicado, em teoria, através do princípio de indução eletromagnética, experimentada pela primeira vez em 1831, por Michael Faraday.

Ao variar o campo magnético no interior de um enrolamento de fio condutor, há o surgimento de tensão elétrica nos terminais deste enrolamento. Pode-se variar um campo magnético movimentando um ímã permanente ou variando a corrente elétrica em um eletroímã.

A variação no tempo do fluxo magnético está diretamente ligada ao número de espiras (quantidade de voltas do enrolamento), entretanto a tensão elétrica é oposta ao sentido do fluxo, como provado pela lei de Lenz. Então, matematicamente, pode-se escrever a indução eletromagnética como:

$$e(t) = -N \frac{d\Phi}{dt} \quad (1)$$

Onde:

$N \Rightarrow$ número de espiras,

$\Phi \Rightarrow$ fluxo magnético e

$e \Rightarrow$ tensão elétrica.

2.1.2.2 Força eletromagnética

Quando um fio condutor percorrido por uma corrente elétrica está sob um campo magnético, pode-se observar uma força mecânica de natureza magnética. Esta força é matematicamente descrita por:

$$F = B \cdot L \cdot i \cdot \sin\theta \quad (2)$$

Onde:

$F \Rightarrow$ força magnética;

$B \Rightarrow$ densidade de campo magnético;

$L \Rightarrow$ comprimento do condutor;

$I \Rightarrow$ corrente elétrica;

$\emptyset \Rightarrow$ ângulo entre a direção do fio e as linhas de campo magnético.

2.1.3 Aspectos construtivos

2.1.3.1 Carcaça

As carcaças são peças, em geral, de ferro fundido utilizadas para suportar o rotor e o estator, bem como dissipar o calor produzido pelas reações do motor. A carcaça é fixada a uma tampa traseira e uma tampa dianteira onde estão os rolamentos de esfera que sustentam o eixo. Há carcaças abertas que não possuem aletas e há carcaças fechadas com aletas. As carcaças também tem o papel de isolamento elétrico.

2.1.3.2 Estator

O estator é a parte fixa do motor onde são gerados os fluxos magnéticos transmitidos ao rotor. Em uma analogia aos transformadores, o estator pode ser considerado como um primário do ponto de vista elétrico. Em sua composição estão as bobinas dos enrolamentos que são utilizadas por esse fluxo magnético e que possui seus terminais ligados à caixa com terminais para alimentação do motor.

2.1.3.3 Rotor

Os motores de indução podem apresentar rotor do tipo gaiola de esquilo ou rotor bobinado.

O rotor de gaiola de esquilo consiste em barras condutoras conectadas em anéis em forma de gaiola.

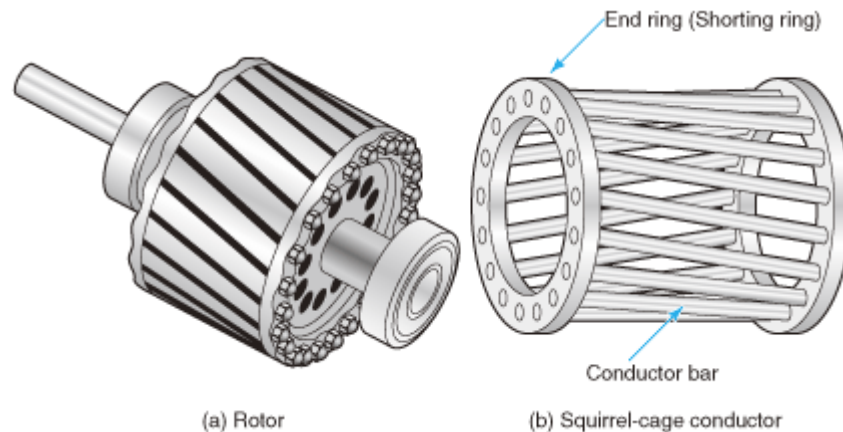


Figura 2: Rotor de um motor de indução do tipo gaiola de esquilo [2].

O motor de rotor bobinado possui enrolamentos com três fases complementares aos enrolamentos do estator. As fases deste rotor são geralmente conectadas em estrela e ligadas ao anel coletor no eixo do motor. As ligações do rotor bobinado são acessíveis nas escovas do estator, onde é possível inserir resistências extras. Esta característica ajuda a modificar as curvas de torque x velocidade do motor. Este motor é mais caro que o motor de gaiola de esquilo e sua manutenção também é mais cara por causa das escovas e dos anéis coletores.

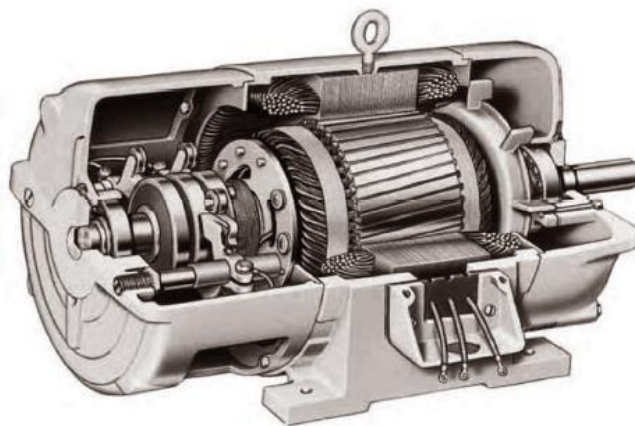


Figura 3: Motor de indução do tipo rotor bobinado [2].

2.1.4 Princípio de funcionamento

Quando uma tensão elétrica trifásica de corrente alternada é aplicada nos terminais de um motor, os enrolamentos do estator, interligados em modo estrela ou triângulo, sob efeito de uma corrente elétrica, criam um campo magnético.

O rotor do motor de indução interage com esse campo magnético do estator e produz também uma corrente elétrica alternada e consequentemente um campo magnético.

Esta corrente elétrica no rotor em defasagem de ângulo com o campo elétrico do estator, segundo a lei de Lorentz, produz uma força binária no motor capaz de movimentá-lo.

Na prática, pode-se verificar a velocidade do estator, conhecida também como velocidade síncrona, através dos parâmetros da frequência aplicada e do número de polos do motor, como a seguir:

$$n_s = \frac{120f_e}{P} \quad (3)$$

Onde:

$n_s \Rightarrow$ velocidade síncrona

$f_e \Rightarrow$ frequência de entrada

$P \Rightarrow$ número de polos

A tensão induzida em uma barra do rotor, para o caso do rotor de gaiola de esquilo, será:

$$e_{ind} = (v \times B) \cdot l \quad (4)$$

Onde:

$e_{ind} \Rightarrow$ tensão induzida

$v \Rightarrow$ velocidade da barra em relação ao campo magnético

$B \Rightarrow$ vetor de densidade de fluxo magnético

$l \Rightarrow$ comprimento do condutor em um campo magnético

O torque induzido será o produto vetorial do campo magnético do rotor vezes uma constante e o campo magnético do estator:

$$\tau_{ind} = k B_r \times B_s \quad (5)$$

Onde:

$\tau_{ind} \Rightarrow$ torque induzido

$k \Rightarrow$ constante

$B_r \Rightarrow$ densidade de fluxo magnético no rotor

$B_s \Rightarrow$ densidade de fluxo magnético no estator

A velocidade de escorregamento será a diferença entre a velocidade síncrona e a velocidade mecânica no eixo do motor:

$$n_{esc} = n_s - n_m \quad (6)$$

Onde:

n_{esc} \Rightarrow velocidade de escorregamento

n_s \Rightarrow velocidade síncrona

n_m \Rightarrow velocidade mecânica

O escorregamento será:

$$s = \frac{n_{esc}}{n_s} (\times 100\%) = \frac{(n_s - n_m)}{n_s} (\times 100\%) \quad (7)$$

2.1.5 Características de torque e velocidade

As características operacionais de um motor de indução estão relacionadas principalmente a três grandezas fundamentais: torque, velocidade e potência.

A partir de um campo elétrico aplicado aos terminais do estator, um campo magnético é criado por causa de seus enrolamentos. Como visto anteriormente, seu comportamento é similar ao de um transformador. Neste caso, o campo magnético está relacionado à corrente de magnetização.

No rotor, uma tensão é induzida em função do campo magnético do estator. Esta tensão produz uma corrente e um campo magnético.

A equação do torque produzido pelo rotor pode ser extraída da equação:

$$F = Il \times B \quad (8)$$

Como o torque é proporcional à força e a corrente em uma barra é proporcional ao campo do rotor, a fórmula para o torque induzido é:

$$\tau_{ind} = k B_r \times B_{res} = k B_r B_{res} \text{sen} \delta \quad (9)$$

Onde:

B_r \Rightarrow Densidade de campo magnético no rotor.

B_{res} \Rightarrow Densidade de campo magnético resultante no estator.

k \Rightarrow Constante.

δ \Rightarrow Ângulo entre o B_r e B_{res} .

O campo magnético no rotor é diretamente proporcional ao fluxo de corrente até a sua saturação, obviamente. Esta corrente aumenta com o aumento do escorregamento e é inversamente proporcional à velocidade mecânica do motor.

O campo resultante no estator é diretamente proporcional à tensão aplicada nos enrolamentos e por isso é aproximadamente constante.

O δ é o ângulo B_{res} e B_r . Este ângulo é sempre igual ao ângulo do fator de potência (θ_r) mais 90° , pois θ_r é o ângulo entre a tensão e a corrente no rotor. Esta tensão tem a mesma fase que B_{res} e a corrente é perpendicular à B_r . Então:

$$\text{sen} \delta = \text{sen}(\theta_r + 90) = \cos \theta_r \quad (10)$$

A partir da análise dos três elementos da equação do torque induzido em relação à velocidade síncrona do motor, pode-se estabelecer a curva de torque pela velocidade síncrona, como segue nas figuras abaixo:

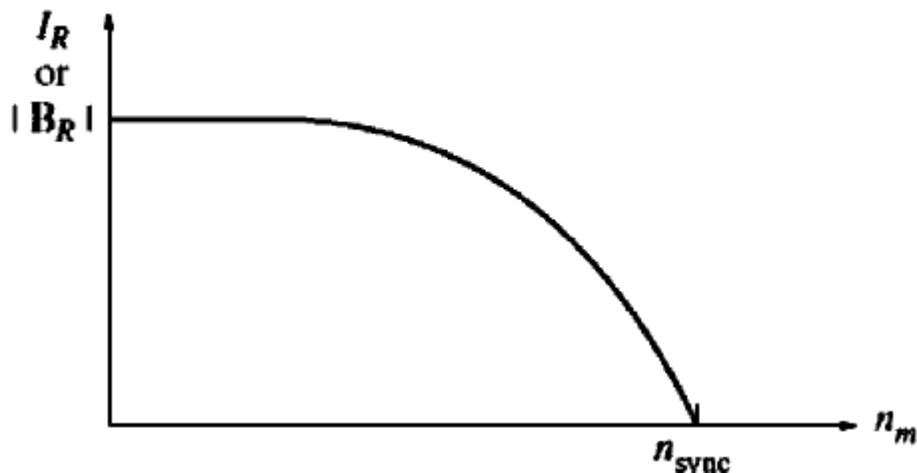


Figura 4: Densidade de campo magnético no rotor x velocidade síncrona [2].

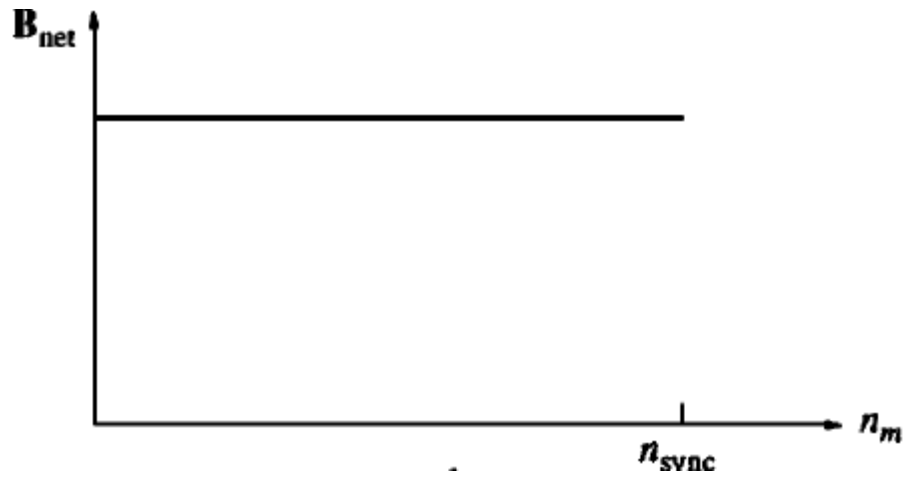


Figura 5: Densidade de campo magnético resultante x velocidade síncrona [2].

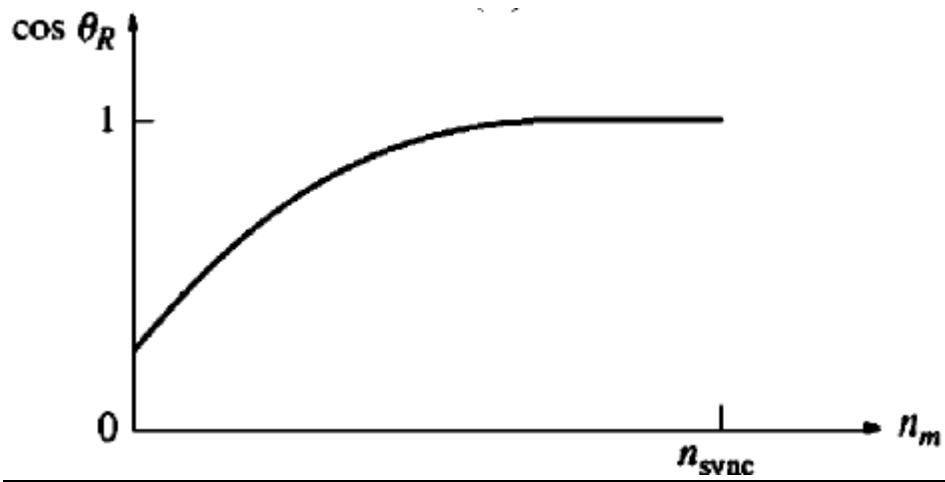


Figura 6: Fator de potência x velocidade síncrona [2].

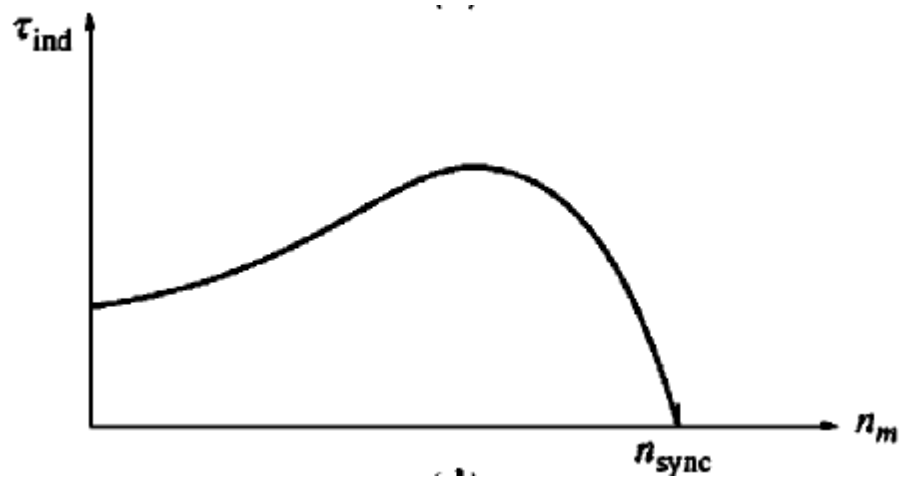


Figura 7: Torque induzido x velocidade síncrona [2].

2.1.6 Classes

Para atender às necessidades de cada aplicação envolvendo motores elétricos de indução, alguns parâmetros como as resistências dos enrolamentos no estator e o núcleo magnético são modificados. Estas modificações são caracterizadas como classes. Na figura abaixo, pode-se observar as curvas de torque x velocidade síncrona em cada classe:

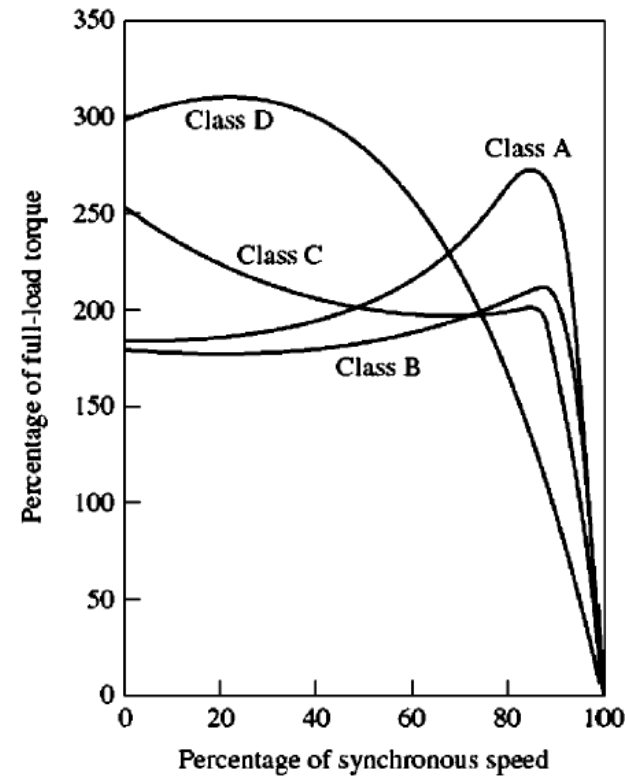


Figura 8: Classes de motores de indução [2].

2.1.7 Variação de velocidade

Mesmo com os motores de indução operando em uma velocidade aproximadamente constante para um mesmo tipo de carga, em certas aplicações há a necessidade de que essa velocidade varie, como no caso dos elevadores, por exemplo.

Há três maneiras de alterar a velocidade de um motor de indução: variação mecânica (método indireto), alteração do número de polos do motor (método direto) ou variação da frequência (método direto). Sendo este último escopo do presente trabalho.

O método de variação mecânica consiste em alterar a velocidade de operação com o uso de polias variadoras, engrenagens, correias ou outros elementos mecânicos. A velocidade é alterada de acordo com a relação de transmissão entre os elementos.

Para a alteração de velocidade diretamente no motor, há somente duas formas possíveis de acordo com a Equação 1.

Ou se altera o número de polos do motor ou sua frequência. Para alterar o número de polos, utiliza-se a ligação conhecida como Dahlander, onde a comutação ocorre pela mudança dos polos em uma relação de 1:2.

Para a ligação Dahlander, pode-se utilizar a ligação Δ/YY para cargas com conjugado constante, ligação YY/Δ para cargas com potência constante ou ligação $Y-YY$ para cargas com conjugado variável.

O ultimo método direto de controle de velocidade e o mais robusto é a alteração da frequência do motor. Para que esta variação ocorra são necessários equipamentos de eletrônica de potência que produzem uma corrente senoidal com as características de frequência e tensão apropriadas para variar a velocidade do motor. Estes equipamentos são conhecidos como inversores de frequência.

2.2 Inversores de frequência

2.2.1 Definição

Os inversores de frequência são equipamentos eletrônicos capazes de variar a tensão, frequência e corrente fornecida a um motor elétrico de indução ou síncrono.

Cabe ressaltar que o escopo do presente trabalho são os inversores de frequência que transformam uma corrente alternada em contínua e depois convertem novamente para corrente alternada.

Há também, os conversores alimentados com corrente contínua (cicloconversores) e os que são alimentados com corrente alternada e convertem diretamente para a forma de onda alternada para alimentação do motor (conversor matricial). Neste caso, o equipamento tem um fluxo de energia bidirecional, fornecendo potência ao motor durante a operação e devolvendo energia para a rede no período de regeneração.

Também conhecidos como variadores, os inversores de frequência atuais fazem uso da tecnologia PWM (*Pulse Width Modulation*) para produzir formas de ondas senoidais com tensão e frequência variáveis. Deste modo, aplicando-se uma tensão elétrica trifásica aos terminais de entrada do equipamento, haverá em sua saída uma tensão elétrica trifásica também senoidal. Esta forma de onda senoidal alimenta o motor e controla duas grandezas: velocidade angular e torque.

2.2.2 Princípio de funcionamento

O princípio de funcionamento de um inversor de frequência consiste em converter uma forma de onda senoidal na entrada em corrente contínua e, usando dispositivos de chaveamento, converter essa corrente em senoidal novamente, mas com amplitude e frequência definidos pelo circuito de controle.

A figura abaixo ilustra um inversor de frequência em blocos:

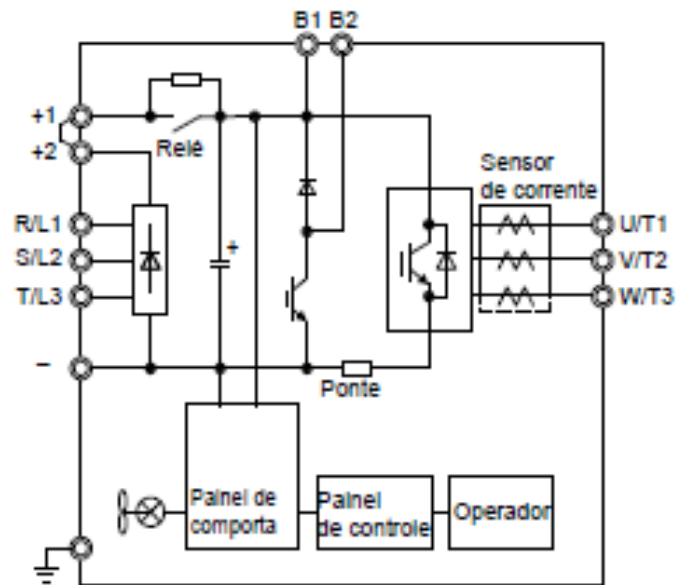


Figura 9: Diagrama de potência de um inversor de frequência [4].

Mesmo com muitos elementos de eletrônica de potência e microeletrônica, as etapas essenciais para o funcionamento do inversor de frequência são:

- Retificação;
- Circuito de pré-carga;
- Filtragem;
- Circuito de frenagem;
- Circuito de chaveamento;
- Circuito de controle;
- Operador.

2.2.2.1 Retificação

A partir do ponto inicial de alimentação R, S e T, a corrente trifásica senoidal é convertida em corrente monofásica pulsativa, ou seja, sem a parte negativa da senóide. Esta conversão é realizada por uma ponte retificadora trifásica.

2.2.2.2 Circuito de pré-carga

Esta corrente pulsativa percorre um circuito conhecido como circuito de pré-carga, composto por um resistor em paralelo a contatos de um relé em estado normalmente aberto.

Este circuito tem a finalidade de limitar a corrente inicial do inversor e posteriormente acionar o relé fazendo com que a corrente deixe de circular pelo resistor.

O circuito de pré-carga limita a corrente inicial, pois neste primeiro momento toda corrente é utilizada para carregar os capacitores que filtram a forma de onda pulsativa. Como a amplitude da corrente inicial em um capacitor descarregado é altíssima, o resistor de pré-carga tem o papel de limitar seu valor.

2.2.2.3 Filtragem

A forma de onda pulsativa agora é filtrada através de um banco de capacitores de alto valor que a convertem em corrente contínua. Esta corrente é utilizada para alimentar o circuito de controle do inversor de frequência e os dispositivos de potência.

2.2.2.4 Circuito de frenagem

Além disso, a Figura 8, mostra no circuito de potência do variador, os terminais B1 e B2 que são utilizados para a conexão de resistores com capacidade de dissipar potências elevadas, conhecidos como resistores de frenagem.

O circuito de frenagem dissipa através desses resistores a energia produzida pelo motor durante o período de regeneração que ocorre durante sua frenagem.

2.2.2.5 Circuito de chaveamento

A conversão de corrente contínua em corrente alternada é realizada baseada na tecnologia PWM. Esta tecnologia consiste em chavear um circuito de corrente contínua para produzir uma tensão média variável.

A modulação por largura de pulso (PWM) pode ser explicada em um circuito de iluminação, por exemplo. Em um circuito contendo uma lâmpada LED e um interruptor para acender e apagar a lâmpada deseja-se reduzir seu brilho pela metade. Baseando-se no princípio de funcionamento da tecnologia PWM, este brilho pode ser reduzido ao chavear o interruptor acendendo e apagando a lâmpada em intervalos contínuos de tempo. Deste modo, se a corrente elétrica é de 10 miliampères com a lâmpada acesa, chaveando o interruptor em intervalos de tempo idênticos, este circuito apresentará uma corrente média na saída de 5

miliamperes, reduzindo o brilho da lâmpada pela metade. Ou seja, o controle do chaveamento em intervalos de tempo controla a corrente ou a tensão média na carga, neste caso a lâmpada, e por consequência, pode-se variar sua amplitude.

Analogamente aos inversores de frequência, a lâmpada representa o motor e o interruptor representa os dispositivos eletrônicos acionados por circuitos de controle que provocam em sua saída uma tensão média com uma forma de onda senoidal. Para a produção desta forma de onda são necessários pelo menos seis dispositivos de chaveamento no circuito de potência do inversor.

Atualmente, o principal dispositivo de chaveamento utilizado no inversor de frequência é o IGBT (*Insulated Gate Bipolar Transistor*). Este transistor bipolar possui alta capacitância em sua porta que é controlada por circuitos de controle que trabalham em conjunto para produzir a forma de onda senoidal. Enquanto em sua porta a tensão atinge cerca de 10V, seus outros dois terminais de potência atingem tensões máximas que variam normalmente entre 600V à 1200V.

2.2.2.6 Circuito de controle

O circuito de controle é responsável por enviar sinais para as portas (*gate*) dos IGBTs para o chaveamento que transforma a corrente contínua em corrente alternada, como mencionado no item anterior. Além disso, o circuito de controle armazena todos os parâmetros do usuário e gerencia os terminais de controle externos.

Nos inversores de frequência atuais estão disponíveis terminais de entradas digitais, entradas analógicas, saídas digitais e saídas analógicas utilizados para automação e controle de processos. Todas essas funções são processadas e armazenadas na placa de controle que ainda faz a leitura de sensores para corrigir e avisar sobre possíveis falhas no sistema.

2.2.2.7 Operador

O operador, também conhecido como IHM (Interface Homem-Máquina), mostra ao usuário, informações referente à velocidade, corrente de saída do inversor de frequência e histórico de alarmes, por exemplo. A partir do operador, pode-se configurar o inversor de frequência para operar o motor, modificar curvas, criar rampas de aceleração e desaceleração, entre muitas outras configurações.

Abaixo, segue a figura de um operador digital típico:

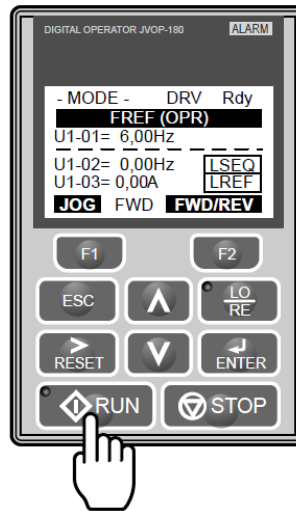


Figura 10: Operador digital [4].

2.2.3 Curva V/F

Por consequência da lei de Faraday para ondas senoidais, a tensão elétrica aplicada à bobina do estator do motor de indução é dada por:

$$E = 4,44 f N \Phi \quad (11)$$

Agrupando as constantes, tem-se:

$$\Phi = k \frac{E}{f} \quad (12)$$

Onde:

$\Phi \Rightarrow$ Fluxo no entreferro

$k \Rightarrow$ Constante 4,44 vezes o número de espiras da bobina (N)

$E \Rightarrow$ Tensão elétrica

$f \Rightarrow$ frequência

O fluxo no entreferro é um elemento indispensável para o desempenho do motor de indução com relação ao seu conjugado. Para que o motor possa trabalhar sempre com as mesmas características, este fluxo deve permanecer constante.

Visto isso, um dos parâmetros mais importantes no ajuste do inversor de frequência é a curva V/F (tensão x frequência).

A configuração desta curva depende da carga e do método de controle utilizado no inversor. Para o controle em malha aberta, ou seja, sem a realimentação da posição do inversor, há dois métodos de controle: o escalar e o vetorial.

O método de controle escalar consiste em padronizar a curva V/F para determinado regime de operação do motor para que seu torque permaneça constante até a velocidade nominal. Porém, em alguns casos há a necessidade de se obter um torque elevado em baixas rotações que não é possível com o este controle escalar.

O controle vetorial soluciona o problema em baixas rotações variando a tensão e a frequência em cada ponto de operação de modo a otimizar o torque mesmo para pontos críticos de trabalho do motor.

As figuras abaixo ilustram a mudança das grandezas de tensão e torque em relação à frequência. Com isso, é possível verificar que, para uma curva V/F típica, onde a tensão e a frequência crescem proporcionalmente até a frequência nominal, o torque permanece constante até esta frequência e a potência cresce até este mesmo ponto. Para frequências maiores do que a nominal, por características do próprio motor há o enfraquecimento do campo e, por isso, o torque diminui com o aumento da frequência e a potência permanece constante.

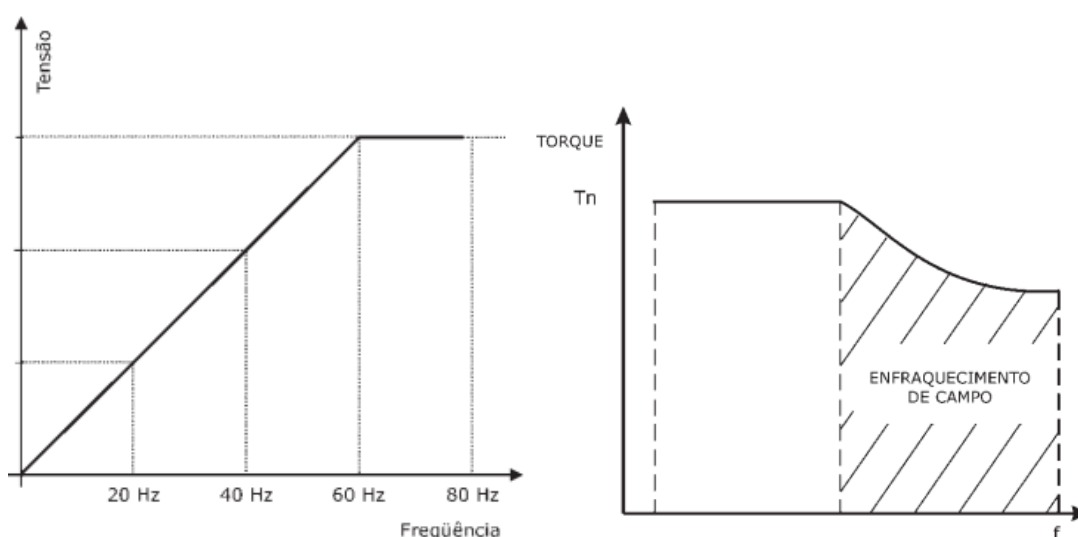


Figura 11: Curvas de tensão de saída e torque em relação a frequência no inversor [1].

2.2.4 Comunicação serial

Além das entradas e saídas analógicas e digitais, mencionados anteriormente, os inversores de frequência permitem a comunicação serial através do protocolo como Modbus.

Utilizando um controlador, como o caso de um CLP (Controlador Lógico Programável), por exemplo, é possível controlar até 255 inversores em uma mesma rede de comunicação.

Em muitas indústrias, essa forma de comunicação é utilizada para monitorar e gerenciar processos, bem como acionar inversores de frequência e modificar curvas através do controlador. Além disso, com a comunicação Modbus, é possível, por exemplo, integrar o inversor a um sistema supervisório que monitora todo o processo em uma única tela.

2.2.5 Aplicações

Há basicamente três tipos de aplicações para o motor e o inversor de frequência: deslocamento de fluidos, manipulação de cargas e processamento de materiais.

Algumas máquinas que trabalham com deslocamento de fluido são as bombas, os ventiladores e os compressores. A manipulação de cargas é realizada por talhas, guinchos, elevadores, pontes rolantes, escadas rolantes, entre outras. O processamento de materiais pode ser dividido em materiais metálicos e materiais não-metálicos. Alguns exemplos de máquina para processamento são os moinhos, os misturadores e as máquinas agrícolas.

2.3 Redes industriais

Algum tempo atrás as indústrias eram obrigadas a comprar seus equipamentos de um mesmo fabricante, pois assim era garantida a compatibilidade entre eles. Porém com a atuação conjunta de diversas áreas como controle, comunicação e computação, foi possível que um sistema heterogêneo surgisse nas indústrias e a partir desse momento não é mais necessário a utilização de uma solução completa de determinado fabricante.

O grande responsável pela heterogeneidade de equipamentos na indústria foi o desenvolvimento de padrões de comunicação, que permite a intercomunicação de equipamentos diversos independente de seus fornecedores. Tendo em vista a interoperabilidade de operação foram desenvolvidos diversos padrão de redes de campo.

E apesar de um único desenvolvedor não estar mais dominando todo o processo de produção dentro uma indústria ele pode trabalhar sobe demanda e o cliente tem uma maior variedade de produtos no mercado, pois agora é garantida a compatibilidade entre eles.

Na década de 40 os sinais pneumáticos de 3-15 psi eram o padrão utilizado na indústria. Com o passar do tempo, por volta da década de 60, e a evolução tecnológica, os sinais analógicos de 4-20mA foram introduzidos na monitoração do processo. O próximo passo seria a utilização da computação e sensores inteligentes para monitoração e controle da fábrica.

Foi nesse período que surgiu a necessidade da criação de um padrão de comunicação, e diversos grupos se juntaram com essa finalidade, entre eles a *International Society of Automation* (ISA), a *International Electrotechinal Commission* (IEC), o comitê do PROFIBUS e o comitê do FIP, normas alemã e francesa respectivamente. Surge então o comitê internacional IEC/ISA SP50 *Fieldbus*.

O mercado de redes industriais se caracteriza pelos diversos tipos de padrão, apesar de alguns poucos centralizarem os negócios. São eles MODBUS, ProfiBus, ASi, DeviceNet, Foundation Fieldbus, e mais recentemente, Ethernet e TCP/IP.

2.3.1 Modelo OSI

O modelo OSI (*Open Systems Interconnection*) foi desenvolvido pela *International Organization for Standardization* (ISO) com o objetivo de padronizar os protocolos de comunicação entre os equipamentos de diversos fornecedores.

O padrão OSI foi pensado dessa forma para cada uma das camadas executasse uma função bem definida e o fluxo de dados entre elas fosse o menor possível, sendo necessário para isso a boa definição de limites das camadas.

Caso seja necessário um nível maior de abstração camadas auxiliares devem ser criadas, levando em consideração que o sistema também deve continuar sendo robusto e rápido. Então se funções desnecessárias são executadas em uma camada, ou a mesma função é executada em camadas diferentes o modelo deve ser repensado.

A figura abaixo apresenta a configuração final do modelo OSI:

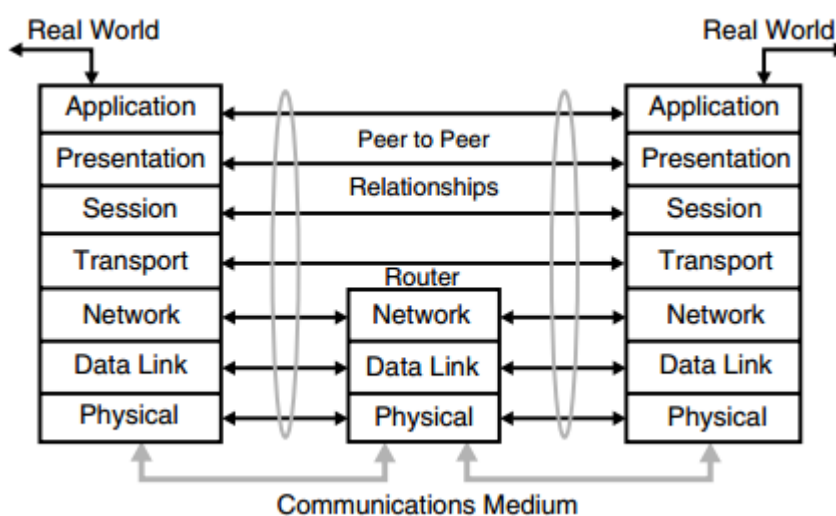


Figura 12: Modelo OSI [5].

O modelo OSI refere-se apenas a interconexão de sistemas abertos, só que a comunicação não se limita apenas a transmissão e recepção de dados, dois sistemas podem ser baseados no modelo e não utilizarem os mesmos protocolos de comunicação e prover os mesmos serviços, por fim não se comunicando. E essa é uma das limitações intrínsecas a este padrão, para os sistemas se interconectarem é necessário que coexistam 3 níveis de compatibilidade: protocolos, serviços e arquitetura.

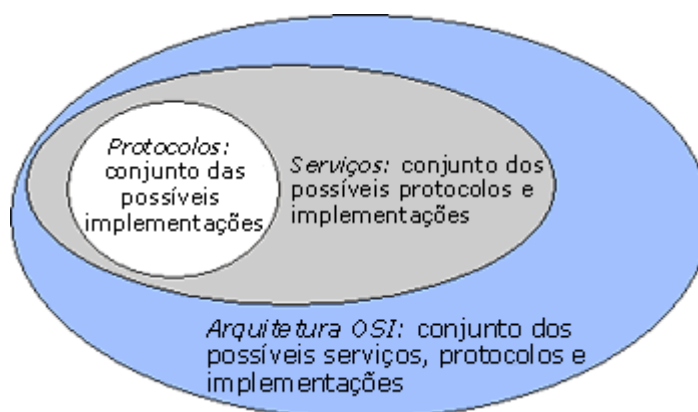


Figura 13: Arquitetura, serviços e protocolos dentro do modelo OSI [6].

Por conta dessa limitação a contribuição do OSI não foi exatamente como modelo para interconexão e sim como referência para outros modelos que viriam a ser implantados na indústria e até mesmo no dia a dia, como é o caso do TCP/IP.

2.3.2 Camada física

A camada física é responsável pela transmissão de dados através de um meio físico, seja ele cabo coaxial, cabo ethernet, fibra ótica, etc. Sua função é garantir que a informação que chega ao receptor seja a mesma que saiu do transmissor, e para isto é preciso que os níveis de tensão do sinal digital seja bem definido, ou um bit que originalmente era 1 passa a ser 0 e é necessária a retransmissão.

No caso deste projeto a interface física será o padrão EIA-485 que é um dos padrões mais utilizados na indústria nos dias de hoje. Ele é uma extensão do EIA-422 e garante a mesma distância e velocidade de transmissão, mas permite um número maior de conexões na mesma linha de transmissão.

O padrão determina que o número máximo de transceptores interconectados em uma mesma malha é 32. Porém alguns fabricantes já desenvolvem equipamentos que contam como $\frac{1}{2}$ ou até mesmo $\frac{1}{4}$ de um equipamento comum, fazendo com que esse limite se estenda a 64 ou 128.

A conexão física de uma interface 485 é feita a 2 fios, A e B, o conector A é conhecido como A-, TxA ou Tx+, e o conector B como B+, TxB ou Tx-. Mas como o princípio de funcionamento é baseado em tensão diferencial o terminal A terá tensão entre -1,5V e -6V em relação ao terminal B quando B em nível lógico alto e +1,5V e +6V quando B nível lógico baixo. A tensão diferencial entre os 2 terminais deverá ser $\pm 5V$.

2.3.3 Camada de enlace

A camada de enlace é a responsável pela formatação do dado a ser enviado para as camadas superiores. Como citado, a camada física emite os dados brutos, 0 e 1, e a camada de enlace deve agrupar esses dados em frames, segmentando-os para serem posteriormente despachados.

Toda informação enviada deve seguir um padrão predefinido pelo protocolo de comunicação utilizado, e o receptor deve enviar ao transmissor uma mensagem informando o recebimento correto do quadro de dados. Caso o quadro esteja incorreto uma mensagem de erro deve ser sinalizado para que aconteça a retransmissão da informação.

O protocolo serial MODBUS será utilizado na camada de enlace do presente trabalho no processo de comunicação entre o terminal e o equipamento. A seguir o mesmo será apresentado de forma mais detalhada.

2.3.4 Rede MODBUS

Desenvolvido pela Gould Electronics, hoje Schneider Electric, o protocolo MODBUS foi um dos padrões que se consolidou nos últimos anos na indústria por ser um protocolo aberto e por ser um dos mais difundidos entre os fabricantes para interconexão de seus equipamentos.

Apesar de realizar uma comunicação segura, o MODBUS sofre com as limitações da interface física o EIA-232/485, pois para os padrões atuais, onde um cabo ethernet e a fibra ótica podem trafegar dados à 10Gbps, um *baud rate* máximo de 115000 bps é bem baixo. Além do fato da interconexão entre muitos dispositivos ser difícil de ser implementada e também de ser mantida.

O MODBUS é um protocolo mestre-escravo, e para cada mestre podem estar associados até 247 dispositivos escravos, e todos eles devem estar configurados com o mesmo tipo de paridade e *baud rate*.

2.3.4.1 Comunicação

A comunicação deverá ser sempre iniciada pelo dispositivo mestre e pode ser destinada a um dispositivo (modo *Unicast*) ou a todos os dispositivos conectados na malha (modo *Broadcast*).

No caso de uma mensagem de *unicast* o mestre envia um pedido a um escravo e entra em aguarda uma resposta deste dispositivo, quando a resposta chegar ela será processada e o processo pode então ser reiniciado. Quando uma mensagem de *broadcast* for iniciada o mestre simplesmente aguarda um tempo pra que todos os escravos recebam a mensagem.

Do outro lado da comunicação os escravos ficam ociosos aguardando que um pedido seja recebido, assim que o mesmo chega é feita a checagem da mensagem, o pedido é realizado e uma mensagem de resposta é enviada de volta ao mestre.

2.3.4.2 Formato da mensagem

A comunicação do MODBUS é baseada em troca de mensagens assim como a maioria dos protocolos. A tabela 1 determina a forma padrão do frame de uma mensagem:

Tabela 1: Formato padrão de uma mensagem que utiliza o Modbus [4].

ENDEREÇO DO SLAVE	CÓDIGO DA FUNÇÃO	DADO	CRC
8 bits	8 bits	n*8 bits	16 bits

Caso seja enviada uma mensagem de *broadcast*, onde todos os dispositivos escravos irão receber a mensagem, os primeiros 8 bits devem ser 0, por definição do protocolo.

Existem diversos códigos de função, mas o presente trabalho se limitará aos 3 disponíveis no equipamento utilizado que são necessários para o desenvolvimento do projeto, o código 0x03 que é responsável pela leitura de registro, o 0x10 que altera os valores dos registros e o 0x08 que é o *loopback test*.

Na leitura de registros o mestre, no caso o sistema supervisor, deve montar um frame com as características supracitadas, endereço, código, dado e CRC. No campo do dado a mensagem deverá conter o registro inicial a ser alterado e quantidade de registros a serem lidos. Ao final deverão ser contabilizados 8 bytes de informação.

O *loopback test* é um teste simples para verificação do funcionamento da comunicação. Esse teste consiste em enviar um dado, que será definido previamente pelo usuário, e o frame de resposta deverá ser exatamente igual ao frame do pedido. O frame será constituído assim como no caso anterior de 8 bytes, e o campo dos dados será composto por 4 bytes, onde 2 serão um código de teste e os outros 2 serão o dado.

A alteração de múltiplos registros irá cobrir três etapas da comunicação, acionar e parar o motor, além de alterar a frequência de referência. O frame irá conter dessa vez 11 bytes, os 4 destinados a endereçamento, código da função e CRC, e mais 7 bytes com os dados para a mudança dos valores. Esses 7 bytes iram ter 2 bytes destinados ao registro inicial a ser alterado, 2 bytes com o número de registros que irão sofrer alterações, incluindo o registro inicial, no caso deste projeto apenas 1 registro. 1 byte informando a quantidade de bytes dos registros, por exemplo seja 1 registro, 2 bytes. E finalmente os 2 bytes que irão conter o dado a ser escrito.

2.3.4.3 Erros de comunicação

O item 2.3.4.1 descreve o processo de comunicação executado com sucesso, porém nem sempre a troca de mensagens será bem sucedida. Por isso o protocolo define padrões para sinalização de erros.

O erro de comunicação pode acontecer no envio ou recebimento da mensagem, na execução do pedido ou erro de CRC. No caso da ocorrência de um erro o dispositivo mestre irá receber no frame de resposta o function code da mensagem enviada alterada seguindo o padrão:

$$\text{Function Code (error)} = \text{Function Code} + 0x80. \quad (13)$$

Se por exemplo numa mensagem de leitura de registro incorreto o function code da resposta será 0x83.

O dado contido no frame de resposta irá conter um determinado error code, que será responsável pela identificação do erro. Abaixo está a tabela de error codes.

Tabela 2: Códigos de erro do protocolo MEMOBUS/MODBUS [4].

Error Code	Error Name
	Cause
01H	Function Code Error
	• Attempted to set a function code from a PLC other than 03H, 08H, and 10H.
02H	Register Number Error
	• None of the register numbers exist. • Attempted to send a broadcast message that did not start with 0001H or 0002H.
03H	Bit Count Error
	• Read data or write data is greater than 16 bits. • While the number of bits in the write data message is not ???
21H	Data Setting Error
	• Control data or parameter write data is outside the allowable setting range. • Attempted to write a contradictory parameter setting.
22H	Write Mode Error
	• Attempted to write while the drive was operating to a parameter that cannot be written to during run. • During an EEPROM data error (CPF06), the PLC attempted to write to a parameter other than A1-00 to -05, E1-03, or o2-04. • Attempted to write to read-only data.
23H	DC Bus Undervoltage Write Error
	• Attempted to write from the PLC during an undervoltage fault (Uv1). • Attempted to execute and Enter command from the PLC during Uv1.
24H	Write Error During Parameter Process
	• PLC attempted writing to the drive while the drive was processing parameter data.

A Tabela 2 especifica tanto erros de troca de mensagens quanto de execução. O erro 01H é característico de um frame montado de forma incorreta, enquanto o erro 22H é o exemplo de um erro de execução, quando o escravo não executar determinada ação naquele momento.

2.3.5 Outras redes utilizadas em ambientes industriais

Como citado na introdução de redes industriais, não existe um único padrão que dominante no que diz respeito a controle de processos e comunicação entre dispositivos. Mas existem os que mais se destacam, e esse tópico tem o intuito de descrever alguns deles.

2.3.5.1 ProfiBus

O ProfiBus (Process Field Bus) é um protocolo desenvolvido pelo BMFT (BundesMinisterium für Forschung und Technologie) em conjunto com diversas empresas de automação.

Esse padrão posteriormente dominou a Europa e é largamente utilizado na América do Norte, América do Sul e algumas partes da África e Ásia.

O ProfiBus permite a ligação de até 127 dispositivos num raio de até 24km. As taxas de transmissão variam entre 9600 bps e 12 Mbps. A distância que o dispositivo se encontra do servidor irá influenciar diretamente na taxa de transmissão de dados.

2.3.5.2 ASi

O ASi (Actuator Sensor Interface) é sistema de rede aberto desenvolvido para conectar sensores binários e atuadores. Foi criada uma organização composta por diversas empresas, entre elas Siemens, Datalogic Products e Turck, para verificar que os produtos que são desenvolvidos atualmente estejam cientes desse novo padrão. Essa organização foi nomeada ATO (ASi Trade Organization).

Grande parte dos sensores binários e atuadores não precisam de uma grande quantidade de bytes para informar sua situação, e o padrão ASi tem como objetivo otimizar o processo de troca de mensagens entre esses dispositivos. Como consequência a ligação a dispositivos controladores inteligentes não é realizada, devido a capacidade do tamanho da mensagem ser insuficiente.

2.3.5.3 DeviceNet

DeviceNet é uma rede de baixo nível orientada ao dispositivo, baseia-se em CAN (Controller Area Network) e foi desenvolvida por Allen Bradley. O objetivo deste padrão é interconectar dispositivos de baixo nível, como sensores, a dispositivos de alto nível, controladores.

O conector do DeviceNet é responsável tanto pela alimentação como pela troca de dados entre os dispositivos. É possível a ligação através de um barramento, onde cada interconexão deste barramento pode ser ramificada em diversas outras.

Pode ser montada uma rede com até 64 nós, que podem ser removidos a qualquer momento sem comprometer a rede. As taxas de transmissão podem ser de 125, 250 ou 500 kbps.

2.3.5.4 Foundation Fieldbus

Foundation Fieldbus é uma arquitetura aberta para integração de um ambiente industrial. O protocolo foi criada da união de diversos grupos interessados no desenvolvimento de um padrão único a ser utilizado para comunicação de equipamentos industriais e substituir o padrão que era utilizado, o HART.

Uma das grandes vantagens do Foundation Fieldbus em relação ao DeviceNet e ao ProfiBus, é que não é necessário a utilização de um CLP para o controle e supervisão do processo, pois os equipamentos de campo podem realizar essas funções.

A camada física difere-se no que diz respeito ao equipamento que será ligado. Se for um equipamento de campo existe o padrão H1 que limita-se a uma taxa de transferência de 31,25 kbps e pode conectar equipamentos a até 1,9 km de distância. Já para interligar CLPs, computadores, etc., é utilizado o padrão HSE que utiliza cabos ethernet e pode atingir taxas de transmissão de até 100 Mbps, porém é limite pelo cabo ethernet a uma distância de 100 m.

Capítulo 3

Sistemas embarcados

3.1 Histórico

É indiscutível que a guerra é um dos grandes propulsores para os avanços tecnológicos, e não foi diferente com os computadores. Durante a segunda guerra mundial era necessário que as mensagens nazistas codificadas que fossem interceptadas pudessem ser lidas.

E após a obtenção de uma das máquinas Enigma, pode-se então desenvolver o Mark I que apesar de ser dedicado a uma única função não pode ser definido como um sistema embarcado, pelas razões que serão explicitadas mais à frente.

Durante o período da guerra fria e a corrida espacial foi desenvolvido o Apollo Guidance Computer, que iria operar a Apollo nas expedições lunares. Esse seria considerado o primeiro sistema embarcado criado. Mas a produção em massa de um sistema embarcado só iria acontecer posteriormente com o sistema guia para os mísseis nucleares.

O próximo grande avanço se daria com o avanço da pesquisa com semicondutores e o surgimento dos circuitos integrados, onde um grande número de transistores era encapsulado em um único chip, reduzindo dessa forma o custo e o tamanho de forma significativa.

A partir desse momento seria possível então o desenvolvimento de circuitos com tamanho reduzido, alto poder de processamento, com um baixo custo. Essa combinação de fatores foi essencial para a evolução dos sistemas embarcados, pois é mais barato desenvolver um chip para um sistema que execute um conjunto de funções reduzidos em larga escala a comprar um sistema complexo e fazê-lo dedicado.

3.2 Definição

Para que se defina um sistema embarcado, primeiro é necessário que se explique o que é um sistema. “Um sistema é uma maneira de se trabalhar organizando ou executando uma ou diversas tarefas seguindo uma série de regras estabelecidas, um programa ou plano. E também

a forma na qual diversas partes se agrupam e trabalham em conjunto de acordo com um programa ou plano.” [10].

A partir da definição de sistema pode-se então definir o que é um sistema embarcado. “Um sistema embarcado é um sistema que tem um software embarcado em um hardware ou computador tornando ele específico para determinada aplicação, parte de uma aplicação, produto ou parte de um sistema maior.” [10].

Um sistema embarcado deve ser pensado para que tenha peso e tamanho reduzido, pois quando dois sistemas executam a mesma função e apresentam o mesmo desempenho, o que possuir dimensões reduzidas será um sistema melhor, porque apresenta maior integração que o outro.

3.3 Computadores de placa única (SBC)

Um computador de placa única é definido como um computador construído em uma única placa contendo processador, memória e entradas/saídas. Como já mencionado, o principal fator proporcionador pela criação desse tipo de computador é o avanço na pesquisa de semicondutores.

Cada vez mais no dia a dia é possível notar a presença de um SBC, nos smartphones, nos tablets, e também passou a ser um grande atrativo para desenvolvedores autônomos. A *The Linux Foundation* e a *LinuxGizmos.com* realizaram uma pesquisa através de um sistema online chamado *SurveyMonkey* que teve início em 8 de maio de 2014 com duração de 10 dias para montar um perfil do usuário dessas placas.

Foram entrevistadas 777 pessoas e o sistema de ranking foi montado da seguinte forma: $\text{total} = (1^{\text{a}} \text{ opção}) \times 3 + (2^{\text{a}} \text{ opção}) \times 2 + (3^{\text{a}} \text{ opção}) \times 1$

As Figuras 13,14, 15 e 16 mostram os resultados dessa pesquisa:

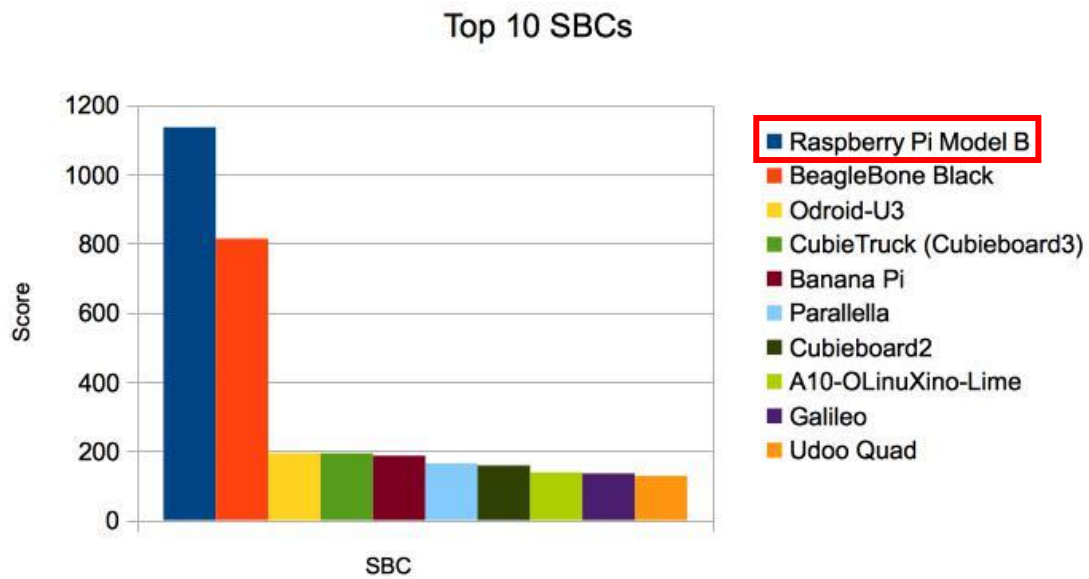


Figura 14: Top 10 SBCs – Pesquisa realizada em 05/2014 [13].

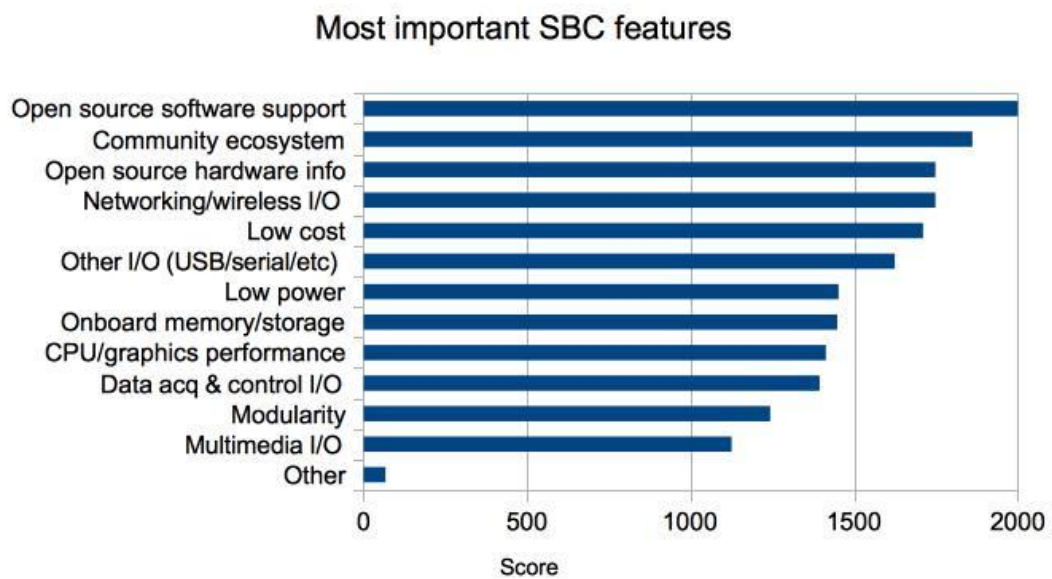


Figura 15: Características mais importantes de SBCs – Pesquisa realizada em 05/2014 [13].

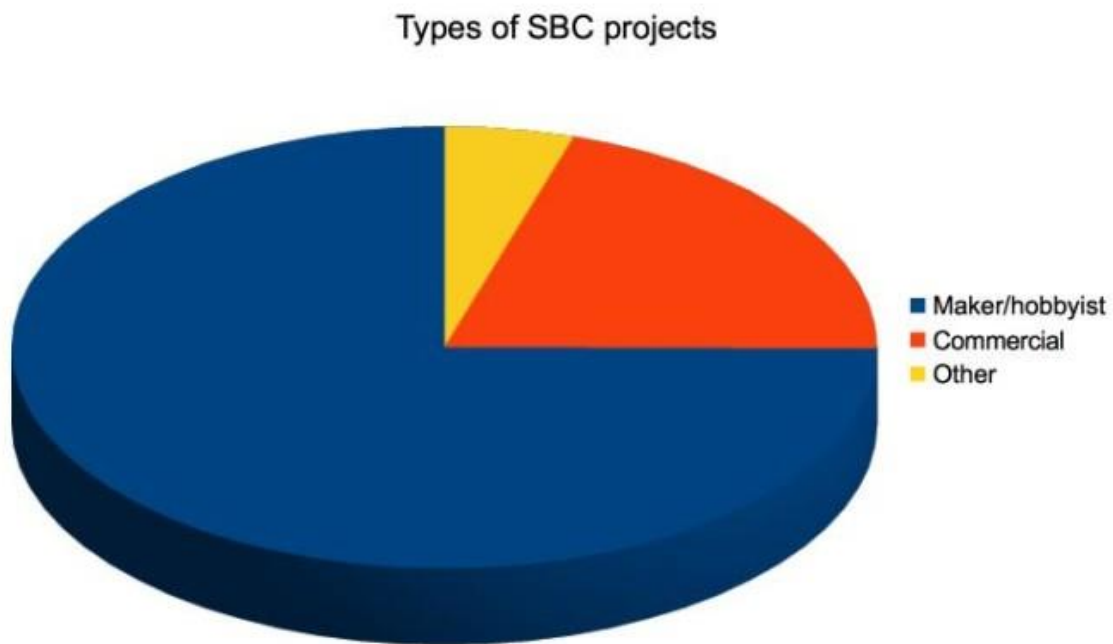


Figura 16: Tipos de projetos com SBCs – Pesquisa realizada em 05/2014 [13].

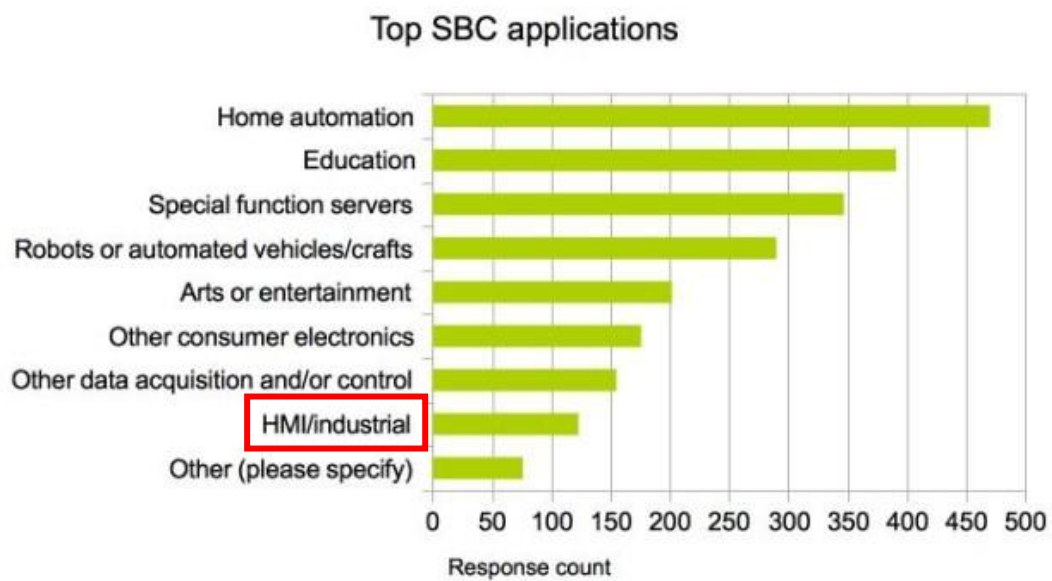


Figura 17: Aplicações com SBCs – Pesquisa realizada em 05/2014 [13].

A partir desses dados pode-se notar que a grande maioria dos consumidores é de desenvolvedores e também que existem dois modelos que se destacam dos demais que são o Raspberry Pi Model B e o BeagleBone Black. Ambos possuem características parecidas, porém o Raspberry Pi é mais barato, o que o torna um produto muito atraente.

Os outros dados mostram que o principal fator de escolha da placa a ser utilizado é o fator da utilização de um software aberto e que as duas principais aplicações são a automação residencial, o que pode explicar a grande quantidade de pessoas que compram por hobby, e a área educacional. Destaque para a aplicação de IHM/Industrial, que é o foco deste trabalho.

3.4 Tipos de SBCs

Como visto na pesquisa existe uma grande quantidade de computadores de uma placa no mercado atualmente. Um dos principais motivos é que um determinado modelo não serve para todas as aplicações possíveis e nem todos os modelos fornecem os mesmos recursos, como por exemplo, um processador de alta velocidade.

Os recursos e configurações não são as únicas coisas que diferem os SBCs, outra questão importante é o tipo de processador com que cada placa utiliza. Como já mencionado anteriormente, uma das grandes aplicações dos SBCs é a comunicação móvel, os processadores usados nessa aplicação é o processador ARM (*Acorn RISC Machine*). A arquitetura ARM é caracterizada pelo baixo consumo de energia, devido a simplificação das instruções para uma maior eficiência num ciclo de clock.

Já a algum tempo existem no mercado placas que utilizam processadores baseados na arquitetura x86, a mesma arquitetura utilizada nos computadores pessoais. Que inclusive é uma de suas possíveis aplicações, substituir os computadores que executam funções básicas, como navegação na internet. Pois ocupam um espaço bem menor e consomem menos.

Capítulo 4

Sistema de interface e comunicação com o inversor de frequência

4.1 Inicialização e parametrização do inversor de frequência

4.1.1 Seleção

A escolha do inversor de frequência depende da escolha do motor. E a escolha do motor depende da aplicação. Basicamente, dois são os parâmetros necessários para o dimensionamento de um inversor de frequência: a tensão de alimentação e a corrente nominal (ou potência) do motor. Para que o equipamento não opere em uma região próxima da saturação, o valor da corrente nominal do motor é ainda multiplicado por um fator que depende da aplicação.

Para o presente projeto, o inversor de frequência foi selecionado com uma margem de tolerância bem elevada em relação ao motor utilizado. Para o desenvolvimento e teste com a comunicação serial, utilizou-se um inversor de frequência da marca YASKAWA, linha A1000 e um modelo da classe de 200 Volts e 56 Amperes (CIMR-AU2A0056).



Figura 18: Fotografia do inversor de frequência utilizado no projeto [4].

Este inversor de frequência tem a capacidade de operar em frequências que variam de 0 Hertz até 400 Hertz. Além disso, há a possibilidade de trabalhar com os métodos de controle escalar ou vetorial em malha aberta ou em malha fechada (com a realimentação através de *encoder*).

4.1.2 Instalação e conexão

A instalação e conexão do inversor de frequência são realizadas através de seus terminais de potência e de controle. A figura abaixo ilustra todos os terminais e suas respectivas funções:

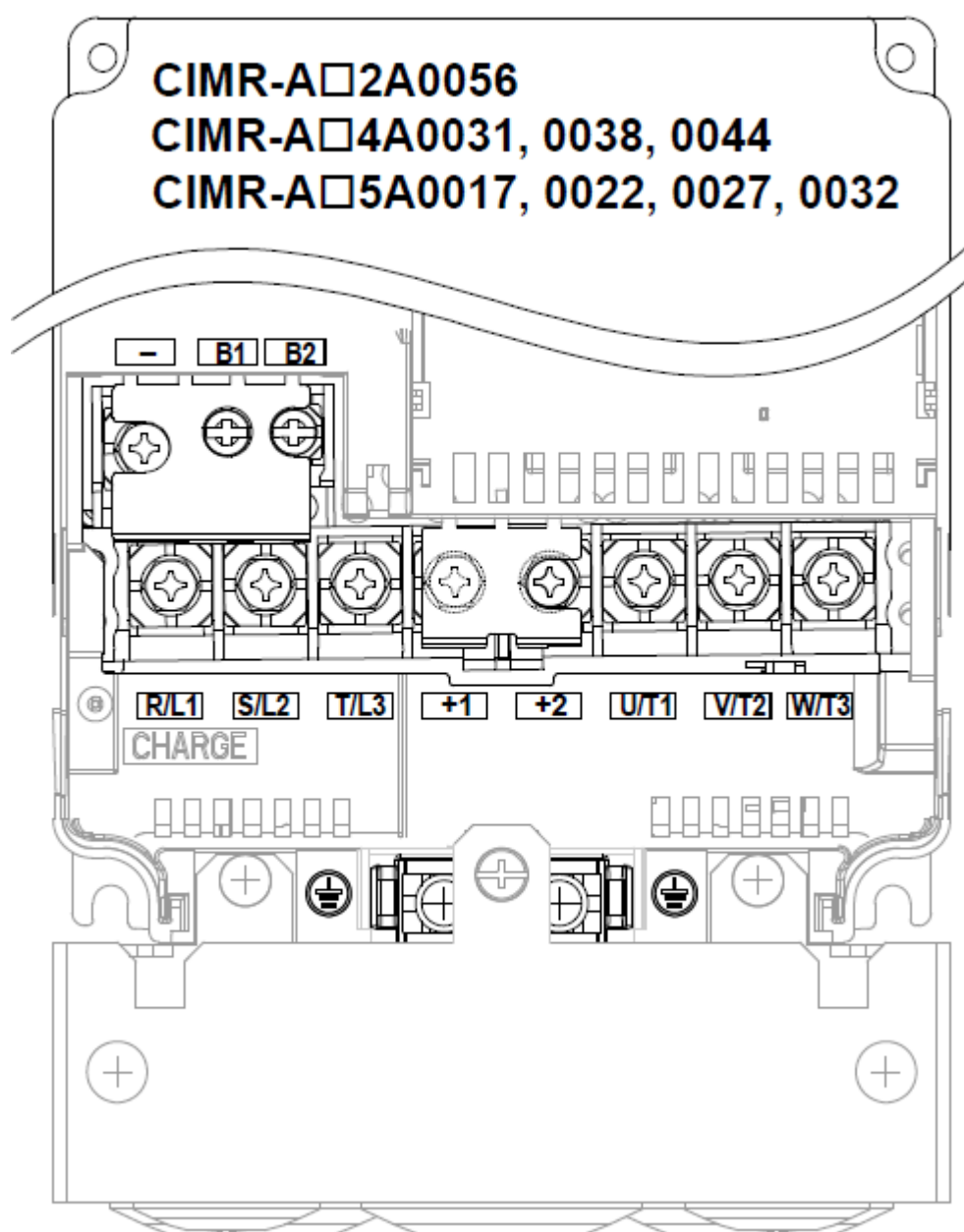


Figura 20: Desenho da localização dos bornes de potência do inversor [4].

Os terminais de controle utilizados foram os referentes à comunicação serial Modbus, são estes os terminais R+, R-, S+, S- e IG. Além disso, para que a rede RS-485 possa ser terminada é necessário alternar para o estado ligado a chave de terminação S2 localizada na placa de controle do inversor de frequência. A figura a seguir mostra a posição física dos terminais de controle.

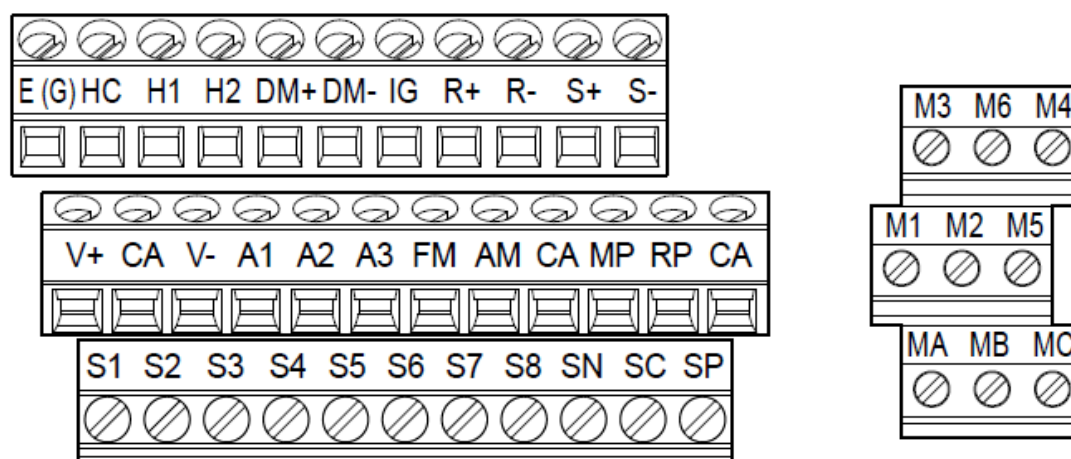


Figura 21: Desenho da localização dos bornes de controle do inversor [4].

4.1.3 Parametrização

Após realizar as conexões mencionadas no item anterior, o inversor de frequência pode ser energizado.

Com o mostrador do operador digital aceso é possível realizar as operações e parametrizações no equipamento. A figura a seguir ilustra todos os modos disponíveis do operador que podem ser vistos ao pressionar os botões de seta pra cima ou para baixo localizados embaixo do visor.











Modo	Conteúdo	Display do Operador
Inicialização	Referência de frequência (padrão)	
Modo de Operação		
	Display de Monitoração	
Modo Programação		
	Menu de Verificação	
	Grupo de Configuração	
	Modo de Configuração de Parâmetro	
	Modo de Autoajuste	
Modo de Operação	Referência de frequência	

Figura 22: Modos do operador digital [4].

Ao pressionar a tecla enter no menu de configuração de parâmetro, pode-se modificar os parâmetros do inversor de acordo com a aplicação. Além disso, no menu de autoajuste é possível ajustar o equipamento para operar com o motor com maior eficiência. O operador apresenta em sua tela todas as configurações de moto autoexplicativo, o que facilita sua

configuração. A Tabela 3 ilustra todos os grupos de parâmetros disponíveis para alteração pelo inversor de frequência:

Tabela 3: Grupos de parâmetros do inversor de frequência [4].

Grupo de Parâmetros	Nome	Grupo de Parâmetros	Nome
A1	Parâmetros de inicialização	H2 	Saídas digitais multifunções
A2	Parâmetros escolhidos pelo Usuário	H3 	Entradas analógicas multifunções
b1	Seleção do modo de operação	A4	Saídas Analógicas Multifuncionais
b2	Frenagem por injeção de CC e Frenagem por curto-circuito	H5	Comunicação serial MEMOBUS/Modbus
b3 	Busca rápida	H6	Entrada/saída do trem de pulsos
b4	Função do temporizador	L1 	Proteção do motor
b5	Controle de PID	L2	Ociosidade temporária por perda de energia
b6	Função de contato	L3 	Prevenção de estol
b7	Controle de droop	L4	Deteção de velocidade
b8	Economia de energia	L5	Reinício por falha
b9	Zero servo	L6	Deteção de torque
C1	Tempos de aceleração e desaceleração	L7	Limite de torque
C2	Características de curva em S	L8 	Proteção do inversor
C3 	Compensação de escorregamento	n1	Prevenção de oscilação
C4	Compensação de torque	n2	Ajuste do controle de detecção de realimentação de velocidade (AFR)
C5	Regulador automático de velocidade (ASR)	n3	Frenagem de alto escorregamento (HSB) e frenagem de excesso de excitação
C6 	Frequência portadora	n5	Controle de feed-forward
d1	Referência de frequência	n6	Ajuste on-line
d2	Limites superiores/inferiores de frequência	n8 	Ajuste do controle de motor PM
d3	Frequência de salto	o1	Seleção do visor digital do operador
d4	Manutenção de referência de frequência de função Aumentar/Diminuir 2	o2	Funções do teclado digital do operador
d5	controle de torque	o3	Função de Cópia
d6	Enfraquecimento de campo e Imposição de campo	o4	Configurações do monitor de manutenção
d7	Frequência de deslocamento	q	Parâmetros do DriveWorksEZ
E1	Padrão de V/f para motor 1	r	Parâmetros de conexão do DriveWorksEZ
E2 	Parâmetros do motor 1	T1	Autoajuste do Motor de Indução
E3	Padrão de V/f para motor 2	T2	Autoajuste do motor PM
E4 	Parâmetros do motor 2	T3	Ajuste de inércia e ASR
E5	Configurações do motor PM	U1 	Monitores com estado de operação
F1	Cartão de Controle de Velocidade de PG (PG-B3/PG-X3)	U2 	Rastreio de falha
F2	Cartão de entrada analógica (AI-A3)	U3	Histórico de falhas
F3	Cartão de entrada digital (DI-A3)	U4 	Monitores de manutenção
F4	Cartão analógico do monitor (AO-A3)	U5	Monitores PID
F5	Cartão digital de saída (DO-A3)	U6	Monitores com estado de operação
F6, F7	Cartão opcional de comunicação	U8	Monitores do DriveWorksEZ
H1	Entradas digitais programáveis		

Para o uso da comunicação serial Modbus é necessário alterar os parâmetros do grupo H5 no menu de configuração de parâmetros do equipamento. Os parâmetros de comunicação utilizados no projeto são:

- H5-01 = 1F (em hexadecimal) - Endereço do inversor de frequência;
- H5-02 = 3 (9600 bps) – Velocidade de comunicação;
- H5-03 = 1 (par) – Seleção da paridade de comunicação;
- H5-04 = 3 (apenas alarme) – Método de parada após erro de comunicação;
- H5-05 = 0 (desativado) – Seleção de detecção de falhas;
- H5-06 = 25 (milisegundos) – Tempo de espera da transmissão do inversor;
- H5-07 = 1 (ativado) – Seleção do controle RTS (Request-to-send);
- H5-09 = 2 (segundos) – Tempo de detecção de erro de comunicação;
- H5-10 = 0 (0,1 Volts) – Seleção de unidade para registro Modbus 0025H;
- H5-11 = 1 (sem necessidade de enter) – Seleção da função enter;
- H5-12 = 0 (Frente/Parar, Reverso/Parar) – Seleção do método de comando.

Além dos parâmetros mencionados acima, para a operação do motor é necessário que haja uma referência de frequência (velocidade) e um comando para rodar. Podem-se ajustar esses modos de operação para a comunicação serial através dos seguintes parâmetros no inversor de frequência:

- b1-01 = 2 (comunicação serial) – Seleção da referência de frequência;
- b1-02 = 2 (comunicação serial) – Seleção do comando rodar.

Após a alteração dos parâmetros mencionados acima, o inversor de frequência estará pronto para operar com a comunicação serial Modbus.

4.2 Rede para comunicação

4.2.1 Camada Física

A comunicação do projeto consiste em enviar sinais contendo comandos, através da interface desenvolvida, para o inversor de frequência que será responsável por controlar o motor. Para tal é necessário que o sinal enviado do computador seja entendido pelo inversor.

A porta USB é a interface responsável pela comunicação no computador, já no inversor é utilizada a interface RS485. Para que essas interfaces possam se comunicar foi utilizado um conversor USB/RS232/RS485/TTL. Segue abaixo sua fotografia e o esquemático da placa:

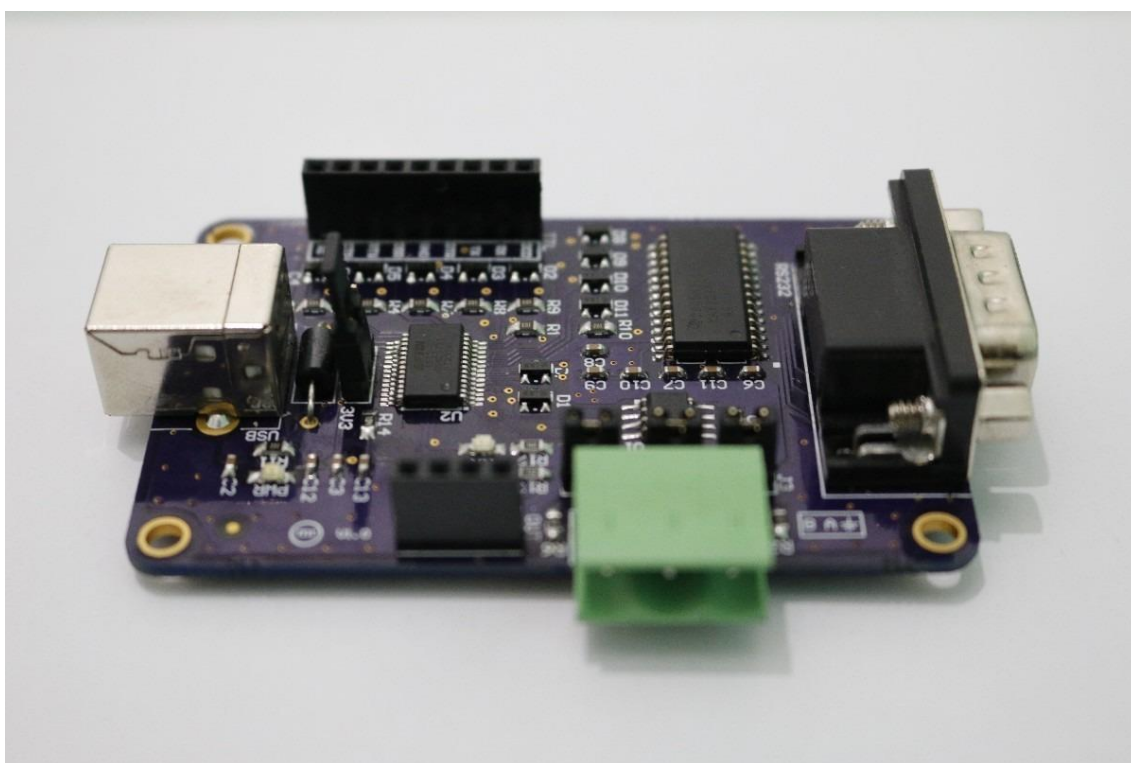


Figura 23: Conversor USB/TTL/RS232/RS485 [18].

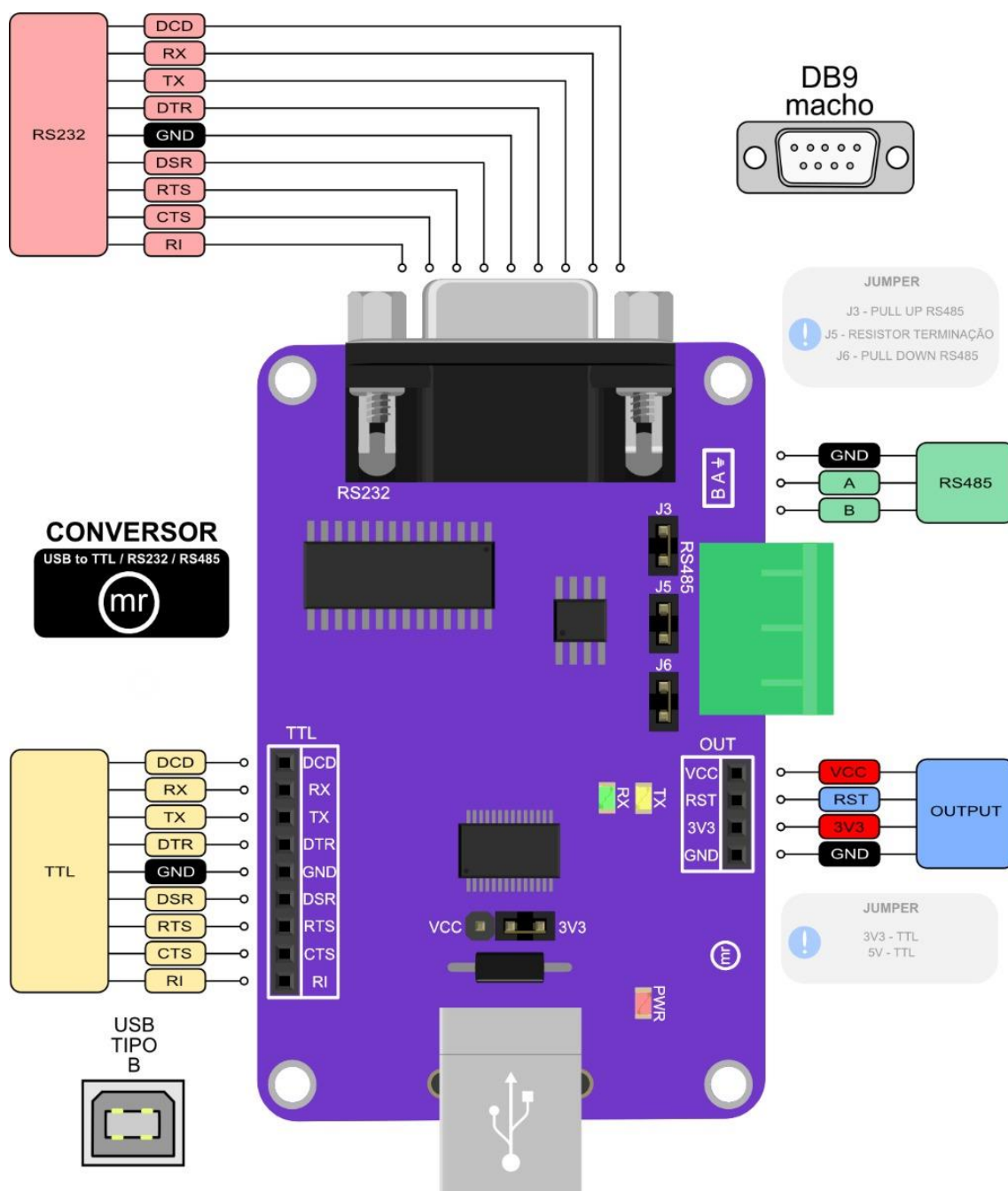


Figura 24: Diagrama do conversor USB/TTL/RS232/RS485 [18].

O padrão RS485 foi apresentado previamente, para a discussão do conversor é preciso se lembrar que o padrão EIA-485 é um padrão de transmissão de dados diferencial.

Um conector USB padrão é composto por quatro pinos: 1 pino de alimentação (Vcc), dois pinos para a transmissão do dado (D+ e D-) e um pino de aterramento (GND). O pino de alimentação é essencial para algumas aplicações onde o dispositivo não dispõe de uma fonte de alimentação própria.

Assim como o RS485, o USB também se utiliza do sinal diferencial para transmitir os dados, então a conversão do padrão depende então de ajustes nos níveis de tensão. Para transmissão em baixa ou alta velocidade o USB interpreta como nível lógico 1, quando:

$$D_+ \geq 2,8V \text{ e } D_- \leq 0,3V \quad (14)$$

Para nível lógico 0 deve ocorrer a inversão dos sinais, ou seja:

$$D_+ \leq 0,3V \text{ e } D_- \geq 2,8V \quad (15)$$

Quando se trata do receptor os níveis lógicos mudam conforme a velocidade da transmissão. É definido como nível lógico 1 quando:

$$D_+ > D_- + 200mV \quad (16)$$

E nível lógico 0 para:

$$D_- > D_+ + 200mV \quad (17)$$

4.2.2 Camada de Enlace

O conversor serve para garantir que os padrões elétricos enviados pelo computador possam ser entendidos pelo inversor, porém só a compatibilidade elétrica não é suficiente. Por isso a utilização do Modbus na camada de enlace.

O objetivo do Modbus é prover nível de abstração suficiente para que os dispositivos possam trocar mensagens. É a partir dele que um sinal diferencial irá ser interpretado como um comando parar, rodar, etc.

Uma das características do Modbus é a ligação entre a camada física e a camada de aplicação, modo mestre/escravo, através do Modbus sobre linha serial. E é essa característica

que permite a comunicação diretamente do EIA-485 e a parte funcional do inversor de frequência.

A camada de enlace é representada pelo Modbus sobre linha serial, e existem duas formas de se fazer a transmissão: a RTU (*Remote Terminal Unit*) e ASCII. No modo RTU os bytes são enviados de forma consecutiva e um intervalo maior que 3 caracteres e meio determina o fim da transmissão. Já no modo ASCII existe um start byte definido como 3A e dois bytes de parada, *Carriage Return* (CR), e *Line Feed* (LF) definidos como 0D e 0A, respectivamente.

Como consequência das características citadas, o modo RTU envia um pacote com número menor de dados se comparado a uma mesma mensagem no modo ASCII. Em contrapartida, o modo ASCII é mais confiável quando a mensagem precisa cruzar um caminho maior, já que o *delay* na mensagem não é um empecilho.

Pelo fato da distância entre o mestre e o escravo ser bastante pequena, menos de 2 metros, o modo escolhido é o RTU, pois existe a otimização na montagem da mensagem e eventuais.

4.3 Raspberry Pi

4.3.1 Definição

O Raspberry Pi é uma placa eletrônica de baixo custo do tamanho de um cartão de crédito criada com objetivo de ser um mini computador que ajuda pessoas no aprendizado da programação.

Com a maturação do produto ao longo do tempo, desenvolvedores perceberam o potencial dessa placa para aplicações com mídia, interfaces com entradas e saídas e aplicações com o uso de internet, por exemplo.

Além disso, os criadores da placa perceberam sua demanda e por isso, desenvolvem continuamente atualizações de *software* e *hardware*, além de acessórios como câmeras para integração.

A fundação Raspberry Pi é uma instituição de caridade educacional com sede no Reino Unido e tem como objetivo principal o desenvolvimento ferramentas tecnológicas para auxiliar crianças e adultos no estudo da computação. O nome Raspberry Pi foi escolhido porque Raspberry significa framboesa, em tradução direta do inglês e Pi, refere-se ao Python, a linguagem de programação. A referência à fruta se deve ao fato de outras empresas de grande porte utilizarem esta ideia como marca, por exemplo, a gigante Apple.

4.3.2 Características

Cronologicamente, foram desenvolvidas as placas Raspberry modelo A, B, B+ e A+, além disso, há um módulo industrial contendo apenas o microprocessador e uma memória tipo flash de 4 Gigabytes.

Para o presente projeto foi utilizado a placa Raspberry Pi modelo B+ por ter o melhor *hardware* dentre as outras. Abaixo segue sua fotografia:

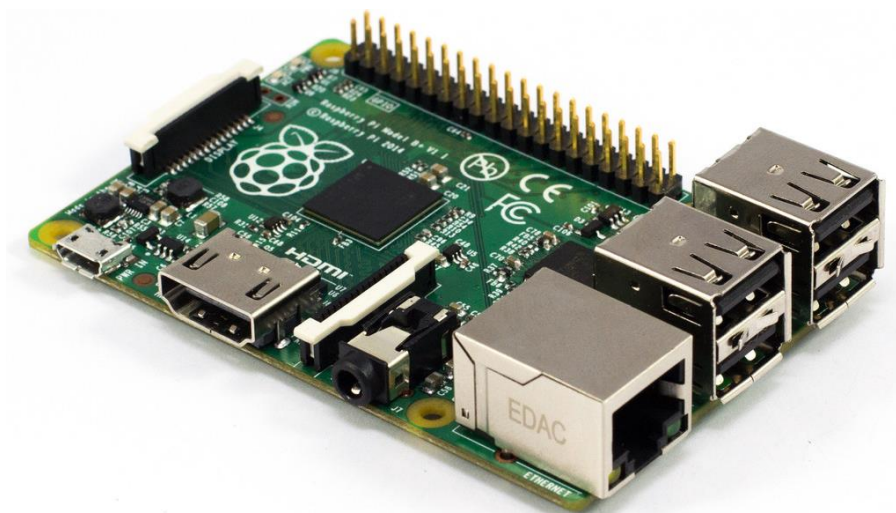


Figura 25: Fotografia da placa Raspberry Pi modelo B+.

Suas principais características são:

- CPU Broadcom BCM2835 (ARM1176JZ-F);
- Memória RAM de 512 Megabytes;
- 4 Portas USB 2.0;
- Entrada para cartão de memória micro SD;
- Alimentação de 5 Volts através de entrada micro USB.
- 40 Pinos de entrada e saída para comunicação SPI, I2C e UART, pinos de entrada de saída de propósito geral (GPIO) e saída PWM (Pulse Width Modulation).

Com todas as características mencionadas acima, pode-se notar que a placa Raspberry Pi é um microcomputador com potencial para a programação do nível mais baixo, utilizando programação em Assembly, até o nível mais alto, com recursos de processamento de dados multimídia.

As figuras abaixo mostram o esquema mecânico e o esquema do conector de 40 pinos placa:

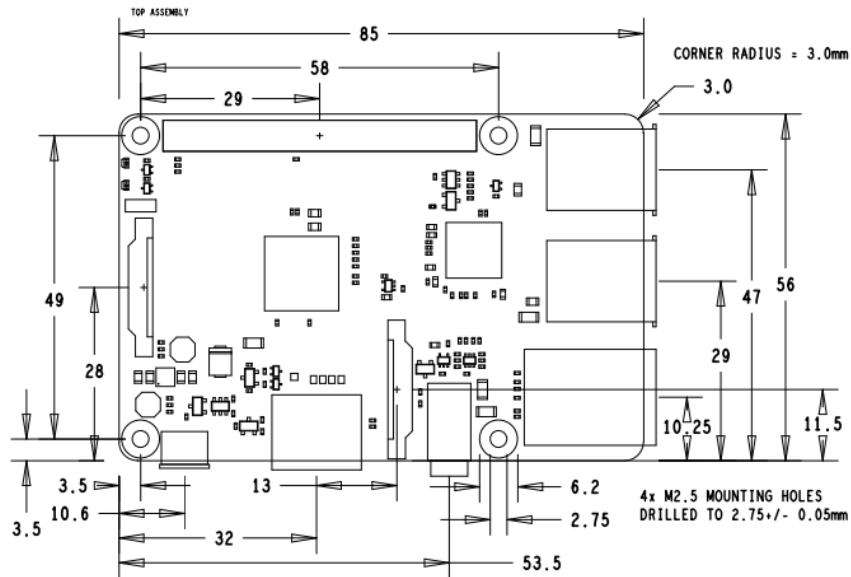


Figura 26: Desenho mecânico da placa Raspberry Pi modelo B+ [14].

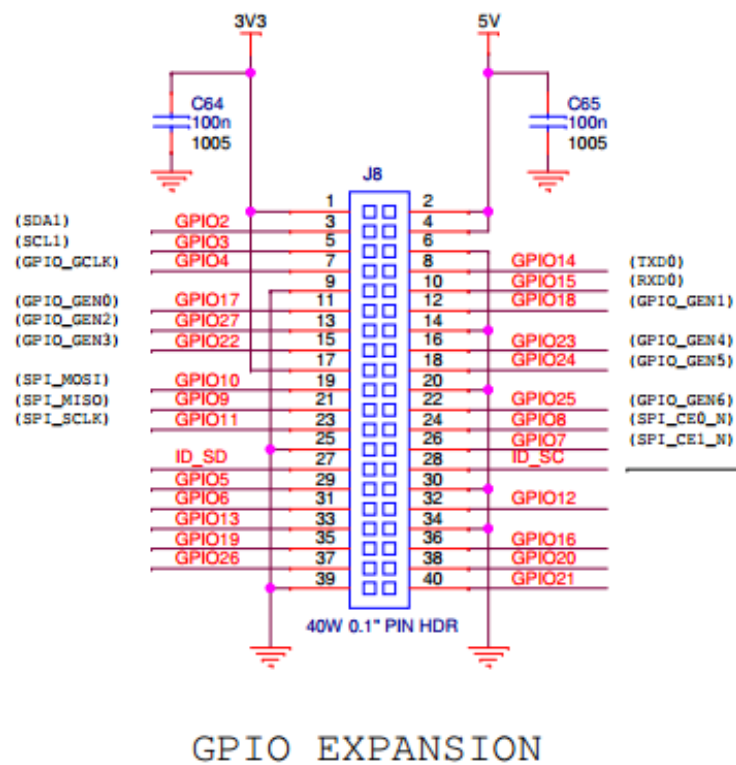


Figura 27: Diagrama da barra de pinos da placa Raspberry Pi modelo B+ [14].

4.3.3 Alimentação e conexão

Como o presente projeto se baseia na comunicação entre a placa Raspberry Pi e o inversor de frequência, utilizou-se um fonte de alimentação de 5 Volts conectada a entrada

micro USB; a saída de vídeo HDMI foi conectada a um conversor HDMI para VGA para que a imagem da interface possa ser projetada em um monitor, esta conversão foi importante, pois a maioria dos monitores no mercado utilizam a conexão VGA; e uma porta USB que foi utilizada para comunicação serial através de uma placa conversora entre o padrão USB e o RS-485. Além disso, utilizou-se uma entrada de teclado numérico USB para a interface com o usuário.

4.3.4 Instalação do Sistema Operacional

Há muitas possibilidades de programação com a placa Raspberry Pi, inclusive a de se trabalhar diretamente em seu microprocessador através da linguagem assembly.

Como a ideia de comunicação e interface pressupõe, além do processamento de dados, uma interface gráfica, utilizou-se no projeto o sistema operacional padrão da placa, o Raspbian.

Este Sistema Operacional é baseado na versão do Linux conhecida como Debian, modificado especialmente para seu uso em conjunto com a placa. Uma das principais vantagens deste sistema é, por exemplo, a possibilidade de programação de comunicação, entradas e saídas através do terminal (shell) do Linux.

Para a instalação do Raspbian em um cartão de memória que foi posteriormente inserido na placa, foram necessários os seguintes procedimentos.

1. Formatar e preparar o cartão de memória para utilizar o Raspberry Pi, o que pode ser feito através do software SD Formatter 4.0;
2. Baixar o arquivo NOOBS a partir do site oficial do Raspberry Pi (<http://www.raspberrypi.org>). Este arquivo permite a instalação do sistema Raspbian;
3. Copiar o arquivo NOOBS para o cartão de memória;
4. Inserir o cartão de memória no Raspberry Pi.

A partir deste ponto, pode-se conectar e ligar a fonte de alimentação à placa Raspberry Pi conectando também um mouse e um teclado às portas USB para a instalação e configuração inicial do sistema.

Com o sistema inicializado, seleciona-se a opção “Raspbian” e em seguida “Install”. Depois da instalação concluída, podem-se efetuar as principais configurações, caso seja necessário, a partir do comando “raspi-config” no terminal.

Para iniciar no modo gráfico do sistema, é necessário entrar no terminal com o usuário “pi” e senha “raspberry” logo em sua inicialização. Após isso, o comando “startx” leva o usuário à tela inicial do sistema, conforme ilustrado na figura abaixo:

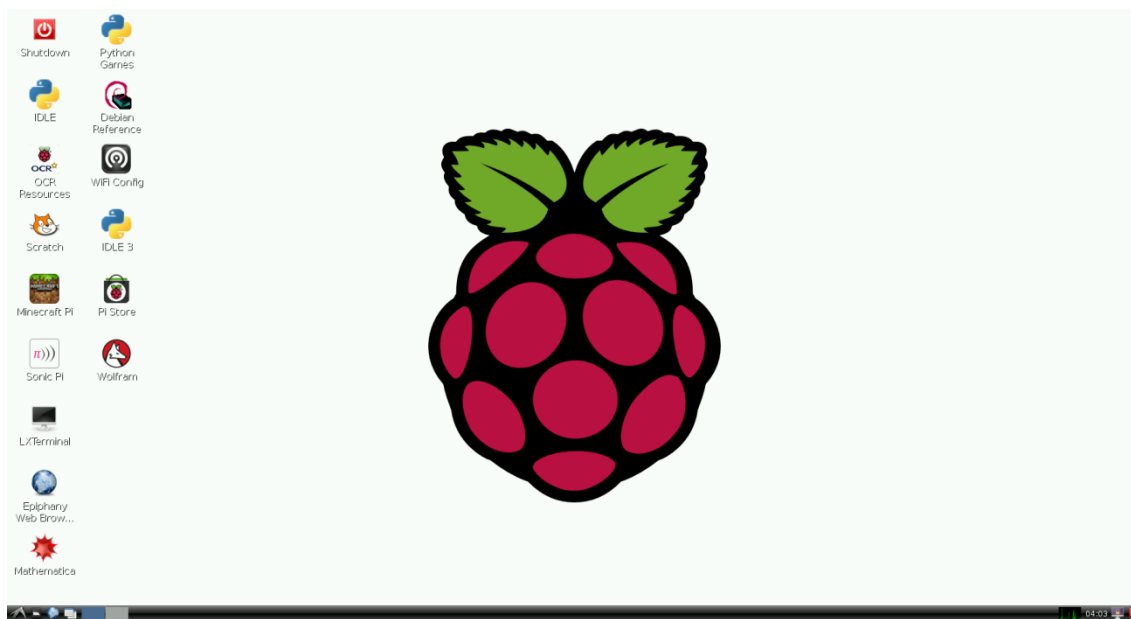


Figura 28: Imagem da tela inicial do Sistema Operacional Raspbian

4.3.5 Modo embarcado

Para utilizar a placa Raspberry Pi de modo a executar a aplicação desenvolvida no início do sistema é necessário criar um arquivo com a extensão .desktop na pasta autostart dentro da pasta config na pasta pessoal do Raspbian.

O arquivo com_vdf.desktop contem a seguinte configuração:

[Desktop Entry]

Encoding=UTF-8

Name=com_vfd

Exec=sudo /home/pi/com_vfd-build-desktop-Qt_4_8_2__System__Debug/com_vfd

4.4 QT

O Qt é uma software para desenvolvimento de interfaces gráficas multiplataformas, atendendo hoje em dia a todos os sistemas mais conhecidos, Windows, Android e iOS, além de ser utilizado no desenvolvimento de sistemas embarcados.

O software foi idealizado por Haavard Nord e Eirik Chambe-Eng em 1991, atuais CEO e presidente, respectivamente, da Trolltech. Mesma empresa que foi responsável pelo Qt durante muito tempo, até que a Nokia adquirisse a empresa em 2008 e o utilizasse para desenvolver seu sistema operacional, na época o Symbian.

A Nokia ficou responsável pelo software durante 3 anos, aproximadamente, até que vendesse a sua licença comercial a Digia em 2012, que é responsável pelo desenvolvimento até os dias de hoje.

4.4.1 Características

A sua principal característica é a possibilidade de um código pensado para um determinado sistema, Windows por exemplo, possa ser utilizado em outros sistemas com poucos mudanças no seu código fonte, ou mesmo nenhuma alteração.

Essa possibilidade de plataformas cruzadas faz com que o Qt seja bastante usado por desenvolvedores independentes contribuindo para a criação de uma comunidade relativamente grande e um acervo considerável de soluções já pensadas e executadas.

Existe também um grande atrativo as pessoas que desejam iniciar a trabalhar com a plataforma que é a disponibilidade de documentação bem detalhada das bibliotecas disponíveis. Além da documentação também é possível a utilização de exemplos, que já são previamente instalados junto com o software, com aplicações práticas, como terminais, e outras mais simples, como a criação de janelas.

Aliado as características acima existe a possibilidade de se criar a interface usando método gráfico, que é mais simples mesmo para usuários avançados, ou pode-se utilizar programação em XML. É possível também a utilização de ambos para corrigir e alterar pequenos detalhes como cores e tamanho dos itens dispostos na interface.

4.4.2 Instalação em modo nativo

Existem duas formas de se utilizar o Qt com o Raspberry Pi, instalando-o em modo nativo ou utilização da compilação cruzada. Como o software demanda um hardware de alta capacidade, o ideal é a utilização da compilação cruzada. Porém é necessária a instalação de uma grande quantidade de repositórios Linux e alguns deles não estão mais disponíveis, dificultando assim sua utilização.

A solução utilizada então foi a instalação em modo nativo, que apesar de ser mais lenta poupa bastante tempo, pois não é necessário a criação dos repositórios indisponíveis. O processo de instalação passo a passo é o seguinte:

1. Baixar e instalar o kit de desenvolvimento. Usa-se o comando `sudo apt-get install qt4-dev-tools`
2. Instalar o Qt Creator, ferramenta responsável pela criação do aplicativo através do comando `sudo apt-get install qtcreator`
3. Baixar repositórios auxiliares `gcc`, `xterm`, `git-core` e `subversions`. Todos baixados usando `sudo apt-get install "nome do repositório"`

O programa a partir de momento está instalado, mas ainda não é suficiente para funcionar de forma adequada, pois ainda falta a *toolchain* para poder compilar o projeto no Raspberry Pi. Para adiciona-la o Qt Creator deve ser aberto e então acessar o seguinte caminho: “Tools/Options > build & run > tab tool chains > button add > GCC”.

Selecionada a *toolchain* as localizações do compilador e *debugger* devem ser apontadas. O compilador encontra-se em “/usr/bin/arm-linux-gnueabi-hf-gcc-4.6” enquanto o *debugger* está em “/usr/bin/gdb”. Deixar `mkspec` em default.

Terminadas a configurações deve-se então acessar o menu “Help > About plug-ins” e desmarcar a opção “device support > remote Linux”. Reiniciar o Qt Creator e adicionar “/usr/bin/qmake-qt4” em “Tools > Options > Build & Run > Qt Versions”. Esse último procedimento é para que o programa não procure por um dispositivo remoto para compilar o projeto, afinal está sendo utilizado o modo nativo.

4.4.3 Bibliotecas e Programação

O Qt dispõe de um grande número de bibliotecas disponíveis, porém uma das bibliotecas essenciais para o desenvolvimento desse projeto não está disponível

imediatamente após a instalação do software, sendo necessário adicioná-la separadamente. Essa biblioteca é responsável por tornar acessível funções básicas da porta serial.

Segue abaixo os passos necessários para a instalação da biblioteca:

1. Abrir um terminal e fazer uma cópia da biblioteca através do comando `sudo git clone git://gitorious.org/qt/qtserialport.git`
2. Acessar a pasta copiada usando `cd qtserialport`
3. Instalar a biblioteca (`sudo qmake > sudo make > sudo make install`)
4. Abrir o programa através do terminal com `sudo qtcreator`
5. Ir em Tools > Options > Build & Run > General > Projects Directory > Directory > home/pi
6. Em Qt Versions selecionar a versão marcada por um aviso ir em browse e selecionar qmake
7. Na guia Toolchain clicar adicionar e selecionar gcc
8. Compiler path: /usr/bin/gcc
9. Debugger: /usr/bin/gdb
10. Verificar em “Help > About plugins” se a opção remote Linux continua desmarcada.
11. Reiniciar o Qt

A biblioteca está agora instalada, para utiliza-la basta adicionar ao programa a linha de código “CONFIG += serialport”, caso uma versão 5.x esteja sendo usada o comando é “QT += serialport”.

As demais bibliotecas usadas estão divididas no pacote CORE, GUI e LABEL, mais especificamente QKeyEvent, responsável pela leitura de comandos do teclado, QTimer, usado na implementação do timeout da transmissão, e QIODevice, usado para leitura e escrita de dados nas portas de entrada e saída.

A programação no Qt é feita em C++, uma linguagem orientada ao objeto, ou QML que é uma meta linguagem específica para a ferramenta. Outras linguagens também podem ser utilizadas, contando que sejam utilizadas as respectivas bibliotecas para abstração do código, por exemplo PyQt usado para associação à linguagem Python.

4.5 Interface do usuário

A interface do usuário foi desenvolvida utilizando o Qt. Toda a parte gráfica foi arrumada usando macroblocos disponíveis na ferramentas, já as associações e funções de cada um desses blocos foi apontada posteriormente no código fonte.

Essa interface é constituída de um teclado que servirá como fonte da entrada de dados e um tela que mostra o estado atual do sistema bem como mensagens de sucesso ou insucesso no completamento de um pedido.

4.5.1 Entrada de Dados

Para entrada de dados será utilizado um teclado numérico, uma vez que a não é necessário um número muito grande de teclas para executar todas as ações possíveis do sistema. O teclado executará as funções de selecionar determinada ação, rodar, parar, teste de loop ou alteração de frequência, além de selecionar o novo valor de frequência.

A placa Raspberry Pi utilizada para a realização de todo o processamento dos dados possui uma entrada USB que detecta a entrada de dados através de teclado automaticamente (dispositivo *plug and play*).



Figura 29: Fotografia do teclado numérico utilizado no projeto.

As funções de cada botão no sistema são:

Seta pra cima => +

Seta para baixo => -

Enter => Enter

Esc => del

Leitura de frequência => 5

Sair do programa => Seta para esquerda (backspace)

Endereço 1 => 0

Endereço 2 => 1

4.5.2 Tela



Figura 30: Imagem da tela do sistema de interface e comunicação.

A Figura 28 apresenta a tela da interface do usuário. Essa interface dispõe as ações executadas pelo sistema à esquerda. Ao centro-direita são mostradas as informações referentes ao *set point* da frequência e a real frequência do motor. O canto inferior direito são mostradas os frames do MODBUS enviados pelo Raspberry Pi (*Buffer TX*) e os frames enviados pelo inversor de frequência (*Buffer RX*).

Ainda no canto inferior direito estão informações sobre o status da comunicação, sucesso ou insucesso, e qual o estado atual da máquina. Esse estado é referente a máquina de estado utilizada no código fonte, e serve como referência para identificar a execução correta do processo.

Voltando ao status da comunicação, é nesse espaço que serão especificadas qualquer falha na comunicação. Como explicado no item 2.3.4.3, erros de comunicação, diversas falhas podem ocorrer, por isso a mensagem enviada pelo inversor é processada e um eventual error code é então identificado e mostrado ao operador.

No topo da tela são exibidas as informações sobre o projeto, autores, título, professor orientador e instituição de ensino.

Capítulo 5

Código-fonte

5.1 Visão geral

O código-fonte do projeto foi escrito na linguagem de programação C/C++ utilizando as bibliotecas disponíveis pela cadeia de ferramentas do Qt. Este software foi executado em modo nativo na placa Raspberry Pi para compilar, depurar e executar a comunicação serial e a interface gráfica com o inversor de frequência.

Conforme mencionado no item 4.5, a interface de usuário foi desenvolvida em uma tela colorida com auxílio das ferramentas do Qt Designer. Para a entrada de dados através do teclado numérico USB, utilizou-se a biblioteca QKeyEvent. Para efetuar a comunicação, utilizou-se a biblioteca QSerialPort.

Além disso, para trabalhar com os recursos disponíveis na placa Raspberry Pi, neste caso, as saídas digitais, foi necessário a utilização da biblioteca bcm2835 criado por Mike MacCaley. [19]

O projeto de software completo conta com o programa de inicialização através do arquivo main.c, além disso para a inicialização das classes da linguagem C++ bem como a escrita do código fonte principal, criou-se, respectivamente, os arquivos mainwindow.h e mainwindow.cpp.

A interface gráfica é gerada pelo arquivo xml mainwindow.ui criado pelo próprio Qt. O arquivo resource.qrc permite a importação da imagem logotipo do CEFET/RJ utilizada na interface, bem como o arquivo cefet.jpg que contém a imagem a ser importada.

Além das bibliotecas mencionadas bcm2835.c e bcm2835.h, para que o projeto possa ser efetivamente compilado, é necessário que o arquivo com_vdf.pro esteja configurado com todas informações de compilação para o Qt. Todo o código do programa principal mainwindow.cpp pode ser visto em anexo.

5.2 Entrada de dados

A entrada de dados pelo teclado numérico USB é inicializada pela biblioteca QKeyEvent e as variáveis de sinalização de cada botão pressionado são ajustados para o valor 1 toda vez que um botão é pressionado.

A função keyPressEvent é inicializada toda vez que ocorre um evento de mudança de estados em algum botão do teclado. Esta inicialização ocorre através de uma espécie de interrupção muito utilizada pelo Qt e que o torna diferente em relação aos outros programas.

Esta interrupção ocorre com emissão de sinais correspondente a mudanças de estados conhecidos como signals e a recepção da mudança de estados através de uma função, por exemplo, chama-se slot. Então a interrupção via software é conhecida como signal/slot. Neste caso, a função keyPressEvent é um slot para o evento de mudança de estado de teclas.

5.3 Menu

O menu presente na parte esquerda da tela mostra todas as funções disponíveis do programa: rodar, parar, ler frequência, alterar frequência e teste de loop.

Todo gerenciamento da entrada de dados e visualização das respostas na tela foram criados utilizando uma máquina de estados robusta que descreve todo o processo. Esta máquina pode ser vista na figura a seguir.

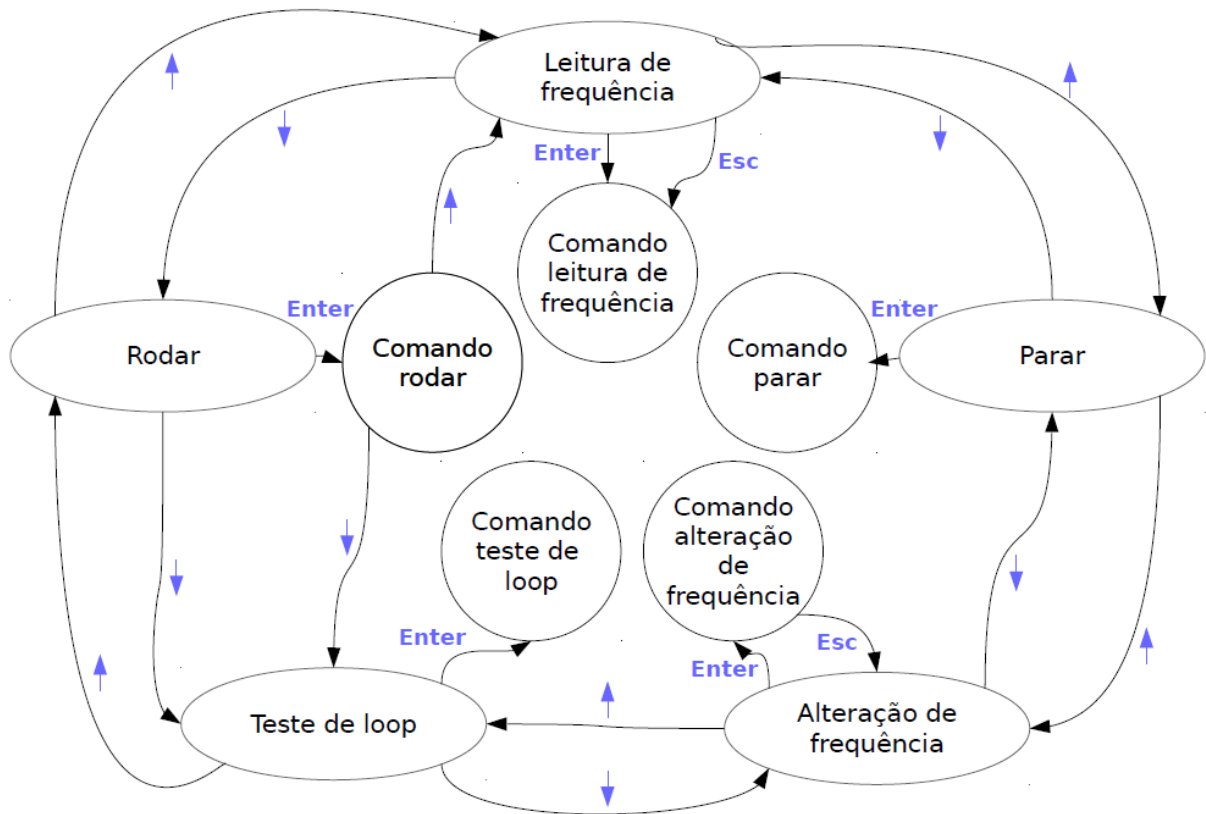


Figura 31: Máquina de estados do menu do sistema.

5.4 Leitura e escrita no inversor de frequência

O bloco de leitura e escrita no inversor de frequência conta com as informações de referência de frequência dado pelo usuário através do comando alterar frequência no menu à esquerda e a frequência de saída que pode ser vista com o comando saída de frequência também neste mesmo menu.

A informação da frequência enviada ao motor, a frequência de saída, está disponível no pacote de dados Modbus recebido através da variável `f_saida`.

A alteração de frequência é atualizada na interface do programa com o incremento ou decremento do código na função de alterar frequência de acordo com a máquina de estados. A variável correspondente para alterar a frequência é chamada de `r_freq`.

5.5 Dados do programa

O bloco dados de programa mostra a máquina de estados do menu e a posição do estado atual. Além disso, há o buffer correspondente ao envio e recebimento das mensagens de acordo com o protocolo Modbus. As mensagens são enviadas e no seu recebimento ou na falta de recebimento, pode-se visualizar o estado da comunicação através da barra de comunicação. Em caso de sucesso, uma tarja na cor verde aparece na tela, caso haja problema no tempo de resposta da mensagem ou inversor de frequência desconectado a barra de comunicação ficará na cor vermelha e se houver algum tipo de erro, como no cálculo do CRC a barra estará na cor roxa.

Os buffers de envio e recebimento da mensagem são criados a partir dos vetores `buffer_envia` e `buffer_recebe`. Seus tamanhos variam de acordo com a mensagem a ser enviada ou recebida.

5.6 Envio da mensagem

O envio das mensagens é realizado de acordo com cada comando dado através das teclas de entrada de dados de acordo com a posição da máquina vista no menu à esquerda da interface.

No envio das mensagens há uma variável de sinalização utilizada para acionar as funções de envio e processamento de cada formato da mensagem para transformá-la ao protocolo Modbus. Nestas funções são também feitos os cálculos de CRC para checagem de erros.

5.7 Recebimento da mensagem

O recebimento das mensagens é realizada pela função tipo slot criada com o nome de `serialRecebido`. Nesta função deve ser verificada a variável de sinalização da mensagem enviada para saber qual o tipo de mensagem foi recebida e o seu tamanho. A partir daí as informações são separadas e processadas, bem como o cálculo de checagem de erros e em caso de sucesso ou insucesso são acionadas as sinalizações correspondentes que atuam na interface para responder ao usuário.

Capítulo 6

Campo de aplicação

A comunicação e interface entre o inversor de frequência e um computador permite ampliar a interação e o controle de processos em muitas aplicações que envolvem o motor de indução.

Conforme mencionado no tópico 2.1, sobre motores elétricos, as aplicações dos motores de indução se estendem por todos os segmentos da indústria, movimentação de cargas e outras áreas, como em lavanderias.

Através deste sistema de comunicação é possível interagir com até 255 motores de indução ligados à inversores de frequência em uma mesma rede. Isso viabiliza aplicações que envolvem a monitoração dinâmica, a geração de estatísticas gerenciais e a atuação diretamente sobre o processo para sua otimização.

Além disso, em ocasiões como no mercado das lavanderias, por exemplo, além de monitorar é possível processar as informações de sensores e atuadores envolvidos para automatizar todo o processo com interface gráfica e armazenamento de receitas que controlam os produtos químicos utilizados.

A generalização do conceito de interface, processamento e monitoração em indústrias torna o sistema apto a atuar em qualquer segmento, inclusive no controle de equipamento à distância, podendo-se utilizar protocolos industriais ou até mesmo armazenar e transferir dados através da internet pela placa Raspberry Pi.

Em aplicações onde há muitas máquinas é também possível, controlar o consumo de energia, monitorar o estado da máquina para reconhecer falhas à distância e atuar sobre o motor. Estas aplicações são possíveis em shopping centers, transportes ferroviários, aeroportos, com máquinas aplicadas às escadas rolantes, elevadores e refrigeração.

Capítulo 7

Conclusão

O presente projeto teve como objetivo o desenvolvimento de uma interface e comunicação com inversor de frequência utilizando o protocolo de comunicação serial MODBUS. Para tanto, criou-se um sistema eletrônico para processar, enviar e receber todas as informações referentes à troca de mensagens entre os dispositivos.

Desde o início do projeto, mesmo com a ideia bem definida, a escolha do hardware foi um fator importante para o sucesso do sistema. A placa Raspberry Pi viabiliza uma tendência comportamental de desenvolvedores que consiste em utilizar o tempo de projeto para produção de um software robusto, partindo do princípio da escolha de um hardware geral e completo.

A acessibilidade à tecnologia e à cadeia de ferramentas, incluindo bibliotecas e documentação, foi fundamental para o desenvolvimento do código-fonte com mais de 1200 linhas no arquivo principal.

Os testes realizados com o inversor de frequência apresentaram sucesso devido ao planejamento das rotinas, máquinas de estado e algoritmos desenvolvidos. A ideia, muito utilizada na engenharia, de dividir problemas em blocos facilitou a escrita e testes de códigos complexos que abrangem conhecimentos de hardware e de software.

A engenharia de solução de problemas, especificamente a engenharia eletrônica, busca o desenvolvimento de projetos tanto na parte de escrita de códigos quanto da parte de viabilidade de componentes, máquinas, placas eletrônicas e equipamentos. Neste ponto, o presente trabalho se mostra eficiente em criar um sistema que facilita o processamento de informações e oferece uma interface moderna em relação às interfaces tradicionais utilizadas no mercado.

Além disso, como já mencionado no item 7, o campo de aplicação deste sistema se estende a todas as áreas da indústria e transporte que utilizam o motor elétrico para acionar máquinas para as mais variadas finalidades.

Baseado nos valores fundamentais utilizados pelos profissionais da engenharia em criar projetos que proporcionam o desenvolvimento tecnológico, este sistema serve como base para facilitar o gerenciamento de processos e o processamento de dados em aplicações industriais.

Sua evolução pode servir à integração de uma rede com mais equipamentos em comunicação sem fios entre eles para monitorar e processar dados. Isso torna o sistema ainda mais completo e moderno seguindo tendências mundiais em relação à servidores capazes de monitorar e controlar todos os dispositivos eletrônicos abrangidos pela rede. Este conceito é conhecido atualmente como internet das coisas.

APÊNDICE A: Código-fonte – mainwindow.cpp

```
// *****
// SISTEMA DE INTERFACE E COMUNICACAO COM INVERSORES DE FREQUENCIA
// *****
//
// Descricao: Este programa permite a comunicacao com um inversor de
// frequencia utilizando a comunicacao serial atraves da porta USB.
// A entrada de dados atraves de um teclado numerico USB permite a
// interacao do usuario com a interface grafica e com o processamento
// de dados que efetua a comunicacao com o inversor de frequencia.
//
// Autores: Atila Santos e Jose Alvarenga
// Versao: 1.0
// Data: Novembro/2014
//
// *****

// Inicializacao de bibliotecas
// -----
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <QKeyEvent>
#include <QDebug>
#include <QTimer>
#include <QIODevice>
#include <bcm2835.h>

// Inicializacao de constantes
// -----
#define TIMEOUT 500 // Definicao do tempo para aguardar
// resposta em milisegundos.

#define EST_RODAR 0 // Estados da maquina (menu).
#define EST_LEITURA 1
#define EST_PARAR 2
#define EST_ALTERA 3
#define EST_LOOP 4
#define EST_COM_RODAR 5
#define EST_COM_LEITURA 6
#define EST_COM_PARAR 7
#define EST_COM_ALTERA 8
#define EST_COM_LOOP 9

#define ENDERECO1 0x1f // Define enderecos
#define ENDERECO2 0x20

// Inicializaçãõ de variaveis
// -----
int flag_bt_entra = 0; // Flag das teclas
int flag_bt_esc = 0;
int flag_bt_sobe = 0;
int flag_bt_desce = 0;

int flag_rodar = 0; // Flag para acionamento de funcoes
int flag_leitura = 0;
int flag_parar = 0;
int flag_altera = 0;
int flag_loop = 0;

int flag_leitura_loop = 0; // Flag auxiliar para leitura de frequencia
```

```

int r_freq = 0;           // Referencia de frequencia (Modbus)
float s_freq = 0;         // Referencia de frequencia em escala(display)

int estados               = 0;      // Estados da maquina

char buffer_envia[13];     // Buffers
char buffer_recebe[13];
char buffer_crc[2];
char buffer_recebido[13];

int flag_timeout = 0;      // Flag para timeout

int endereco = ENDERECO1;    // Variavel de endereco

// *****
// FUNCAO PRINCIPAL
// *****

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);

    // Zera variaveis:
    // -----

    dado_serial = 0;
    tamanho_serial = 0;
    leitura = 0;
    flag_rodar_rx = 0;
    flag_parar_rx = 0;
    flag_altera_rx = 0;
    flag_leitura_rx = 0;
    flag_loop_rx = 0;
    tamanho_buffer_envia = 0;
    tamanho_buffer_recebe = 0;
    f_saida = 0;
    barra_com_ui="Inicio";
    barra_cor = "background-color: rgb(0,0,100);";

    // Inicializacao da biblioteca do Raspberry Pi:
    // -----
    bcm2835_init();

    // Inicializacao da porta serial USB:
    // -----
    serial = new QSerialPort(this);

    serial->setPortName("ttyUSB0");
    serial->setBaudRate(QSerialPort::Baud9600);
    serial->setDataBits(QSerialPort::Data8);
    serial->setParity(QSerialPort::NoParity);
    serial->setStopBits(QSerialPort::OneStop);
    serial->setFlowControl(QSerialPort::NoFlowControl);
    serial->open(QIODevice::ReadWrite);

    // Inicializacao do LED de inicio do programa:
    // -----
    bcm2835_gpio_fsel(RPI_BPLUS_GPIO_J8_11, BCM2835_GPIO_FSEL_OUTP);
    bcm2835_gpio_set_pud(RPI_BPLUS_GPIO_J8_11, BCM2835_GPIO_PUD_DOWN);
    bcm2835_gpio_write(RPI_BPLUS_GPIO_J8_11, HIGH);

    // Inicio dos temporizadores
    // -----

```

```

// Temporizador de loop do programa
timer = new QTimer(this);
connect(timer, SIGNAL(timeout()), this, SLOT(maquina_estado()));
timer->start(0);

// Temporizador de timeout
timeout_modbus = new QTimer(this);
connect(timeout_modbus, SIGNAL(timeout()), this, SLOT(timeout_alert()));

// Acionamento da funcao serialRecebido (SLOT)
// -----
connect(serial, SIGNAL(readyRead()), this, SLOT(serialRecebido()));

}

// *****
// FUNCAO PARA O FIM DO PROGRAMA
// *****

MainWindow::~MainWindow()
{
    bcm2835_gpio_write(RPI_BPLUS_GPIO_J8_11, LOW);
    delete ui;
}

// *****
// FUNCAO PARA MUDANCA DE ESTADOS NAS TECLAS
// *****

void MainWindow::keyPressEvent(QKeyEvent *k) {

    if (k->key() == Qt::Key_Minus) { flag_bt_sobe = 1; flag_bt_desce = 0;
flag_bt_esc = 0; flag_bt_entra = 0; } else flag_bt_sobe = 0;
    if (k->key() == Qt::Key_Plus) { flag_bt_sobe = 0; flag_bt_desce = 1;
flag_bt_esc = 0; flag_bt_entra = 0; } else flag_bt_desce = 0;
    if (k->key() == Qt::Key_Comma) { flag_bt_sobe = 0; flag_bt_desce = 0;
flag_bt_esc = 1; flag_bt_entra = 0; } else flag_bt_esc = 0;
    if (k->key() == Qt::Key_Enter) { flag_bt_sobe = 0; flag_bt_desce = 0;
flag_bt_esc = 0; flag_bt_entra = 1; } else flag_bt_entra = 0;
    if (k->key() == Qt::Key_Backspace) { QApplication::quit(); }
    if (k->key() == Qt::Key_5) { estados = EST_COM_LEITURA; }
    if (k->key() == Qt::Key_0) { endereco = ENDERECO1; }
    if (k->key() == Qt::Key_1) { endereco = ENDERECO2; }

}

// *****
// FUNCAO PARA CALCULO DO CRC
// *****

void MainWindow::crc(char *buf, int bufLen, char *crc) {

    unsigned long crc_0 = 0xffff;
    unsigned long crc_1 = 0x0000;
    int i,j;
    for (i=0;i<bufLen;i++) {
        crc_0 ^= ((unsigned long)buf[i] & 0x0ff);
        for (j=0;j<8;j++) {
            crc_1 = (crc_0>>1) & 0x7fff;
            if (crc_0 & 0x0001)
                crc_0 = (crc_1) ^ 0xa001;
            else
                crc_0 = crc_1;
        }
    }
    crc[0] = 0x00;
    crc[1] = 0x00;
}

```

```

    crc[0] = (unsigned char)((crc_0/256) & 0x00ff);
    crc[1] = (unsigned char)((crc_0 & 0x00ff));
    return;
}

// *****
// FUNCAO DE LOOP PRINCIPAL
// *****

void MainWindow::maquina_estado() {

    // *****
    // MÁQUINA DE ESTADOS
    // *****

    switch (estados) {

        case EST_RODAR:

            // Operacao:
            // -----
            ui->rodar->setStyleSheet("background-color: rgb(0, 136, 100); color:
rgb(255, 255, 255);");
            ui->leitura->setStyleSheet("background-color: none; color: rgb(255,
255, 255);");
            ui->parar->setStyleSheet("background-color: none; color: rgb(255, 255,
255);");
            ui->altera->setStyleSheet("background-color: none; color: rgb(255, 255,
255);");
            ui->loop->setStyleSheet("background-color: none; color: rgb(255, 255,
255);");

            // Mudanca de estado:
            // -----
            if (flag_bt_sobe == 1) { flag_bt_sobe = 0; flag_bt_desce = 0;
flag_bt_entra = 0; flag_bt_esc = 0; estados = EST_LOOP; }
            if (flag_bt_desce == 1) { flag_bt_sobe = 0; flag_bt_desce = 0;
flag_bt_entra = 0; flag_bt_esc = 0; estados = EST_PARAR; }
            if (flag_bt_entra == 1) { flag_bt_sobe = 0; flag_bt_desce = 0;
flag_bt_entra = 0; flag_bt_esc = 0; estados = EST_COM_RODAR; }
            if (flag_bt_esc == 1) { flag_bt_sobe = 0; flag_bt_desce = 0;
flag_bt_entra = 0; flag_bt_esc = 0; estados = EST_COM_PARAR; }

            break;

        case EST_PARAR:

            // Operacao:
            // -----
            ui->rodar->setStyleSheet("background-color: none; color: rgb(255, 255,
255);");
            ui->leitura->setStyleSheet("background-color: none; color: rgb(255,
255, 255);");
            ui->parar->setStyleSheet("background-color: rgb(0, 136, 100); color:
rgb(255, 255, 255);");
            ui->altera->setStyleSheet("background-color: none; color: rgb(255, 255,
255);");
            ui->loop->setStyleSheet("background-color: none; color: rgb(255, 255,
255);");

            // Mudanca de estado:
            // -----
            if (flag_bt_sobe == 1) { flag_bt_sobe = 0; flag_bt_desce = 0;
flag_bt_entra = 0; flag_bt_esc = 0; estados = EST_RODAR; }
            if (flag_bt_desce == 1) { flag_bt_sobe = 0; flag_bt_desce = 0;
flag_bt_entra = 0; flag_bt_esc = 0; estados = EST_LEITURA; }

```



```

        if (flag_bt_entra == 1) { flag_bt_sobe = 0; flag_bt_desce = 0;
flag_bt_entra = 0; flag_bt_esc = 0; estados = EST_COM_PARAR; }
        if (flag_bt_esc == 1) { flag_bt_sobe = 0; flag_bt_desce = 0;
flag_bt_entra = 0; flag_bt_esc = 0; estados = EST_COM_PARAR; }

        break;

    case EST_LEITURA:

        // Operacao:
        // -----
        ui->rodar->setStyleSheet("background-color: none; color: rgb(255, 255,
255);");
        ui->leitura->setStyleSheet("background-color: rgb(0, 136, 100); color:
rgb(255, 255, 255);");
        ui->parar->setStyleSheet("background-color: none; color: rgb(255, 255,
255);");
        ui->altera->setStyleSheet("background-color: none; color: rgb(255, 255,
255);");
        ui->loop->setStyleSheet("background-color: none; color: rgb(255, 255,
255);");

        // Mudanca de estado:
        // -----
        if (flag_bt_sobe == 1) { flag_bt_sobe = 0; flag_bt_desce = 0;
flag_bt_entra = 0; flag_bt_esc = 0; estados = EST_PARAR; }
        if (flag_bt_desce == 1) { flag_bt_sobe = 0; flag_bt_desce = 0;
flag_bt_entra = 0; flag_bt_esc = 0; estados = EST_ALTERA; }
        if (flag_bt_entra == 1) { flag_bt_sobe = 0; flag_bt_desce = 0;
flag_bt_entra = 0; flag_bt_esc = 0; estados = EST_COM_LEITURA; flag_leitura_loop =
1;}

        if (flag_bt_esc == 1) { flag_bt_sobe = 0; flag_bt_desce = 0;
flag_bt_entra = 0; flag_bt_esc = 0; estados = EST_COM_PARAR; }

        break;

    case EST_ALTERA:

        // Operacao:
        // -----
        ui->rodar->setStyleSheet("background-color: none; color: rgb(255, 255,
255);");
        ui->leitura->setStyleSheet("background-color: none; color: rgb(255,
255, 255);");
        ui->parar->setStyleSheet("background-color: none; color: rgb(255, 255,
255);");
        ui->altera->setStyleSheet("background-color: rgb(0, 136, 100); color:
rgb(255, 255, 255);");
        ui->loop->setStyleSheet("background-color: none; color: rgb(255, 255,
255);");

        // Mudanca de estado:
        // -----
        if (flag_bt_sobe == 1) { flag_bt_sobe = 0; flag_bt_desce = 0;
flag_bt_entra = 0; flag_bt_esc = 0; estados = EST_LEITURA; }
        if (flag_bt_desce == 1) { flag_bt_sobe = 0; flag_bt_desce = 0;
flag_bt_entra = 0; flag_bt_esc = 0; estados = EST_LOOP; }
        if (flag_bt_entra == 1) { flag_bt_sobe = 0; flag_bt_desce = 0;
flag_bt_entra = 0; flag_bt_esc = 0; estados = EST_COM_ALTERA; }
        if (flag_bt_esc == 1) { flag_bt_sobe = 0; flag_bt_desce = 0;
flag_bt_entra = 0; flag_bt_esc = 0; estados = EST_COM_PARAR; }

        break;

    case EST_LOOP:

```

```

        // Operacao:
        // -----
        ui->rodar->setStyleSheet("background-color: none; color: rgb(255, 255,
255);");
        ui->leitura->setStyleSheet("background-color: none; color: rgb(255,
255, 255);");
        ui->parar->setStyleSheet("background-color: none; color: rgb(255, 255,
255);");
        ui->altera->setStyleSheet("background-color: none; color: rgb(255, 255,
255);");
        ui->loop->setStyleSheet("background-color: rgb(0, 136, 100); color:
rgb(255, 255, 255);");

        // Mudanca de estado:
        // -----
        if (flag_bt_sobe == 1) { flag_bt_sobe = 0; flag_bt_desce = 0;
flag_bt_entra = 0; flag_bt_esc = 0; estados = EST_ALTERA; }
        if (flag_bt_desce == 1) { flag_bt_sobe = 0; flag_bt_desce = 0;
flag_bt_entra = 0; flag_bt_esc = 0; estados = EST_RODAR; }
        if (flag_bt_entra == 1) { flag_bt_sobe = 0; flag_bt_desce = 0;
flag_bt_entra = 0; flag_bt_esc = 0; estados = EST_COM_LOOP; }
        if (flag_bt_esc == 1) { flag_bt_sobe = 0; flag_bt_desce = 0;
flag_bt_entra = 0; flag_bt_esc = 0; estados = EST_COM_PARAR; }

        break;

    case EST_COM_RODAR:

        // Flags de comunicacao:
        // -----
        flag_rodar = 1;
        flag_leitura = 0;
        flag_parar = 0;
        flag_altera = 0;
        flag_loop = 0;

        // Operacao:
        // -----
        ui->rodar->setStyleSheet("background-color: rgb(0, 136, 100); color:
rgb(255, 0, 0);");
        ui->leitura->setStyleSheet("background-color: none; color: rgb(255,
255, 255);");
        ui->parar->setStyleSheet("background-color: none; color: rgb(255, 255,
255);");
        ui->altera->setStyleSheet("background-color: none; color: rgb(255, 255,
255);");
        ui->loop->setStyleSheet("background-color: none; color: rgb(255, 255,
255);");

        // Mudanca de estado:
        // -----
        if (flag_bt_sobe == 1) { flag_bt_sobe = 0; flag_bt_desce = 0;
flag_bt_entra = 0; flag_bt_esc = 0; estados = EST_LOOP; }
        if (flag_bt_desce == 1) { flag_bt_sobe = 0; flag_bt_desce = 0;
flag_bt_entra = 0; flag_bt_esc = 0; estados = EST_PARAR; }
        if (flag_bt_entra == 1) { flag_bt_sobe = 0; flag_bt_desce = 0;
flag_bt_entra = 0; flag_bt_esc = 0; estados = EST_RODAR; }
        if (flag_bt_esc == 1) { flag_bt_sobe = 0; flag_bt_desce = 0;
flag_bt_entra = 0; flag_bt_esc = 0; estados = EST_COM_PARAR; }

        break;

    case EST_COM_PARAR:

        // Flags de comunicacao:
        // -----

```

```

        flag_rodar = 0;
        flag_leitura = 0;
        flag_parar = 1;
        flag_altera = 0;
        flag_loop = 0;

        // Operacao:
        // -----
        ui->rodar->setStyleSheet("background-color: none; color: rgb(255, 255,
255);");
        ui->leitura->setStyleSheet("background-color: none ; color: rgb(255,
255, 255);");
        ui->parar->setStyleSheet("background-color: rgb(0, 136, 100); color:
rgb(255, 0, 0);");
        ui->altera->setStyleSheet("background-color: none; color: rgb(255, 255,
255);");
        ui->loop->setStyleSheet("background-color: none; color: rgb(255, 255,
255);");

        // Mudanca de estado:
        // -----
        if (flag_bt_sobe == 1) { flag_bt_sobe = 0; flag_bt_desce = 0;
flag_bt_entra = 0; flag_bt_esc = 0; estados = EST_RODAR; }
        if (flag_bt_desce == 1) { flag_bt_sobe = 0; flag_bt_desce = 0;
flag_bt_entra = 0; flag_bt_esc = 0; estados = EST_LEITURA; }
        if (flag_bt_esc == 1) { flag_bt_sobe = 0; flag_bt_desce = 0;
flag_bt_entra = 0; flag_bt_esc = 0; estados = EST_PARAR; }

        break;

    case EST_COM_LEITURA:

        // Flags de comunicacao:
        // -----
        flag_rodar = 0;
        flag_leitura = 1;
        flag_parar = 0;
        flag_altera = 0;
        flag_loop = 0;

        // Operacao:
        // -----
        ui->rodar->setStyleSheet("background-color: none; color: rgb(255, 255,
255);");
        ui->leitura->setStyleSheet("background-color: rgb(0, 136, 100); color:
rgb(255, 0, 0);");
        ui->parar->setStyleSheet("background-color: none; color: rgb(255, 255,
255);");
        ui->altera->setStyleSheet("background-color: none; color: rgb(255, 255,
255);");
        ui->loop->setStyleSheet("background-color: none; color: rgb(255, 255,
255);");

        // Mudança de estado:
        // -----
        if (flag_bt_esc == 1) { flag_bt_sobe = 0; flag_bt_desce = 0;
flag_bt_entra = 0; flag_bt_esc = 0; estados = EST_LEITURA ; }

        break;

    case EST_COM_ALTERA:

        // Flags de comunicacao:

```

```

// -----
flag_rodar = 0;
flag_leitura = 0;
flag_parar = 0;
flag_altera = 0;
flag_loop = 0;

// Operacao:
// -----
ui->rodar->setStyleSheet("background-color: none; color: rgb(255, 255,
255);");
ui->leitura->setStyleSheet("background-color: none ; color: rgb(255,
255, 255);");
ui->parar->setStyleSheet("background-color: none; color: rgb(255, 255,
255);");
ui->altera->setStyleSheet("background-color: rgb(0, 136, 100); color:
rgb(255, 0, 0);");
ui->loop->setStyleSheet("background-color: none; color: rgb(255, 255,
255);");

// Mudanca de estado:
// -----
if (flag_bt_sobe == 1) { flag_bt_sobe = 0; flag_bt_desce = 0;
flag_bt_entra = 0; flag_bt_esc = 0; r_freq=r_freq+10; }
if (flag_bt_desce == 1) { flag_bt_sobe = 0; flag_bt_desce = 0;
flag_bt_entra = 0; flag_bt_esc = 0; r_freq=r_freq-10; }
if (flag_bt_esc == 1) { flag_bt_sobe = 0; flag_bt_desce = 0;
flag_bt_entra = 0; flag_bt_esc = 0; estados = EST_ALTERA; }
if (flag_bt_entra == 1) { flag_bt_sobe = 0; flag_bt_desce = 0;
flag_bt_entra = 0; flag_bt_esc = 0; flag_altera = 1;}

break;

case EST_COM_LOOP:

// Flags de comunicacao:
// -----
flag_rodar = 0;
flag_leitura = 0;
flag_parar = 0;
flag_altera = 0;
flag_loop = 1;

// Operacao:
// -----
ui->rodar->setStyleSheet("background-color: none; color: rgb(255, 255,
255);");
ui->leitura->setStyleSheet("background-color: none ; color: rgb(255,
255, 255);");
ui->parar->setStyleSheet("background-color: none; color: rgb(255, 255,
255);");
ui->altera->setStyleSheet("background-color: none; color: rgb(255, 255,
255);");
ui->loop->setStyleSheet("background-color: rgb(0, 136, 100); color:
rgb(255, 0, 0);");

// Mudanca de estado:
// -----
if (flag_bt_esc == 1) { flag_bt_sobe = 0; flag_bt_desce = 0;
flag_bt_entra = 0; flag_bt_esc = 0; estados = EST_LOOP; }
if (flag_bt_entra == 1) { flag_bt_sobe = 0; flag_bt_desce = 0;
flag_bt_entra = 0; flag_bt_esc = 0; estados = EST_LOOP; }

break;

}

```

```

// *****
// PROCESSAMENTO DAS FUNCOES DO PROTOCOLO DE COMUNICACAO
// *****

if (flag_rodar == 1) {

    tamanho_buffer_envia = 11;

    buffer_envia[0] = endereco; // Endereco
    buffer_envia[1] = 0x10;      // Funcao de escrita em multiplas bobinas
    buffer_envia[2] = 0x00;      // Registro inicial:
    buffer_envia[3] = 0x01;      // 0001H
    buffer_envia[4] = 0x00;      // Numero de registros:
    buffer_envia[5] = 0x01;      // 0001H
    buffer_envia[6] = 0x02;      // Numero de registros * 2
    buffer_envia[7] = 0x00;      // Valor do registro:
    buffer_envia[8] = 0x01;      // 0001H - Comando rodar

    crc(buffer_envia, 9, buffer_crc); // Funcao para calculo de CRC.
    buffer_envia[9] = buffer_crc[1];   // Armazena calculo de CRC no
    buffer_envia[10] = buffer_crc[0];  // buffer de envio.

    serial->write(buffer_envia,tamanho_buffer_envia); // Envia dados
    dado_serial = 0;

    flag_rodar = 0; // Zera flag
    estados = EST_RODAR; // Volta ao estado anterior
    flag_rodar_rx = 1; // Inicia flag de recebimento
    tamanho_serial = 0; // Zera tamanho do buffer
    timeout_modbus->setSingleShot(true); // Inicia contador de timeout
    timeout_modbus->start(TIMEOUT);

}

if (flag_parar == 1) {

    buffer_envia[0] = endereco; // Endereco
    buffer_envia[1] = 0x10;      // Funcao de escrita em multiplas bobinas
    buffer_envia[2] = 0x00;      // Registro inicial:
    buffer_envia[3] = 0x01;      // 0001H
    buffer_envia[4] = 0x00;      // Numero de registros:
    buffer_envia[5] = 0x01;      // 0001H
    buffer_envia[6] = 0x02;
    buffer_envia[7] = 0x00;
    buffer_envia[8] = 0x00;

    tamanho_buffer_envia = 11;

    crc(buffer_envia, 9, buffer_crc); // Funcao para calculo de CRC.
    buffer_envia[9] = buffer_crc[1];   // Armazena calculo de CRC no
    buffer_envia[10] = buffer_crc[0];  // buffer de envio.

    serial->write(buffer_envia,tamanho_buffer_envia); // Envia dados
    dado_serial = 0;

    flag_parar = 0; // Zera flag
    estados = EST_PARAR; // Volta ao estado anterior
    flag_parar_rx = 1; // Inicia flag de recebimento
    tamanho_serial = 0; // Zera tamanho do buffer
    timeout_modbus->setSingleShot(true); // Inicia contador de timeout
    timeout_modbus->start(TIMEOUT);

}

if (flag_altera == 1) {

```

```

buffer_envia[0] = endereco; // Endereco
buffer_envia[1] = 0x10;     // Funcao de escrita em multiplas bobinas
buffer_envia[2] = 0x00;     // Registro inicial:
buffer_envia[3] = 0x02;     // 0002H
buffer_envia[4] = 0x00;     // Numero de registros:
buffer_envia[5] = 0x01;     // 0001H
buffer_envia[6] = 0x02;
buffer_envia[7] = ((r_freq)&0xff00)>>8;
buffer_envia[8] = ((r_freq)&0x00ff);

tamanho_buffer_envia = 11;

crc(buffer_envia, 9, buffer_crc); // Funcao para calculo de CRC.
buffer_envia[9] = buffer_crc[1];  // Armazena calculo de CRC no
buffer_envia[10] = buffer_crc[0]; // buffer de envio.

serial->write(buffer_envia,tamanho_buffer_envia); // Envia dados
dado_serial = 0;

flag_altera = 0; // Zera flag
estados = EST_ALTERA; // Volta ao estado anterior
flag_altera_rx = 1; // Inicia flag de recebimento
tamanho_serial = 0; // Zera tamanho do buffer
timeout_modbus->setSingleShot(true); // Inicia contador de timeout
timeout_modbus->start(TIMEOUT);

}

if (flag_leitura == 1) {

    if (flag_leitura_loop == 1) {

        tamanho_buffer_envia = 8;

        buffer_envia[0] = endereco; // Endereco
        buffer_envia[1] = 0x03;     // Funcao de leitura
        buffer_envia[2] = 0x00;     // Registro inicial:
        buffer_envia[3] = 0x24;     // 0024H
        buffer_envia[4] = 0x00;     // Numero de registros:
        buffer_envia[5] = 0x01;     // 0001H

        crc(buffer_envia, 6, buffer_crc); // Funcao para calculo de CRC
        buffer_envia[6] = buffer_crc[1];  // Armazena calculo de CRC no
        buffer_envia[7] = buffer_crc[0];  // buffer de envio.

        serial->write(buffer_envia,tamanho_buffer_envia); // Envia dados
        dado_serial = dado_serial2;
        dado_serial2 = 0;

        flag_leitura = 0; // Zera flag
        flag_leitura_rx = 1; // Inicia flag de recebimento
        tamanho_serial = 0; // Zera tamanho do buffer
        timeout_modbus->setSingleShot(true); // Inicia contador de timeout
        timeout_modbus->start(TIMEOUT);
        flag_leitura_loop = 0;
    }

}

if (flag_loop == 1) {

    buffer_envia[0] = endereco; // Endereco
    buffer_envia[1] = 0x08;     // Funcao de loop
    buffer_envia[2] = 0x39;     //Codigo de teste
    buffer_envia[3] = 0x06;     // definido pelo usuario
    buffer_envia[4] = 0x38;     // Dado a ser enviado
    buffer_envia[5] = 0x71;     // 3871H

    tamanho_buffer_envia = 8;

```

```

        crc(buffer_envia, 6, buffer_crc);    // Funcao para calculo de CRC.
        buffer_envia[6] = buffer_crc[1];    // Armazena calculo de CRC no
        buffer_envia[7] = buffer_crc[0];    // buffer de envio.

        serial->write(buffer_envia,tamanho_buffer_envia);    // Envia dados
        dado_serial = 0;

        flag_loop = 0;                        // Zera flag
        estados = EST_LOOP;                  // Volta ao estado anterior
        flag_loop_rx = 1;                    // Inicia flag de recebimento
        tamanho_serial = 0;                  // Zera tamanho do buffer
        timeout_modbus->setSingleShot(true); // Inicia contador de timeout
        timeout_modbus->start(TIMEOUT);

    }

    // *****
    // ATUALIZACAO DA INTERFACE GRÁFICA
    // *****

    // Estados:
    // -----
    QString estados_ui = "Estado inicial";
    if (estados == 0) estados_ui = "0 - Menu rodar";
    if (estados == 1) estados_ui = "1 - Menu ler frequencia";
    if (estados == 2) estados_ui = "2 - Menu parar";
    if (estados == 3) estados_ui = "3 - Menu alterar frequencia";
    if (estados == 4) estados_ui = "4 - Menu teste de loop";
    if (estados == 5) estados_ui = "5 - Comando rodar";
    if (estados == 6) estados_ui = "6 - Comando ler frequencia";
    if (estados == 7) estados_ui = "7 - Comando parar";
    if (estados == 8) estados_ui = "8 - Comando altera frequencia";
    if (estados == 9) estados_ui = "9 - Comando teste de loop";
    ui->maq_est->setText(estados_ui);

    // Barra de comunicacao:
    // -----

    ui->barra_com->setText(barra_com_ui);
    ui->barra_com->setStyleSheet(barra_cor);

    // Buffer TX:
    // -----
    QByteArray stx = QByteArray(buffer_envia,tamanho_buffer_envia);
    ui->buffer_tx_lcd->setDigitCount(tamanho_buffer_envia*2);
    ui->buffer_tx_lcd->display(QString(stx.toHexString()));

    // Buffer RX:
    // -----
    QByteArray srx = QByteArray(dado_serial,dado_serial.size());
    ui->buffer_rx_lcd->setDigitCount(dado_serial.size()*2);
    ui->buffer_rx_lcd->display(QString(srx.toHexString()));

    // Frequencia de referencia:
    // -----
    if (r_freq < 0) r_freq = 0;
    if (r_freq > 6000) r_freq = 6000;
    s_freq = (float) r_freq / 100;
    ui->ref_freq->display(QString("%1").arg(s_freq,0,'f',2));
    ui->freq_saida->display(QString("%1").arg(f_saida,0,'f',2));

}

// *****
// FUNCAO DE RECEBIMENTO DE DADOS
// *****

```

```

void MainWindow::serialRecebido(void) {

    // RODAR
    // -----

    if (flag_rodar_rx == 1) {

        leitura = serial->readAll();          // Le dados do buffer de leitura

        dado_serial.append(leitura);          // Adiciona dados ao dado_serial

        tamanho_serial += leitura.size();     // Calcula tamanho dos dados

        // Verifica terceiro byte - function code
        // -----
        QString function_code = QString(dado_serial.toHex().at(2));
        function_code.append(QString(dado_serial.toHex().at(3)));

        // Verifica se o terceiro byte (function code) esta ok
        // -----
        if (function_code== "10") {

            if (tamanho_serial >= 8) {
                tamanho_serial = 0;           // Zera buffer
                timeout_modbus->stop();        // Zera contador de timeout
                flag_rodar_rx = 0;            // Zera flag rx

                // Calculo do CRC:
                // -----
                bool ok = true;
                buffer_recebe[0] = dado_serial.toHex().mid(0,2).toInt(&ok,16);
                buffer_recebe[1] = dado_serial.toHex().mid(2,2).toInt(&ok,16);
                buffer_recebe[2] = dado_serial.toHex().mid(4,2).toInt(&ok,16);
                buffer_recebe[3] = dado_serial.toHex().mid(6,2).toInt(&ok,16);
                buffer_recebe[4] = dado_serial.toHex().mid(8,2).toInt(&ok,16);
                buffer_recebe[5] = dado_serial.toHex().mid(10,2).toInt(&ok,16);

                crc(buffer_recebe, 6, buffer_crc); // Funcao para calculo de CRC
                buffer_recebe[6] = buffer_crc[1];
                buffer_recebe[7] = buffer_crc[0];

                if (buffer_recebe[6] ==
dado_serial.toHex().mid(12,2).toInt(&ok,16)) {
                    if (buffer_recebe[7] ==
dado_serial.toHex().mid(14,2).toInt(&ok,16)) {
                        barra_com_ui = "Comando rodar - Comunicacao realizada!";
                        barra_cor = "background-color: rgb(0, 70, 0);";
                    } else {
                        barra_com_ui = "Erro de CRC";
                        barra_cor = "background-color: rgb(70, 0, 0);";
                    }
                } else {
                    barra_com_ui = "Erro de CRC";
                    barra_cor = "background-color: rgb(70, 0, 0);";
                }
            }

            // Terceiro byte com function error
            // -----

            if (function_code == "90") {

                if (tamanho_serial >= 5) {
                    tamanho_serial = 0;       // Zera buffer
                    timeout_modbus->stop();    // Zera contador de timeout

```



```

    flag_rodar_rx = 0;          // Zera flag rx

    // Identificação de erro:
    // -----
    bool ok = true;
    buffer_recebe[0] = dado_serial.toHex().mid(0,2).toInt(&ok,16);
    buffer_recebe[1] = dado_serial.toHex().mid(2,2).toInt(&ok,16);
    buffer_recebe[2] = dado_serial.toHex().mid(4,2).toInt(&ok,16);

    if (buffer_recebe[2] == 0x01) {
        barra_com_ui = "Erro no código da função";
        barra_cor = "background-color: rgb(70, 0, 70);";
    }

    if (buffer_recebe[2] == 0x02) {
        barra_com_ui = "Erro no número do registro";
        barra_cor = "background-color: rgb(70, 0, 70);";
    }

    if (buffer_recebe[2] == 0x03) {
        barra_com_ui = "Erro na contagem de bits";
        barra_cor = "background-color: rgb(70, 0, 70);";
    }

    if (buffer_recebe[2] == 0x21) {
        barra_com_ui = "Erro na configuração de dados";
        barra_cor = "background-color: rgb(70, 0, 70);";
    }

    if (buffer_recebe[2] == 0x22) {
        barra_com_ui = "Erro no modo de escrita";
        barra_cor = "background-color: rgb(70, 0, 70);";
    }

    if (buffer_recebe[2] == 0x23) {
        barra_com_ui = "Subtensão no barramento DC";
        barra_cor = "background-color: rgb(70, 0, 70);";
    }

    if (buffer_recebe[2] == 0x24) {
        barra_com_ui = "Erro de escrita durante parametrização";
        barra_cor = "background-color: rgb(70, 0, 70);";
    }

    }
}

// PARAR
// -----

if (flag_parar_rx == 1) {

    leitura = serial->readAll();          // Le dados do buffer de leitura

    dado_serial.append(leitura);          // Adiciona dados ao dado_serial

    tamanho_serial += leitura.size();     // Calcula tamanho dos dados

    // Verifica terceiro byte - function code
    // -----
    QString function_code = QString(dado_serial.toHex().at(2));
    function_code.append(QString(dado_serial.toHex().at(3)));

    // Verifica se o terceiro byte (function code) está ok
    // -----
    if (function_code == "10") {

```

```

if (tamanho_serial >= 8) {
    tamanho_serial = 0;          // Zera buffer
    timeout_modbus->stop();      // Zera contador de timeout
    flag_parar_rx = 0;          // Zera flag rx

    // Calculo do CRC:
    // -----
    bool ok = true;
    buffer_recebe[0] = dado_serial.toHex().mid(0,2).toInt(&ok,16);
    buffer_recebe[1] = dado_serial.toHex().mid(2,2).toInt(&ok,16);
    buffer_recebe[2] = dado_serial.toHex().mid(4,2).toInt(&ok,16);
    buffer_recebe[3] = dado_serial.toHex().mid(6,2).toInt(&ok,16);
    buffer_recebe[4] = dado_serial.toHex().mid(8,2).toInt(&ok,16);
    buffer_recebe[5] = dado_serial.toHex().mid(10,2).toInt(&ok,16);

    crc(buffer_recebe, 6, buffer_crc); // Funcao para calculo de CRC
    buffer_recebe[6] = buffer_crc[1];
    buffer_recebe[7] = buffer_crc[0];

    if (buffer_recebe[6] ==
dado_serial.toHex().mid(12,2).toInt(&ok,16)) {
        if (buffer_recebe[7] ==
dado_serial.toHex().mid(14,2).toInt(&ok,16)) {
            barra_com_ui = "Comando parar - Comunicacao realizada!";
            barra_cor = "background-color: rgb(0, 70, 0);";
        } else {
            barra_com_ui = "Erro de CRC";
            barra_cor = "background-color: rgb(70, 0, 0);";
        }
    } else {
        barra_com_ui = "Erro de CRC";
        barra_cor = "background-color: rgb(70, 0, 0);";
    }
}

}

// Terceiro byte com function error
// -----

if (function_code == "90") {

    if (tamanho_serial >= 5) {
        tamanho_serial = 0;          // Zera buffer
        timeout_modbus->stop();      // Zera contador de timeout
        flag_parar_rx = 0;          // Zera flag rx

        // Identificacao de erro:
        // -----
        bool ok = true;
        buffer_recebe[0] = dado_serial.toHex().mid(0,2).toInt(&ok,16);
        buffer_recebe[1] = dado_serial.toHex().mid(2,2).toInt(&ok,16);
        buffer_recebe[2] = dado_serial.toHex().mid(4,2).toInt(&ok,16);

        if (buffer_recebe[2] == 0x01) {
            barra_com_ui = "Erro no codigo da funcao";
            barra_cor = "background-color: rgb(70, 0, 70);";
        }

        if (buffer_recebe[2] == 0x02) {
            barra_com_ui = "Erro no numero do registro";
            barra_cor = "background-color: rgb(70, 0, 70);";
        }

        if (buffer_recebe[2] == 0x03) {

```

```

        barra_com_ui = "Erro na contagem de bits";
        barra_cor = "background-color: rgb(70, 0, 70);";
    }

    if (buffer_recebe[2] == 0x21) {
        barra_com_ui = "Erro na configuracao de dados";
        barra_cor = "background-color: rgb(70, 0, 70);";
    }

    if (buffer_recebe[2] == 0x22) {
        barra_com_ui = "Erro no modo de escrita";
        barra_cor = "background-color: rgb(70, 0, 70);";
    }

    if (buffer_recebe[2] == 0x23) {
        barra_com_ui = "Subtensao no barramento DC";
        barra_cor = "background-color: rgb(70, 0, 70);";
    }

    if (buffer_recebe[2] == 0x24) {
        barra_com_ui = "Erro de escrita durante parametrizacao";
        barra_cor = "background-color: rgb(70, 0, 70);";
    }
}

}

// ALTERA
// -----

if (flag_altera_rx == 1) {
    leitura = serial->readAll();           // Le dados do buffer de leitura

    dado_serial.append(leitura);           // Adiciona dados ao dado_serial

    tamanho_serial += leitura.size();      // Calcula tamanho dos dados

    // Verifica terceiro byte - function code
    // -----
    QString function_code = QString(dado_serial.toHex().at(2));
    function_code.append(QString(dado_serial.toHex().at(3)));

    // Verifica se o terceiro byte (function code) esta ok
    // -----
    if (function_code == "10") {

        if (tamanho_serial >= 8) {
            tamanho_serial = 0;           // Zera buffer
            timeout_modbus->stop();        // Zera contador de timeout
            flag_altera_rx = 0;           // Zera flag rx

            // C culo do CRC:
            // -----
            bool ok = true;
            buffer_recebe[0] = dado_serial.toHex().mid(0,2).toInt(&ok,16);
            buffer_recebe[1] = dado_serial.toHex().mid(2,2).toInt(&ok,16);
            buffer_recebe[2] = dado_serial.toHex().mid(4,2).toInt(&ok,16);
            buffer_recebe[3] = dado_serial.toHex().mid(6,2).toInt(&ok,16);
            buffer_recebe[4] = dado_serial.toHex().mid(8,2).toInt(&ok,16);
            buffer_recebe[5] = dado_serial.toHex().mid(10,2).toInt(&ok,16);

            crc(buffer_recebe, 6, buffer_crc); // Fun o para c culo de

            buffer_recebe[6] = buffer_crc[1];
            buffer_recebe[7] = buffer_crc[0];

```

```

        if (buffer_recebe[6] ==
dado_serial.toHex().mid(12,2).toInt(&ok,16)) {
            if (buffer_recebe[7] ==
dado_serial.toHex().mid(14,2).toInt(&ok,16)) {
                barra_com_ui = "Frequencia alterada com sucesso";
                barra_cor = "background-color: rgb(0, 70, 0);";
            } else {
                barra_com_ui = "Erro de CRC";
                barra_cor = "background-color: rgb(70, 0, 0);";
            }
        } else {
            barra_com_ui = "Erro de CRC";
            barra_cor = "background-color: rgb(70, 0, 0);";
        }
    }

}

// Terceiro byte com function error
// -----

if (function_code == "90") {

    if (tamanho_serial >= 5) {
        tamanho_serial = 0;        // Zera buffer
        timeout_modbus->stop(); // Zera contador de timeout
        flag_altera_rx = 0;        // Zera flag rx

        // Identificação de erro:
        // -----
        bool ok = true;
        buffer_recebe[0] = dado_serial.toHex().mid(0,2).toInt(&ok,16);
        buffer_recebe[1] = dado_serial.toHex().mid(2,2).toInt(&ok,16);
        buffer_recebe[2] = dado_serial.toHex().mid(4,2).toInt(&ok,16);

        if (buffer_recebe[2] == 0x01) {
            barra_com_ui = "Erro no código da função";
            barra_cor = "background-color: rgb(70, 0, 70);";
        }

        if (buffer_recebe[2] == 0x02) {
            barra_com_ui = "Erro no número do registro";
            barra_cor = "background-color: rgb(70, 0, 70);";
        }

        if (buffer_recebe[2] == 0x03) {
            barra_com_ui = "Erro na contagem de bits";
            barra_cor = "background-color: rgb(70, 0, 70);";
        }

        if (buffer_recebe[2] == 0x21) {
            barra_com_ui = "Erro na configuração de dados";
            barra_cor = "background-color: rgb(70, 0, 70);";
        }

        if (buffer_recebe[2] == 0x22) {
            barra_com_ui = "Erro no modo de escrita";
            barra_cor = "background-color: rgb(70, 0, 70);";
        }

        if (buffer_recebe[2] == 0x23) {
            barra_com_ui = "Subtensão no barramento DC";
            barra_cor = "background-color: rgb(70, 0, 70);";
        }
    }
}

```

```

        if (buffer_recebe[2] == 0x24) {
            barra_com_ui = "Erro de escrita durante parametrizacao";
            barra_cor = "background-color: rgb(70, 0, 70);";
        }

    }

}

// LOOP
// -----

if (flag_loop_rx == 1) {

    leitura = serial->readAll();           // Le dados do buffer de leitura

    dado_serial.append(leitura);           // Adiciona dados ao dado_serial

    tamanho_serial += leitura.size();      // Calcula tamanho dos dados

    // Verifica terceiro byte - function code
    // -----
    QString function_code = QString(dado_serial.toHex().at(2));
    function_code.append(QString(dado_serial.toHex().at(3)));

    // Verifica se o terceiro byte (function code) esta ok
    // -----
    if (function_code == "08") {

        if (tamanho_serial >= 8) {
            tamanho_serial = 0;           // Zera buffer
            timeout_modbus->stop();        // Zera contador de timeout
            flag_loop_rx = 0;             // Zera flag rx

            // Calculo do CRC:
            // -----
            bool ok = true;
            buffer_recebe[0] = dado_serial.toHex().mid(0,2).toInt(&ok,16);
            buffer_recebe[1] = dado_serial.toHex().mid(2,2).toInt(&ok,16);
            buffer_recebe[2] = dado_serial.toHex().mid(4,2).toInt(&ok,16);
            buffer_recebe[3] = dado_serial.toHex().mid(6,2).toInt(&ok,16);
            buffer_recebe[4] = dado_serial.toHex().mid(8,2).toInt(&ok,16);
            buffer_recebe[5] = dado_serial.toHex().mid(10,2).toInt(&ok,16);

            crc(buffer_recebe, 6, buffer_crc); // Funcao para calculo de CRC
            buffer_recebe[6] = buffer_crc[1];
            buffer_recebe[7] = buffer_crc[0];

            if (buffer_recebe[6] ==
                dado_serial.toHex().mid(12,2).toInt(&ok,16)) {
                if (buffer_recebe[7] ==
                    dado_serial.toHex().mid(14,2).toInt(&ok,16)) {

                    if (dado_serial.toHex() ==
                        QByteArray(buffer_envia,8).toHex()) {

                        barra_com_ui = "Teste de loop realizado com sucesso";
                        barra_cor = "background-color: rgb(0, 70, 0);";
                    }

                } else {
                    barra_com_ui = "Erro de CRC";
                }
            }
        }
    }
}

```

```

        barra_cor = "background-color: rgb(70, 0, 0);";
    }
} else {
    barra_com_ui = "Erro de CRC";
    barra_cor = "background-color: rgb(70, 0, 0);";
}

}

}

// Terceiro byte com function error
// -----

if (function_code == "89") {

    if (tamanho_serial >= 6) {
        qDebug() << "Dado serial: " << QString(dado_serial.toHex());
        tamanho_serial = 0; // Zera buffer
        timeout_modbus->stop(); // Zera contador de timeout
        flag_loop_rx = 0; // Zera flag rx

        // Identificação de erro:
        // -----
        bool ok = true;
        buffer_recebe[0] = dado_serial.toHex().mid(0,2).toInt(&ok,16);
        buffer_recebe[1] = dado_serial.toHex().mid(2,2).toInt(&ok,16);
        buffer_recebe[2] = dado_serial.toHex().mid(4,2).toInt(&ok,16);

        if (buffer_recebe[2] == 0x01) {
            barra_com_ui = "Erro no código da função";
            barra_cor = "background-color: rgb(70, 0, 70);";
        }

        if (buffer_recebe[2] == 0x02) {
            barra_com_ui = "Erro no número do registro";
            barra_cor = "background-color: rgb(70, 0, 70);";
        }

        if (buffer_recebe[2] == 0x03) {
            barra_com_ui = "Erro na contagem de bits";
            barra_cor = "background-color: rgb(70, 0, 70);";
        }

        if (buffer_recebe[2] == 0x21) {
            barra_com_ui = "Erro na configuração de dados";
            barra_cor = "background-color: rgb(70, 0, 70);";
        }

        if (buffer_recebe[2] == 0x22) {
            barra_com_ui = "Erro no modo de escrita";
            barra_cor = "background-color: rgb(70, 0, 70);";
        }

        if (buffer_recebe[2] == 0x23) {
            barra_com_ui = "Subtensão no barramento DC";
            barra_cor = "background-color: rgb(70, 0, 70);";
        }

        if (buffer_recebe[2] == 0x24) {
            barra_com_ui = "Erro de escrita durante parametrização";
            barra_cor = "background-color: rgb(70, 0, 70);";
        }

    }

}

```

```

    }

}

// LEITURA
// -----

if (flag_leitura_rx == 1) {

    leitura = serial->readAll();          // Le dados do buffer de leitura

    dado_serial2.append(leitura);        // Adiciona dados ao dado_serial

    tamanho_serial += leitura.size();    // Calcula tamanho dos dados

    // Verifica terceiro byte - function code
    // -----
    QString function_code = QString(dado_serial2.toHex().at(2));
    function_code.append(QString(dado_serial2.toHex().at(3)));

    // Verifica se o terceiro byte (function code) esta ok
    // -----
    if (function_code== "03") {

        if (tamanho_serial >= 7) {
            qDebug() << "Dado serial: " << QString(dado_serial2.toHex());
            tamanho_serial = 0;          // Zera buffer
            timeout_modbus->stop();      // Zera contador de timeout
            flag_leitura_rx = 0;         // Zera flag rx

            // Calculo do CRC:
            // -----
            bool ok = true;
            buffer_recebe[0] = dado_serial2.toHex().mid(0,2).toInt(&ok,16);
            buffer_recebe[1] = dado_serial2.toHex().mid(2,2).toInt(&ok,16);
            buffer_recebe[2] = dado_serial2.toHex().mid(4,2).toInt(&ok,16);
            buffer_recebe[3] = dado_serial2.toHex().mid(6,2).toInt(&ok,16);
            buffer_recebe[4] = dado_serial2.toHex().mid(8,2).toInt(&ok,16);

            crc(buffer_recebe, 5, buffer_crc);    // Funcao para calculo de CRC
            buffer_recebe[5] = buffer_crc[1];
            buffer_recebe[6] = buffer_crc[0];

            if (buffer_recebe[5] ==
dado_serial2.toHex().mid(10,2).toInt(&ok,16)) {
                if (buffer_recebe[6] ==
dado_serial2.toHex().mid(12,2).toInt(&ok,16)) {
                    f_saida = (int)((buffer_recebe[3]<<8 |
buffer_recebe[4]));
                    f_saida = (float) f_saida/100;

                    if (estados == EST_COM_LEITURA) flag_leitura_loop = 1;
                    else flag_leitura_loop = 0;

                    barra_com_ui = "Leitura de frequencia bem sucedida";
                    barra_cor = "background-color: rgb(0, 70, 0);";
                } else {
                    barra_com_ui = "Erro de CRC";
                    barra_cor = "background-color: rgb(70, 0, 0);";
                }
            } else {
                barra_com_ui = "Erro de CRC";
                barra_cor = "background-color: rgb(70, 0, 0);";
            }
        }

    }

}

```

```

}

// Terceiro byte com function error
// -----

if (function_code == "83") {

    if (tamanho_serial >= 5) {
        tamanho_serial = 0;          // Zera buffer
        timeout_modbus->stop(); // Zera contador de timeout
        flag_leitura_rx = 0;          // Zera flag rx

        // Identificação de erro:
        // -----
        bool ok = true;
        buffer_recebe[0] = dado_serial2.toHex().mid(0,2).toInt(&ok,16);
        buffer_recebe[1] = dado_serial2.toHex().mid(2,2).toInt(&ok,16);
        buffer_recebe[2] = dado_serial2.toHex().mid(4,2).toInt(&ok,16);

        dado_serial = dado_serial2;

        if (buffer_recebe[2] == 0x01) {
            barra_com_ui = "Erro no código da função";
            barra_cor = "background-color: rgb(70, 0, 70);";
        }

        if (buffer_recebe[2] == 0x02) {
            barra_com_ui = "Erro no número do registro";
            barra_cor = "background-color: rgb(70, 0, 70);";
        }

        if (buffer_recebe[2] == 0x03) {
            barra_com_ui = "Erro na contagem de bits";
            barra_cor = "background-color: rgb(70, 0, 70);";
        }

        if (buffer_recebe[2] == 0x21) {
            barra_com_ui = "Erro na configuração de dados";
            barra_cor = "background-color: rgb(70, 0, 70);";
        }

        if (buffer_recebe[2] == 0x22) {
            barra_com_ui = "Erro no modo de escrita";
            barra_cor = "background-color: rgb(70, 0, 70);";
        }

        if (buffer_recebe[2] == 0x23) {
            barra_com_ui = "Subtensão no barramento DC";
            barra_cor = "background-color: rgb(70, 0, 70);";
        }

        if (buffer_recebe[2] == 0x24) {
            qDebug() << "Erro de escrita durante parametrização";
            barra_com_ui = "Erro de escrita durante parametrização";
            barra_cor = "background-color: rgb(70, 0, 70);";
        }

    }

}

}

// *****
// FUNÇÃO PARA ESTOURO DE TEMPO DE RESPOSTA

```



```
// *****  
  
void MainWindow::timeout_alert(void) {  
  
    barra_com_ui = "Erro de timeout";  
    barra_cor = "background-color: rgb(70, 0, 0);";  
    flag_rodar_rx = 0;  
    flag_parar_rx = 0;  
    flag_altera_rx = 0;  
    flag_leitura_rx = 0;  
    flag_loop_rx = 0;  
  
    timeout_modbus->stop();  
  
}
```

ANEXO A: Exemplos de comandos MEMOBUS/Modbus

◆ Leitura do Conteúdo de Registros MEMOBUS/Modbus do Inversor

Usando o código de função 03H (Ler), um máximo de 16 registros MEMOBUS/Modbus podem ser lidos por vez.

A tabela a seguir mostra exemplos de mensagens ao ler sinais de estado, detalhes de erro, estado de link de dados e referências de frequência do inversor auxiliar 2.

Mensagem de Comando			Mensagem de Resposta (normal)			Mensagem de Resposta (falha)		
Endereço de Auxiliar		02H	Endereço de Auxiliar		02H	Endereço de Auxiliar		02H
Código de Função		03H	Código de Função		03H	Código de Função		83H
N° de Início	Superior	00H	Quantidade de Dados		08H	Código de Erro		03H
	Limite	20H	1° registro de armazenamento	Superior	00H	CRC-16	Superior	F1H
Quantidade de Dados	Superior	00H		Limite	65H		Limite	31H
	Limite	04H	Próximo registro de armazenamento	Superior	00H			
CRC-16	Superior	45H		Limite	00H			
	Limite	F0H	Próximo registro de armazenamento	Superior	00H			
				Limite	00H			
			Próximo registro de armazenamento	Superior	01H			
				Limite	F4H			
			CRC-16	Superior	AFH			
				Limite	82H			

◆ Teste de Retorno

O código de função 08H realiza um teste de retorno que retorna uma mensagem de resposta com exatamente o mesmo conteúdo que a mensagem de comando. A mensagem de resposta pode ser usada para verificar a comunicação entre o controlador e o auxiliar. Também é possível configurar código de teste e valores de dados definidos pelo usuário.

A tabela a seguir mostra um exemplo de mensagem ao realizar um teste de retorno com o inversor auxiliar 1.

Mensagem de Comando			Mensagem de Resposta (normal)			Mensagem de Resposta (falha)		
Endereço de Auxiliar		01H	Endereço de Auxiliar		01H	Endereço de Auxiliar		01H
Código de Função		08H	Código de Função		08H	Código de Função		89H
Código do Teste	Superior	00H	Código do Teste	Superior	00H	Código de Erro		01H
	Límite	00H		Límite	00H	CRC-16	Superior	86H
Dados	Superior	A5H	Dados	Superior	A5H		Límite	50H
	Límite	37H		Límite	37H			
CRC-16	Superior	DAH	CRC-16	Superior	DAH			
	Límite	8DH		Límite	8DH			

◆ Gravação em Diversos Registros

O código de função 10H permite que o usuário grave diversos registros MEMOBUS/Modbus do inversor com uma única mensagem. Esse processo é semelhante à leitura de registro, pois o endereço do primeiro registro a ser escrito e a quantidade de dados são configurados na mensagem de comando. Os dados a serem gravados devem ser consecutivos para que os endereços de registro estejam na ordem, começando com o endereço especificado na mensagem de comando. A ordem dos dados deve ser byte alto seguido de byte baixo.

A tabela a seguir mostra um exemplo de uma mensagem na qual uma operação adiante foi estabelecida com uma referência de frequência de 60.0 Hz para o inversor auxiliar 1.

Se valores de parâmetro forem alterados usando o comando Gravar, um comando Enter pode ser necessário para ativar ou salvar os dados, dependendo da configuração de H5-11. *Consulte H5-11: Seleção da Função Enter nas Comunicações na página 606 e Consulte Comando Enter na página 629* para descrições detalhadas.

Mensagem de Comando			Mensagem de Resposta (normal)			Mensagem de Resposta (falha)		
Endereço de Auxiliar		01H	Endereço de Auxiliar		01H	Endereço de Auxiliar		01H
Código de Função		10H	Código de Função		10H	Código de Função		90H
Nº de Início	Superior	00H	Nº de Início	Superior	00H	Código de Erro		02H
	Limite	01H		Limite	01H	CRC-16	Superior	CDH
Quantidade de Dados	Superior	00H	Quantidade de Dados	Superior	00H		Limite	C1H
	Limite	02H		Limite	02H			
Número de Bytes		04H	CRC-16	Superior	10H			
Dados de Início	Superior	00H		Limite	08H			
	Limite	01H						
Próximos Dados	Superior	02H						
	Limite	58H						
CRC-16	Superior	63H						
	Limite	39H						

Nota: Dobre o número da quantidade de dados para o número de bytes na mensagem de comando.

Referências Bibliográficas

- [1] WEG; *Guia de Aplicação – Inversores de Frequência*. Disponível em: <http://www.mundoeletrico.com/downloads/Guia_de_Aplicacao_de_Inversores_de_Frequencia.pdf>. Acesso em 19 out. 2014.
- [2] CHAPMAN, S.J.; *Electric Machinery Fundamentals*, 4 ed New York, McGraw-Hill, 2005.
- [3] FILHO, G. F.; *Motor de Indução*, 8 ed São Paulo, Érica, 2002.
- [4] YASKAWA; *Manual do inversor de frequência A1000*. Disponível em: <https://www.yaskawa.com/pycprd/lookup/bannerDocDownload/query_type=get_document&document_id=SIEPC71061641&name=SIEPC71061641.pdf&web_access=Public>. Acesso em 22 out. 2014.
- [5] MACKAY, S.; WRIGHT, E.; REYNDERS, D.; PARK, J.; *Practical Industrial Data Networks: Design, Installation and Troubleshooting*, 1 ed Great Britain, Elsevier, 2004.
- [6] MATARAZZO, E. A.; SILVEIRA, L. M.; *O Modelo OSI de Interconexão de Sistemas Abertos*. Disponível em: <<http://www.teleco.com.br/tutoriais/tutorialosi/default.asp>>. Acesso em 23 out. 2014.
- [7] TANENBAUM, A. S.; *Computer Networks*, 4 ed Upper Saddle River, NJ, Prentice Hall, 2003.
- [8] LUGLI, A. B.; *REDES INDUSTRIAIS – Evolução, Motivação e Funcionamento*. Disponível em: <http://www.inatel.br/biblioteca/component/docman/doc_download/5078-redes-industriais-evolucao-motivacao-e-funcionamento>. Acesso em 25 out. 2014.
- [9] MODBUS ORGANIZATION; *MODBUS over Serial Line - Specification and Implementation Guide - V1.02*. Disponível em: <http://www.modbus.org/docs/Modbus_over_serial_line_V1_02.pdf>. Acesso em 18 set. 2014.
- [10] KAMAL, R.; *Embedded Systems*. Disponível em: <http://www.dauniv.ac.in/downloads/EmbsysRevEd_PPTs/Chap01Lesson_1Emsys.pdf>. Acesso em 28 out. 2014.
- [11] BARROS, E.; CAVALCANTE, S.; *Introdução aos Sistemas Embarcados*. Disponível em: <<http://www.cin.ufpe.br/~vba/periodos/8th/s.e/aulas/STP%20-%20Intro%20Sist%20Embarcados.pdf>>. Acesso em 28 out. 2014.

- [12] SINGLE BOARD COMPUTER; *The History of Single Board Computers*. Disponível em: <<http://www.singleboardcomputer.org/the-history-of-single-board-computers/>>. Acesso em 29 out. 2014.
- [13] BROWN, E.; *Top 10 Open Source Linux and Android SBCs*. Disponível em: <<http://www.linux.com/news/embedded-mobile/mobile-linux/773852-top-10-open-source-linux-and-android-sbcs>>. Acesso em 29 out. 2014.
- [14] RASPBERRY PI FOUNDATION; *Raspberry Pi Documentation*. Disponível em: <<http://www.raspberrypi.org/documentation/>>. Acesso em 27 out. 2014
- [15] ARM; *ARM Info Center*. Disponível em: <<http://infocenter.arm.com/help/index.jsp?topic=%2Fcom.arm.doc.subset.cortexm.m3%2Findex.html>>. Acesso em 30 out. 2014.
- [16] BROADCOM; *BCM2835ARM Peripherals*. Disponível em: <<http://www.raspberrypi.org/wp-content/uploads/2012/02/BCM2835-ARM-Peripherals.pdf>>. Acesso em 01 nov. 2014.
- [17] PEACOCK, C.; *USB in a NutShell – Making sense of the USB Standard*. Disponível em: <<http://www.beyondlogic.org/usbnutshell/usb2.shtml>>. Acesso em 03 nov. 2014.
- [18] MERCADOLIVRE; *Conversor USB to RS232 / RS485 / TTL (5V/3V3)*. Disponível em: <http://produto.mercadolivre.com.br/MLB-603514928-conversor-usb-p-rs232-rs485-ttl-5v3v3-3-em-1--_JM#redirectedFromParent>. Acesso em 06 jul. 2014.
- [19] MCCAULEY, M; *C library for Broadcom BCM 2835 as used in Raspberry Pi*. Disponível em <<http://www.airspayce.com/mikem/bcm2835/>>. Acesso em 18 set. 2014.
- [20] SIMPLY MODBUS; *Modbus ASCII vs Modbus RTU*. Disponível em: <<http://www.simplymodbus.ca/ASCII.htm>>. Acesso em 10 nov. 2014.
- [21] QT PROJECT; *QtSerialPort*. Disponível em: <<http://qt-project.org/wiki/QtSerialPort>>. Acesso em 19 out. 2014.
- [22] BLANCHETTE, J.; SUMMERFIELD, M.; *C++ GUI Programming with Qt4*, 2 ed Stoughton, Massachusetts, Prentice Hall, 2008.
- [23] QT PROJECTS; *Tutorial: apt-get install Qt4 on the Raspberry Pi*. Disponível em: <http://qt-project.org/wiki/apt-get_Qt4_on_the_Raspberry_Pi>. Acesso em 18 set. 2014.
- [24] QT; *Qt Documentation*. Disponível em: <<http://doc.qt.io/>>. Acesso em 20 set. 2014.