UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
ESCOLA DE ENGENHARIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

PROJETO DE DIPLOMAÇÃO

Módulo de I/O Remoto MODBUS

Guilherme Strack

Porto Alegre
Julho de 2011

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
ESCOLA DE ENGENHARIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

Módulo de I/O Remoto MODBUS

Projeto de Diplomação apresentado ao Departamento de Engenharia Elétrica da Universidade Federal do Rio Grande do Sul, como parte dos requisitos para Graduação em Engenharia Elétrica.

ORIENTADOR: Prof. Dr. Marcelo Götz

Porto Alegre
Julho de 2011

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
ESCOLA DE ENGENHARIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

Guilherme Strack

Módulo de I/O Remoto MODBUS

Este projeto foi julgado adequado para fazer jus aos créditos da Disciplina de "Projeto de Diplomação", do Departamento de Engenharia Elétrica e aprovado em sua forma final pelo Orientador e pela Banca Examinadora.

Orientador:

Prof. Dr. Marcelo Götz, UFRGS

Doutor em Informática pela Universität Paderborn – Paderborn, Alemanha

Banca Examinadora:

Prof. Dr. Luís Fernando Alves Pereira, UFRGS

Doutor em Ciência pelo ITA – São José dos Campos, Brasil

Engenheiro Eletricista Felipe Bresolin Zanon, Altus Sistemas de Informática S/A

Graduado em Engenharia Elétrica pela UFRGS – Porto Alegre, Brasil

Porto Alegre

Julho de 2011

AGRADECIMENTOS

Agradeço, primeiramente, a meus pais, Ademir Strack e Vera Teresinha Strack, e a minha irmã, Patrícia Strack. Sem o amor, o carinho, a educação e o apoio incondicional que essa família maravilhosa sempre me proporcionou com maestria, eu não estaria finalizando essa etapa tão importante de minha de vida.

Agradeço também a meus amigos fantásticos os momentos de descontração vividos a cada dia.

Agradeço a Altus Sistemas de Informática S/A, em especial ao Engenheiro Eletricista Felipe B. Zanon, a oportunidade e o apoio dados.

Agradeço, finalmente, a UFRGS, em especial ao Prof. Dr. Marcelo Götz, meu orientador neste trabalho, o suporte e a confiança proporcionados.

RESUMO

Este trabalho apresenta o desenvolvimento de um módulo de I/O que utiliza o protocolo MODBUS como forma de comunicação com outros dispositivos. Entre os tópicos descritos no presente documento, se encontram o estudo do protocolo de comunicação, a especificação do produto, a análise de alternativas, as etapas seguidas para atingir os objetivos propostos, os resultados obtidos e as propostas de melhoria para o módulo de I/O. O documento serve como base para quem deseja compreender o protocolo MODBUS e obter informações de como embarcá-lo em uma plataforma de hardware. Além disso, 2 leis de controle em malha fechada são apresentadas usando o I/O do módulo, um PLC e o protocolo MODBUS como forma de comunicação.

Palavras-chave: Módulo de I/O, protocolo MODBUS, protocolos de comunicação, automação e controle, software embarcado, microcontrolador.

ABSTRACT

This document presents the development of an I/O module that uses the MODBUS protocol as a communicating way with other devices. Among the topics described in this document, are the study of communication protocol, the product specification, the analysis of alternatives, the steps taken to achieve the proposed objectives, the results and the proposals for improvement in the I/O module. The document serves as base to understand the MODBUS protocol and how to embed it on a hardware platform. In addition, two control laws are presented in closed-loop using the I/O, a PLC and MODBUS protocol as way of communication.

Keywords: I/O module, MODBUS protocol, communication protocols, automation and control, embedded software, microcontroller.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIAÇÕES

ADC	Analog to Digital Converter
ADU	Application Data Unit
ASCII	American Standard Code for Information Interchange
CRC	Cyclic Redundancy Check
DTE	Data Terminal Equipment
EEPROM	Electrically Erasable Programmable Read Only Memory
HART	Highway Addressable Remote Transducer
HDLC	High Level Data Link Control
HMI	Human Machine Interface
IED	Intelligent Electronic Device
IP	Internet Protocol
ISO	International Organization for Standardization
I²C	Inter-Intergrated Circuit
I/O	Input/Output
LCD	Liquid Crystal Display
LED	Light Emitting Diode
LRC	Longitudinal Redundancy Check
MIPS	Millions of Instructions Per Second
OSI	Open System Interconnection
PDU	Protocol Data Unit
PID	Proporcional-Integral-Derivativo
PLC	Programmable Logic Controller
PWM	Pulse Width Modulation
RAM	Random Access Memory
RS	Recommended Standard
RTU	Remote Terminal Unit
SPI	Serial Peripheral Interface
TCP	Transmission Control Protocol
TTL	Transistor-Transistor Logic
UFRGS	Universidade Federal do Rio Grande do Sul
USB	Universal Serial Bus

1 INTRODUÇÃO

As redes de comunicação industriais vêm sendo cada vez mais utilizadas como sistema de comunicação entre sistemas de automação e dispositivos de campo. Nesse tipo de configuração, somente alguns fios são necessários para transmitir informações, que podem ser de Input/Output (I/O), parâmetros e diagnósticos, entre os diversos dispositivos presentes em um sistema automatizado.

Com a conexão de dispositivos em rede, o custo de instalação, configuração e manutenção de fiação é consideravelmente menor se comparado com o uso da tecnologia tradicional, que prevê que cada dispositivo tenha seu próprio conjunto de fios e seu próprio ponto de conexão com o dispositivo central.

Algumas vantagens do uso de redes industriais são citadas abaixo:

- Economia com fiação e com manutenção;
- Simplificação de projeto;
- Possibilidade de divisão de processamento entre os dispositivos;
- Possibilidade de uso de Intelligent Electronic Device (IED) que se comuniquem em rede.

1.1 Motivação

A automação está presente no cotidiano das pessoas e em diferentes níveis de atividades do homem, como, por exemplo:

- Na melhoria do conforto e da segurança de residências realizando o controle de luminosidade em ambientes, o controle de umidade e/ou o controle de acesso por biometria → Automação residencial;
- Na otimização de processos comerciais realizando a identificação de mercadorias por código de barras e/ou o controle de estoque → Automação comercial;
- Na otimização de um ou mais fatores relativos ao funcionamento de uma máquina ou processo industrial realizando assim a maximização da produção, a diminuição do consumo de energia elétrica, a diminuição da emissão de resíduos, a melhoria das condições de segurança e/ou a redução do esforço ou da interferência humana → Automação industrial.

Da mesma forma que a automação está fortemente presente na vida moderna, é natural encontrar equipamentos eletrônicos que possuam portas de comunicação externas.

Através delas é possível interligar esses dispositivos realizando trocas de informações entre eles. Muitas vezes, tais equipamentos são conectados utilizando o conceito de redes, trazendo ao sistema todas as vantagens que esse tipo de conexão possui.

É nesse contexto que a motivação para a realização do presente trabalho surge: a criação de um produto para a área de automação industrial que possua conectividade via redes de comunicação industriais.

Visto que a automação industrial está em pleno crescimento e que a conexão de equipamentos via rede está consolidada, o módulo de I/O remoto MODBUS é um produto totalmente de acordo com a realidade atual e futura do mercado.

1.2 Objetivo

Do exposto nas seções anteriores, o objetivo do presente trabalho é: o desenvolvimento de um módulo de I/O remoto MODBUS.

MODBUS se refere ao protocolo de comunicação utilizado entre o módulo e o dispositivo que realiza o controle do sistema, o Programmable Logic Controller (PLC).

Remoto vem do fato do módulo poder estar localizado distante geograficamente do PLC.

E módulo de I/O pode ser entendido com um equipamento eletrônico microcontrolado que realiza a leitura de determinados sinais elétricos provenientes de um sistema a ser automatizado e gera outros sinais elétricos para tal sistema.

Portanto, em outras palavras, módulo de I/O remoto MODBUS pode ser entendido como um dispositivo eletrônico que lê determinados sinais do sistema e gera outros sinais para o sistema, trocando essas informações com um PLC através de uma rede industrial utilizando o protocolo MODBUS, podendo estar longe geograficamente do PLC. A Figura 1.1 ilustra o funcionamento do produto.

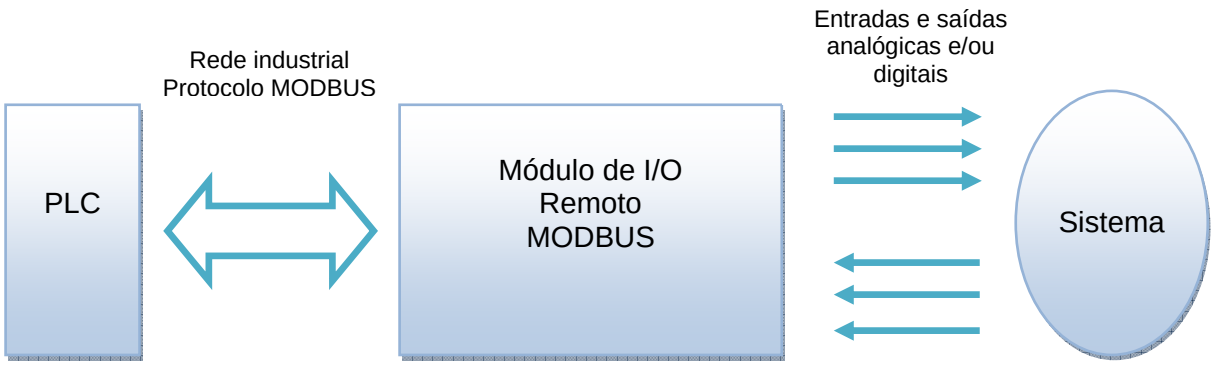

Figura 1.1 – Visão geral do produto.

Pelo fato do produto ser um sistema eletrônico microcontrolado, ele pode realizar algum tipo de processamento nos sinais antes de enviá-los ao PLC ou aplicá-los ao sistema, realizando assim a divisão do processamento com o PLC.

Como objetivo secundário, tem-se a aplicação de leis de controle em malha fechada no PLC utilizando os dados analógicos do módulo de I/O.

Maiores detalhes sobre as características e o funcionamento do módulo de I/O remoto MODBUS são encontrados nos capítulos subsequentes.

1.3 Organização do Trabalho

O trabalho está dividido da seguinte forma:

- Introdução: apresenta uma breve introdução sobre os assuntos a serem tratados no decorrer do trabalho assim como traz a motivação e a visão geral do funcionamento do produto desenvolvido;
- Fundamentação teórica: contextualiza o leitor sobre conceitos teóricos necessários para compreender o desenvolvimento do produto e entender seu funcionamento;
- Especificação e análise de alternativas: mostra a especificação do produto e avalia as possíveis alternativas técnicas para os diversos componentes utilizados no módulo ou em seu desenvolvimento;
- Desenvolvimento do projeto: expõe todas as etapas do desenvolvimento do produto;
- Resultados e conclusões: exibe os resultados obtidos e suas respectivas conclusões assim como as propostas de melhoria para o módulo.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são apresentados os conceitos teóricos necessários para o entendimento do desenvolvimento do produto, de seu funcionamento e de sua integração com outros sistemas.

O capítulo inicia com uma breve explicação sobre automação industrial e sua história. Em seguida, conceitos e topologias de redes de comunicação são apresentados. Definições sobre protocolos de comunicação e um modelo de arquitetura para redes de comunicação são mostrados na sequência. Por fim, o protocolo MODBUS é exposto em detalhes.

2.1 Automação Industrial

A automação, por incrível que pareça, existe desde a pré-história. Com a invenção da roda para transportar materiais, o esforço humano foi consideravelmente reduzido. Mas a automação começou a ser objeto de estudo e de investimento no século XVIII, em função das linhas de produção idealizadas por Henry Ford. Notou-se a necessidade de construção de máquinas para executar as tarefas com maior precisão, rapidez e qualidade aumentando assim a produtividade nas indústrias.

Portanto, pode-se concluir que o objetivo era sempre o mesmo, o de simplificar o trabalho do homem, de forma a substituir o esforço braçal por outros meios e mecanismos, liberando o tempo disponível para outros afazeres, valorizando o tempo útil para outras atividades (SILVEIRA & SANTOS, 1998).

Atualmente, entende-se por automação industrial qualquer sistema, apoiado em equipamentos eletrônicos, que substitua o trabalho humano em favor da segurança, da qualidade, da eficiência produtiva ou da redução de custos.

A automação industrial pode envolver a implementação de sistemas interligados e assistidos por redes de comunicação, compreendendo, entre outros, sistemas de supervisão para auxílio à operação.

Segundo (ROSÁRIO, 2005), a automação industrial pode ser entendida como uma tecnologia integradora de três áreas: a eletrônica responsável pelo hardware, a mecânica na forma de dispositivos mecânicos (sensores e atuadores) e a informática responsável pelo software, o qual irá controlar todo o sistema. Nota-se que uma ampla e diversificada formação é necessária para os profissionais que atuam nessa área.

A automação industrial pode ser dividida em níveis de abstração, conforme ilustra a Figura 2.1.

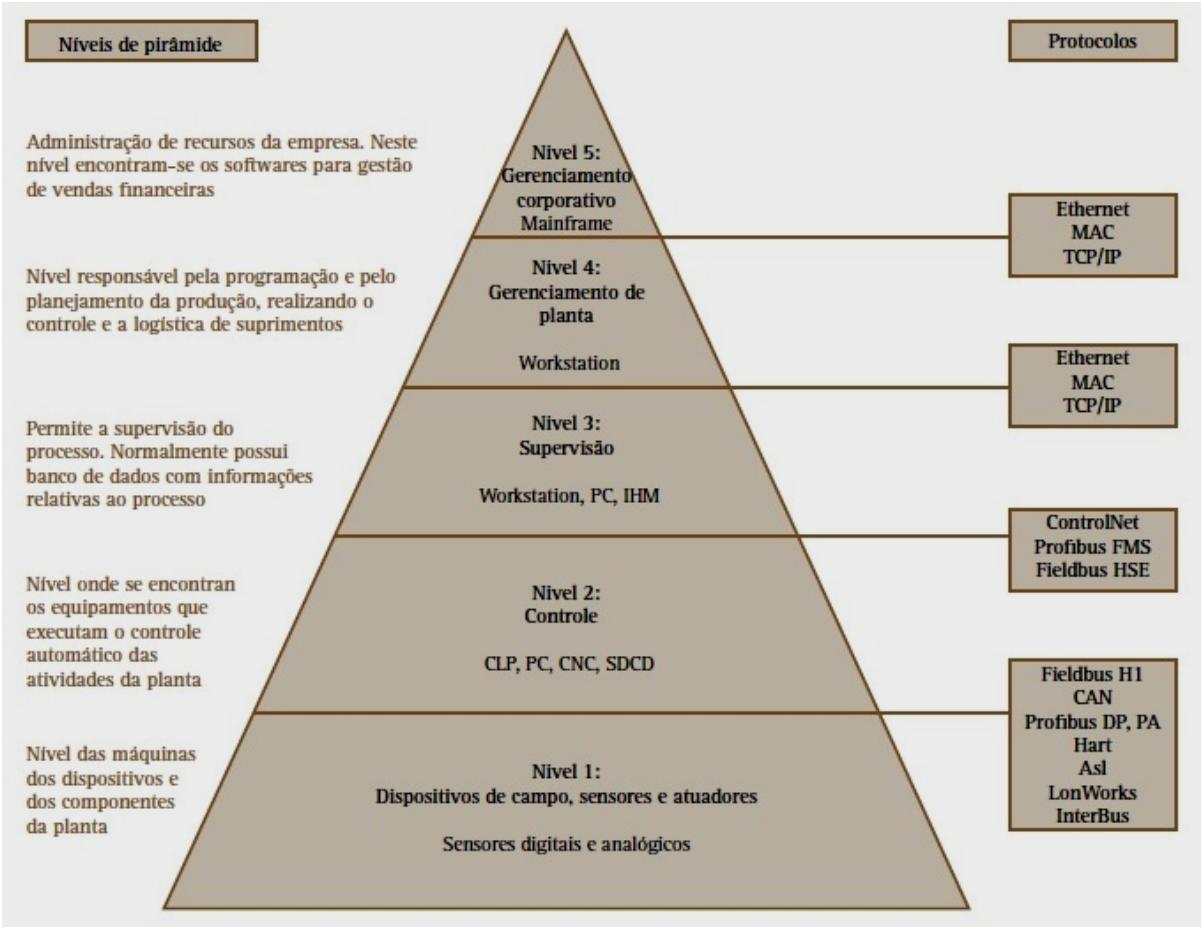

Figura 2.1 – Pirâmide da automação.

A troca de informações entre os níveis da pirâmide acontece via redes de comunicação utilizando protocolos de comunicação. A troca de dados entre o nível 1 (dispositivos de campo) e o nível 2 (controle) pode também ser feita conectando diretamente os dispositivos de campo nos pinos de I/O de um PLC.

Nota-se a importância dos diferentes tipos de protocolos de comunicação para que a troca de informações aconteça entre os diferentes níveis da pirâmide.

A Figura 2.2 mostra uma arquitetura típica de um sistema automatizado utilizando PLC da série Nexto da empresa Altus Sistemas de Informática S/A.

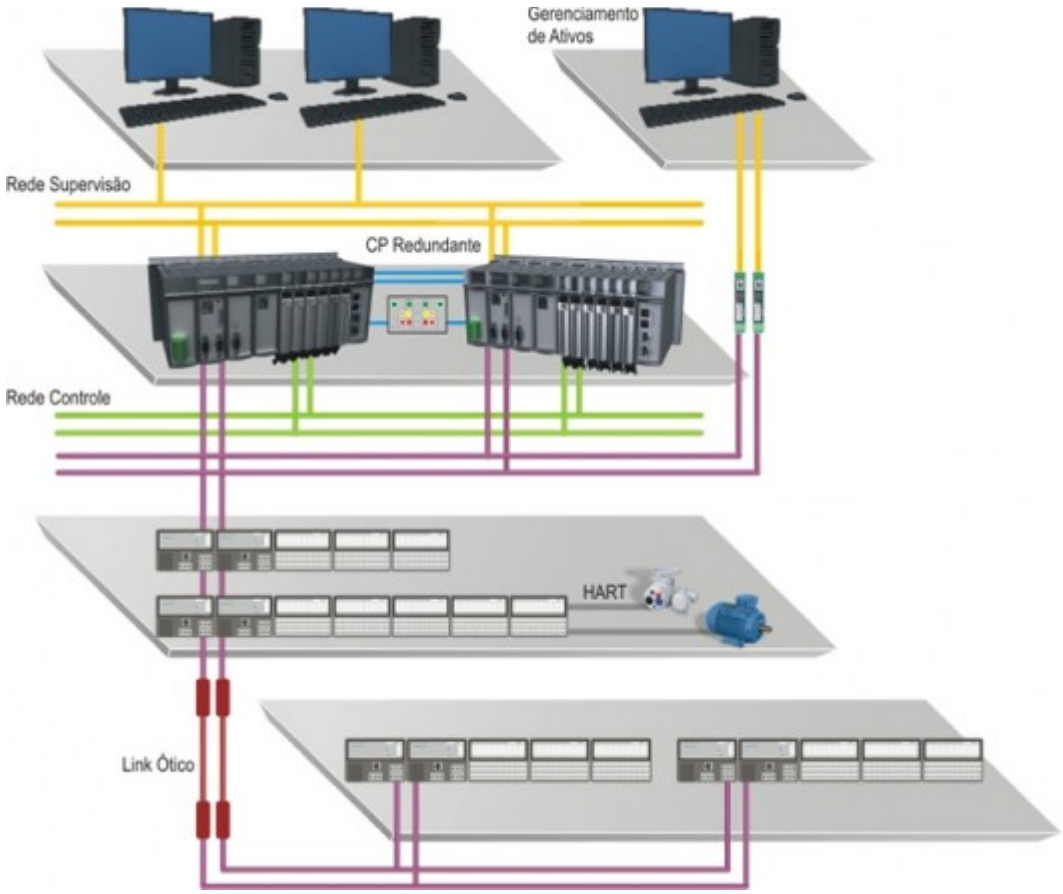

Figura 2.2 – Arquitetura típica de um sistema automatizado.

Percebe-se a clara divisão entre os níveis da pirâmide da automação: nível de campo, onde se encontram dispositivos como sensores e atuadores; nível de controle, onde se encontra o PLC com sua respectiva rede de controle; nível de supervisão, onde os equipamentos estão conectados na rede de supervisão e nível de gerenciamento da planta.

2.2 Redes de Comunicação

O conceito de redes de comunicação consiste em 2 ou mais equipamentos conectados entre si para que possam compartilhar informações. Os tipos de dados compartilhados dependem da aplicação e dos equipamentos utilizados.

Em uma rede de comunicação industrial a comunicação acontece normalmente da seguinte maneira:

- Tipicamente um dispositivo A realiza algum questionamento ou comando para um dispositivo B;
- O dispositivo B processa a informação e retorna uma resposta indicando sucesso ou não da solicitação;
- Caso o dispositivo B não responda, o dispositivo A finaliza a comunicação após certo período de tempo, conhecido como timeout.

O timeout pode ser resultado de algum dos problemas citados abaixo:

- Dispositivo questionado não existe;
- Dispositivo com problema;
- Parametrização na comunicação entre os dispositivos diverge.

Portanto, a comunicação acontece através de requisições e respostas. Um dispositivo requisita a um ou mais dispositivos e este(s) gera(m) resposta em função da requisição.

2.2.1 Topologias de Redes de Comunicação

As 3 topologias básicas de redes de comunicação são mostradas na Figura 2.3, na Figura 2.4 e na Figura 2.5.

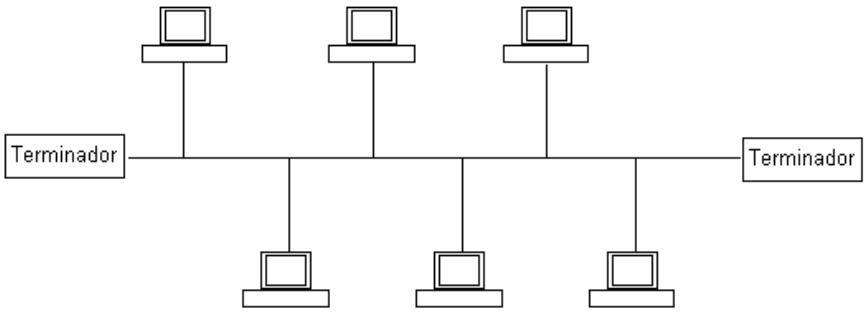

Figura 2.3 – Topologia barramento.

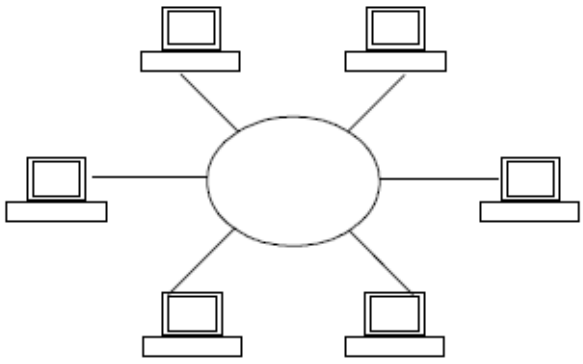

Figura 2.4 – Topologia anel.

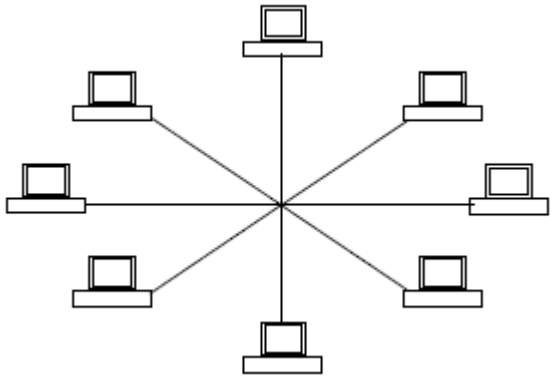

Figura 2.5 – Topologia estrela.

Na topologia barramento, os dados são enviados a todos os dispositivos da rede e somente o dispositivo destinatário processa a informação.

Na topologia anel, os dispositivos são ligados um após o outro em uma linha que se fecha em forma de anel. Pode-se entender essa rede como um barramento sem começo, nem fim.

Na topologia estrela, os dispositivos estão conectados por um ponto ou nó comum, conhecido como concentrador.

Existe ainda a topologia híbrida que consiste no uso de 2 ou mais topologias básicas em uma mesma rede.

Quando somente 2 dispositivos estão conectados para trocar informações, o tipo de ligação é conhecido como ponto a ponto. A Figura 2.6 mostra esse tipo de conexão.

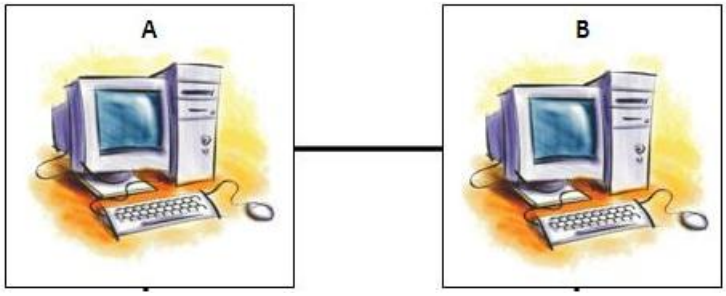

Figura 2.6 – Conexão ponto a ponto.

2.2.2 Modelo Mestre/Escravo

Em redes de comunicação, onde não há a necessidade de tráfego intenso de informações, utiliza-se geralmente a figura de um gestor de rede, ou mestre, que é responsável por iniciar toda e qualquer transmissão. O dispositivo que fica disponível para consulta e que responde às requisições do mestre é chamado de escravo. A Figura 2.7 e a Figura 2.8 ilustram tal tipo de estrutura.

Na Figura 2.7 o mestre inicia a comunicação enviando uma requisição para o escravo 3. Já na Figura 2.8 o escravo 3 reconhece que a requisição é para ele e responde ao mestre.

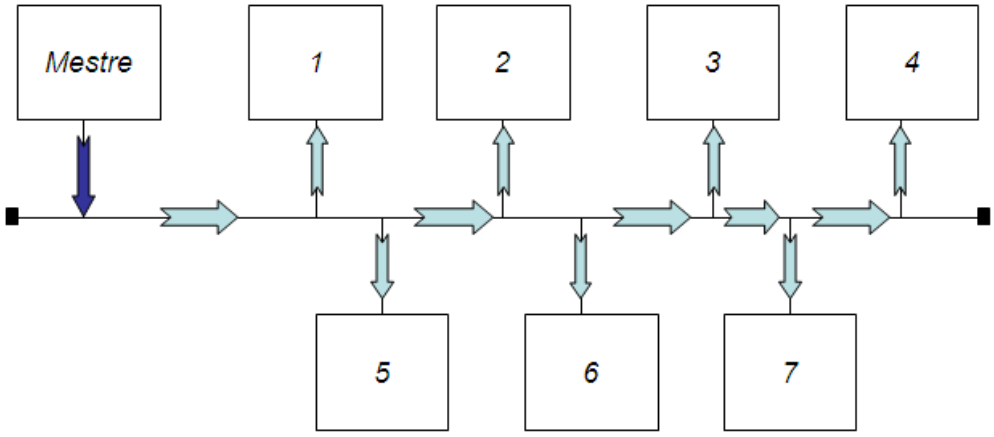

Figura 2.7 – Envio de requisição pelo mestre.

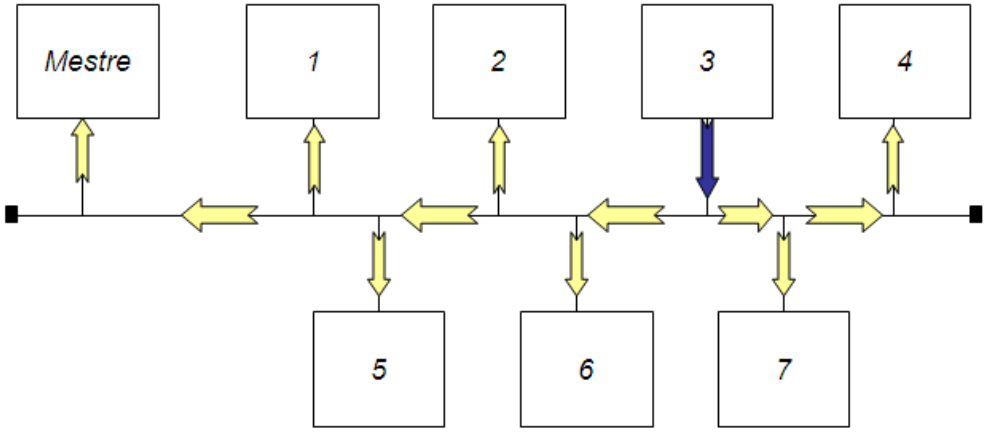

Figura 2.8 – Envio de resposta pelo escravo.

Nota-se que um único mestre é utilizado na rede e que a topologia é do tipo barramento. Todos os escravos têm acesso às requisições, mas somente o escravo destinatário responde à requisição.

2.2.3 Protocolos de Comunicação

Um protocolo pode ser definido como sendo regras que governam a sintaxe, a semântica e a sincronização de dados em uma rede comunicação.

Protocolos definem os formatos, a ordem das mensagens enviadas e recebidas pelos dispositivos da rede, assim como as ações que devem ser tomadas na transmissão e na recepção das mensagens.

Assim, 2 ou mais equipamentos, para se comunicarem em uma rede, devem falar a mesma linguagem, ou seja, usar o mesmo protocolo. Para existir comunicação é necessário existir pelo menos um canal, um emissor e um receptor e a garantia de que ambos utilizem um protocolo comum.

Os protocolos de comunicação podem ser classificados em abertos e fechados.

2.2.3.1 Protocolos Abertos ou Públicos

São protocolos escritos em padrões industriais. Esse tipo de protocolo é público, ou seja, pode ser adotado por qualquer fabricante de hardware ou software, cujas regras e convenções são amplamente divulgadas, geralmente na forma de uma norma técnica internacional, nacional ou regional.

2.2.3.2 Protocolos Fechados ou Proprietários

São protocolos definidos por uma empresa cujas regras não são disponibilizadas aos usuários e a outros fabricantes de dispositivos, fazendo com que somente dispositivos da empresa em questão possam se comunicar.

A Tabela 2.1 mostra alguns protocolos utilizados em automação industrial.

Tabela 2.1 – Exemplos de protocolos.

Protocolos	
Abertos	**Fechados**
MODBUS	Alnet 1 v1.00
PROFIBUS	Alnet II
HART	Alnet II/TCP
Ethernet Industrial	FATEK
DeviceNet	

2.2.4 Modelo OSI

Com o objetivo de facilitar o processo de padronização e obter interconectividade entre máquinas de diferentes fabricantes, a International Organization for Standardization (ISO), aprovou, no início da década de 1980, um modelo de arquitetura para sistemas abertos, visando permitir a comunicação e a construção de redes de comunicação independente da tecnologia de implementação.

Esse modelo foi denominado Open Systems Interconnection (OSI), servindo de base para a implementação de qualquer tipo de rede.

A arquitetura de uma rede é formada por camadas, interfaces e protocolos. As camadas são processos, implementados por hardware ou software, que se comunicam com o processo correspondente na outra máquina. Cada camada oferece um conjunto de serviços ao nível superior, usando funções realizadas no próprio nível e serviços disponíveis nos níveis inferiores.

Nesse tipo de estrutura, os dados a serem transmitidos no transmissor descem as camadas até o nível inicial, onde são transmitidos, para depois subir cada nível na máquina receptora. Com exceção da camada mais alta, cada camada é usuária dos serviços prestados pela camada imediatamente inferior (n-1) e presta serviços para a camada imediatamente superior (n+1). A Figura 2.9 mostra a arquitetura do modelo OSI.

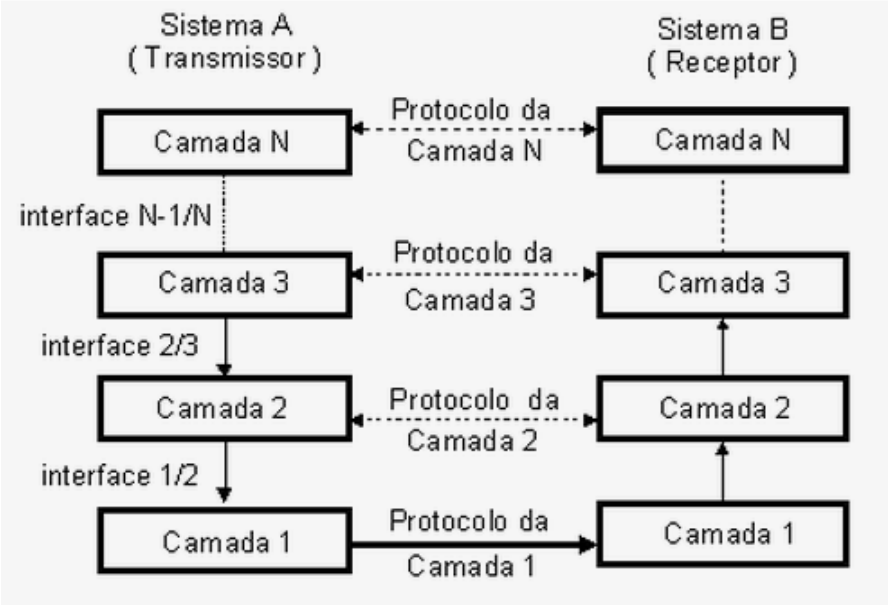

Figura 2.9 – Arquitetura do modelo OSI.

A Tabela 2.2 mostra as 7 camadas presentes no modelo OSI e resume suas principais funções.

Tabela 2.2 – Camadas do modelo OSI.

	Camada	Funções
7	Aplicação	Funções especializadas.
6	Apresentação	Formatação de dados e conversão de caracteres e códigos. Assegurar que a informação seja transmitida de tal forma que possa ser entendida e usada pelo receptor.
5	Sessão	Negociação e estabelecimento de conexão com outro nó. Administrar e sincronizar diálogos entre dois processos de aplicação.
4	Transporte	Meios e métodos para a entrega de dados ponta-a-ponta. Criar conexões para cada requisição vinda do nível superior, multiplexar as várias requisições vindas da camada superior em uma única conexão de rede, dividir as mensagens em tamanhos menores, a fim de que possam ser tratadas pelo nível de rede e estabelecer e terminar conexões através da rede.

3	Rede	Roteamento de pacotes através de uma ou várias redes. Controlar congestionamento e a contabilizar o número de pacotes ou bytes utilizados pelo usuário para fins de tarifação.
2	Enlace	Detecção e correção de erros introduzidos pelo meio de transmissão. Receber/transmitir uma seqüência de bits do/para o nível físico e transformá-los em uma linha que esteja livre de erros de transmissão, a fim de que essa informação seja utilizada pelo nível de rede.
1	Física	Transmissão dos bits através do meio de transmissão. Trabalha basicamente com características elétricas e mecânicas do meio físico: Voltagem para representar 0 (zero) ou 1, velocidade máxima de transmissão, número de pinos do conector, diâmetro dos condutores.

Apesar do modelo OSI ser dividido em 7 camadas, pode-se considerar genericamente que as 3 camadas mais baixas cuidam dos aspectos relacionados à transmissão propriamente dita, a quarta camada lida com a comunicação, enquanto que as três camadas superiores se preocupam com os aspectos relacionados à aplicação, já a nível de usuário.

Cabe ressaltar que nem todas as 7 camadas precisam ser necessariamente implementadas em uma determinada rede. Algumas unem as funções de duas camadas em uma (camadas mais altas) e outras dividem (camadas mais baixas).

Além disso, é importante salientar que o modelo OSI é simplesmente um modelo que especifica as funções a serem implementadas em uma rede, dando liberdade para que empresas e organizações desenvolvam tais funções.

2.3 Protocolo MODBUS

O protocolo MODBUS foi desenvolvido pela Modicon Industrial Automation Systems, hoje Schneider Electric. Sua padronização ocorreu em 1979 e é um dos mais antigos protocolos utilizados atualmente. É um protocolo aberto desde a sua padronização. Também possui fácil operação e manutenção, o que o torna uma solução de baixo custo. Tais requisitos o colocam entre os protocolos industriais de maior utilização no mercado mundial.

O padrão MODBUS define um protocolo de mensagens na camada de aplicação, posicionado na camada 7 do modelo OSI, que proporciona comunicação cliente/servidor entre dispositivos conectados em diferentes tipos de barramentos ou redes. Esse padrão também especifica um protocolo de comunicação serial para requisições entre um mestre e um ou mais escravos.

É um protocolo de requisição/resposta e oferece serviços especificados por funções. Os códigos de funções MODBUS são elementos da Protocol Data Unit (PDU).

O protocolo é atualmente implementado usando:

- Transmission Control Protocol (TCP)/Internet Protocol (IP) sobre Ethernet;

- Transmissão serial assíncrona sobre variados meios físicos (Recommended Standard (RS)-232, RS-422, RS-485);
- MODBUS+ (rede de alta velocidade baseada em token passing).

A Figura 2.10 ilustra a pilha de comunicação MODBUS.

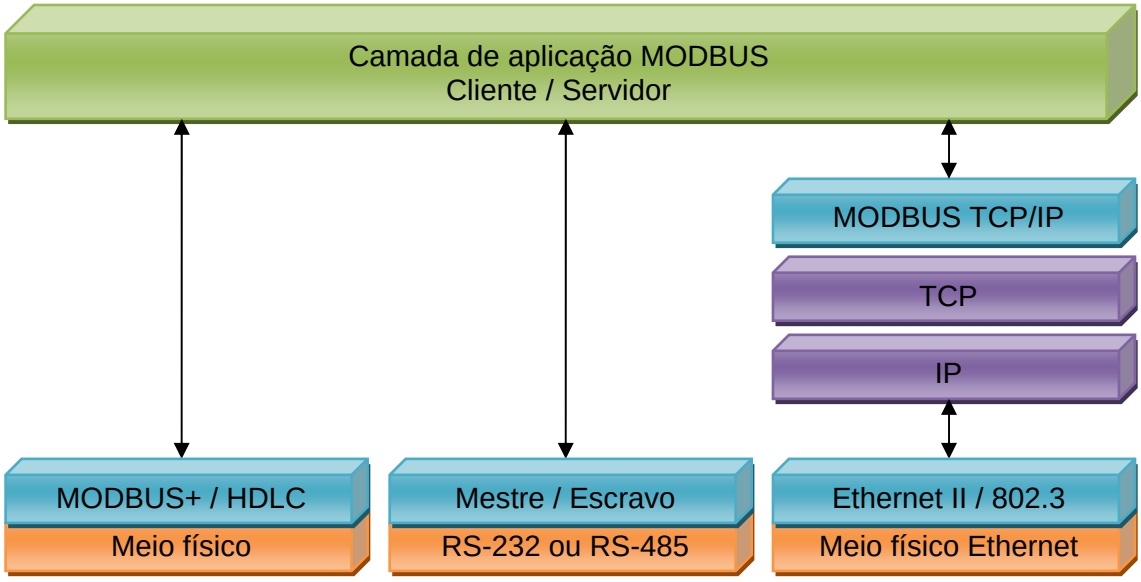

Figura 2.10 – Pilha de comunicação MODBUS.

Percebe-se a existência de um único protocolo para a camada de aplicação e diferentes protocolos para as camadas mais baixas.

No caso de implementação utilizando transmissão serial assíncrona, a pilha de comunicação MODBUS utiliza somente alguns protocolos nas camadas mais baixas, como mostra a Figura 2.11.

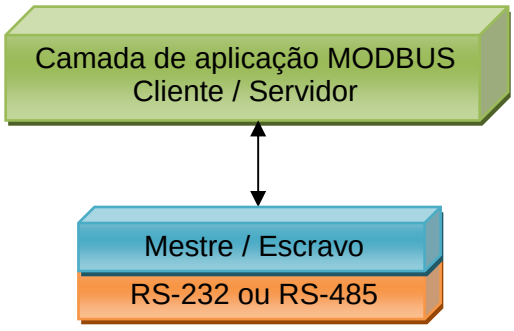

Figura 2.11 – Pilha de comunicação MODBUS utilizando transmissão serial assíncrona.

A comparação entre a pilha de comunicação MODBUS (Figura 2.11) e as 7 camadas propostas pela modelo OSI é mostrada na Tabela 2.3.

Tabela 2.3 – Comparação entre as camadas do modelo OSI e do protocolo MODBUS.

Camada	Modelo OSI	Protocolo MODBUS
7	Aplicação	Protocolo MODBUS de aplicação
6	Apresentação	Vazio
5	Sessão	Vazio
4	Transporte	Vazio
3	Rede	Vazio
2	Enlace	Protocolo MODBUS em linha serial
1	Física	RS-232, RS-422 ou RS-485

As seções subsequentes tratam dos protocolos MODBUS posicionados nas camadas 7 e 2 do modelo OSI.

2.3.1 Protocolo MODBUS de Aplicação

O protocolo MODBUS de aplicação permite fácil comunicação entre diversos dispositivos, como mostra a Figura 2.12. Ele está posicionado na camada 7 (Aplicação) do modelo OSI, como mostra a Tabela 2.3.

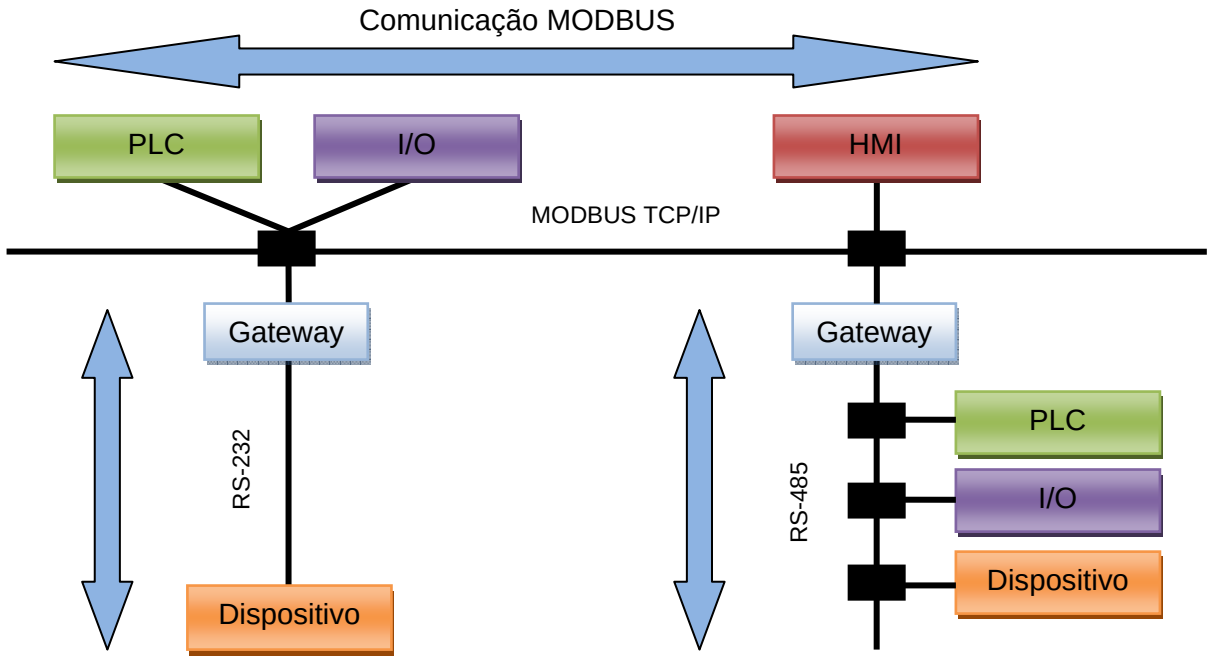

Figura 2.12 – Exemplo de arquitetura de rede MODBUS.

O mesmo protocolo de aplicação pode ser utilizado tanto em implementações em linhas seriais como também sobre Ethernet TCP/IP. Gateways permitem a comunicação entre diversos tipos de barramentos ou redes usando o protocolo MODBUS.

2.3.1.1 Descrição do Protocolo

O protocolo MODBUS de aplicação define uma simples Protocol Data Unit (PDU) independente das outras camadas utilizadas na comunicação. O mapeamento do protocolo MODBUS de aplicação para específicos barramentos ou redes pode introduzir alguns campos adicionais na Application Data Unit (ADU). A Figura 2.13 mostra um frame genérico MODBUS.

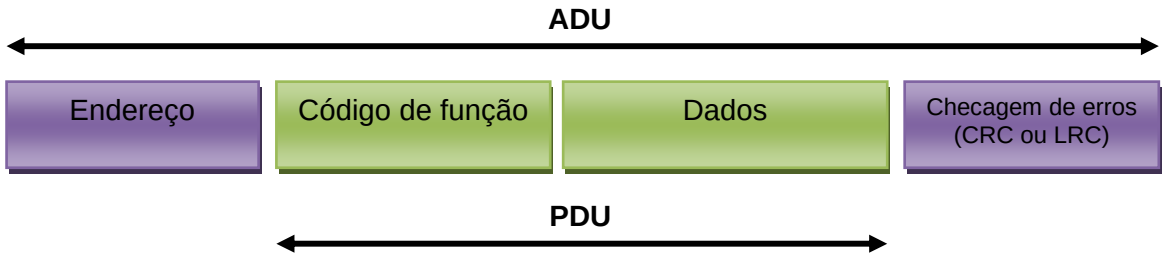

Figura 2.13 – Frame genérico MODBUS.

A ADU é construída pelo cliente que inicia a transação MODBUS. O código de função indica para o servidor qual tipo de ação este deve tomar. O protocolo MODBUS de aplicação estabelece o formato da requisição iniciada pelo cliente.

O campo de código de função é codificado em 1 byte. Códigos válidos estão na faixa de 1 a 255 decimal. A faixa de 128 a 255 é reservada e usada para exception responses. O código de função 0 (zero) não é válido.

O campo de dados das mensagens enviadas pelo cliente contém informações adicionais que o servidor usa para tomar determinadas ações definidas pelo código de função. O campo de dados pode não existir (tamanho zero bytes) em certos tipos de requisição. Nesse caso, o código da função sozinho especifica a ação.

Se nenhum erro ocorrer em uma comunicação MODBUS, o campo de dados da ADU da resposta enviada pelo servidor contém os dados requisitados. Na ocorrência de algum erro, o campo de dados da ADU da resposta contém um exception code. A Figura 2.14 e a Figura 2.15 ilustram os casos recém citados.

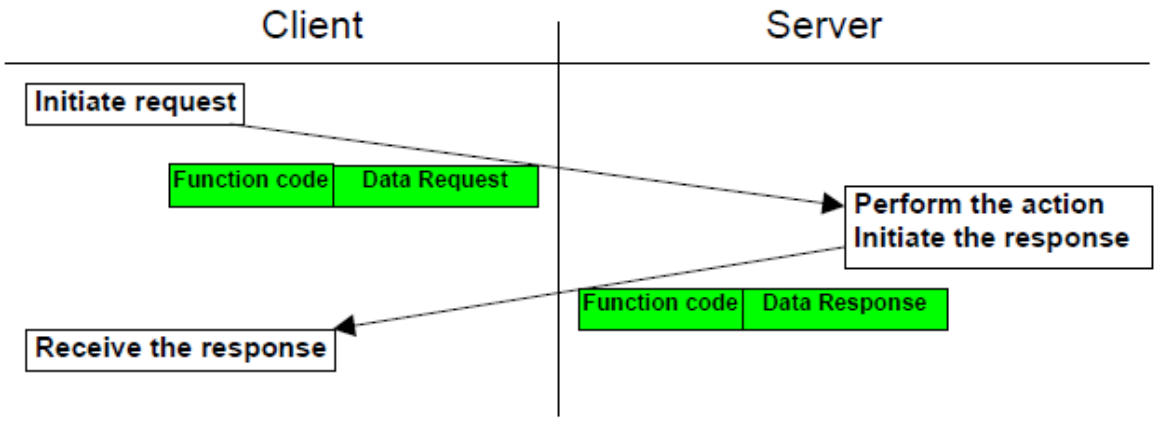

Figura 2.14 – Comunicação MODBUS sem erros.

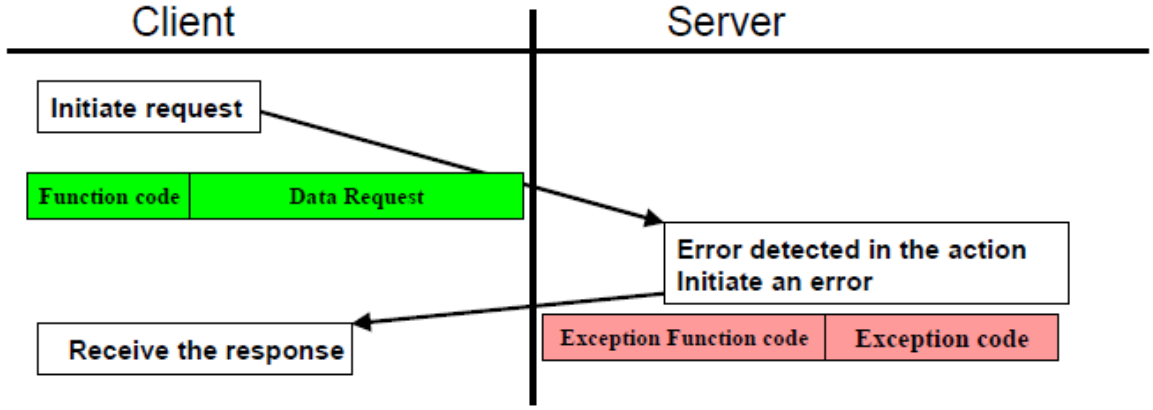

Figura 2.15 – Comunicação MODBUS com erros.

Quando o servidor responde para o cliente, ele usa o campo de código de função para indicar uma resposta normal (sem erros) e também para indicar se algum tipo de erro ocorreu (exception response). Para uma resposta normal, o servidor simplesmente copia para a resposta o código da função original, vide Figura 2.14. Já para uma exception response, o servidor retorna um código de função equivalente ao original, mas com o bit mais significativo em 1, vide Figura 2.15.

O tamanho máximo da PDU MODBUS é limitado pelo tamanho inerente da primeira implementação MODBUS em linha serial: máximo ADU RS-485 = 256 bytes. Portanto:

- PDU MODBUS em linha serial = 256 bytes – endereço servidor (1 byte) – CRC (2 bytes) = 253 bytes.

Consequentemente:

- RS-232/RS-485 MODBUS ADU = 253 bytes + endereço servidor (1 byte) + CRC (2 bytes) = 256 bytes.

O protocolo MODBUS de aplicação define 3 PDU:

- MODBUS Request PDU;

- MODBUS Response PDU;
- MODBUS Exception Response PDU.

2.3.1.2 Modelo de Dados MODBUS

O protocolo MODBUS de aplicação baseia seu modelo de dados numa série de tabelas que possuem características distintas. Tais características são mostradas na Tabela 2.4.

Tabela 2.4 – Características das tabelas MODBUS.

Tabela	Tipo de Objeto	Leitura e/ou Escrita	Comentários
Discretes Inputs	1 bit	Somente leitura	Pode ser gerado por um dispositivo de I/O
Coils	1 bit	Leitura e escrita	Pode ser alterado por um programa de aplicação
Input Registers	1 word (16 bits)	Somente leitura	Pode ser gerado por um dispositivo de I/O
Holding Registers	1 word (16 bits)	Leitura e escrita	Pode ser alterado por um programa de aplicação

Para cada uma das tabelas, o protocolo permite seleção individual de 65536 itens.

Os dados manipulados via MODBUS devem estar localizados na memória do dispositivo de aplicação. Endereços físicos não devem ser confundidos com a referência dos dados, que é usada nas funções MODBUS. A referência lógica é do tipo inteiro sem sinal começando em 0 (zero).

A Figura 2.16 mostra uma possível maneira de organização de dados em um dispositivo. Outras maneiras são também possíveis e cada dispositivo pode ter sua própria organização de acordo com a aplicação.

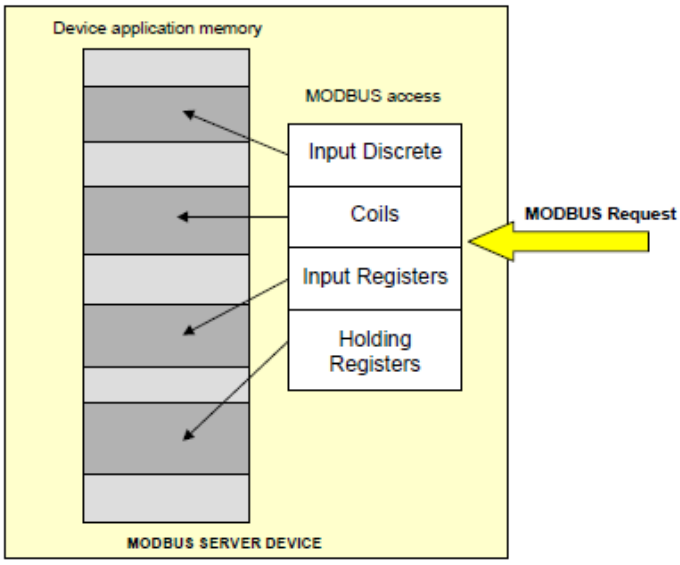

Figura 2.16 – Modelo de dados com separação dos blocos.

Nota-se a separação dos blocos na memória do dispositivo. Os dados dos diferentes blocos não possuem correlação e cada bloco é acessível por diferentes funções MODBUS.

2.3.1.3 Modelo de Endereçamento MODBUS

O protocolo MODBUS de aplicação define precisamente regras de endereçamento da PDU: Em uma PDU MODBUS cada dado é endereçável de 0 a 65535.

Ele também define que cada um dos 4 blocos de dados são numerados de 1 a n: No modelo de dados MODBUS cada elemento dentro de um bloco de dados é numerado de 1 a n.

O endereçamento de dados MODBUS é mostrado na Figura 2.17.

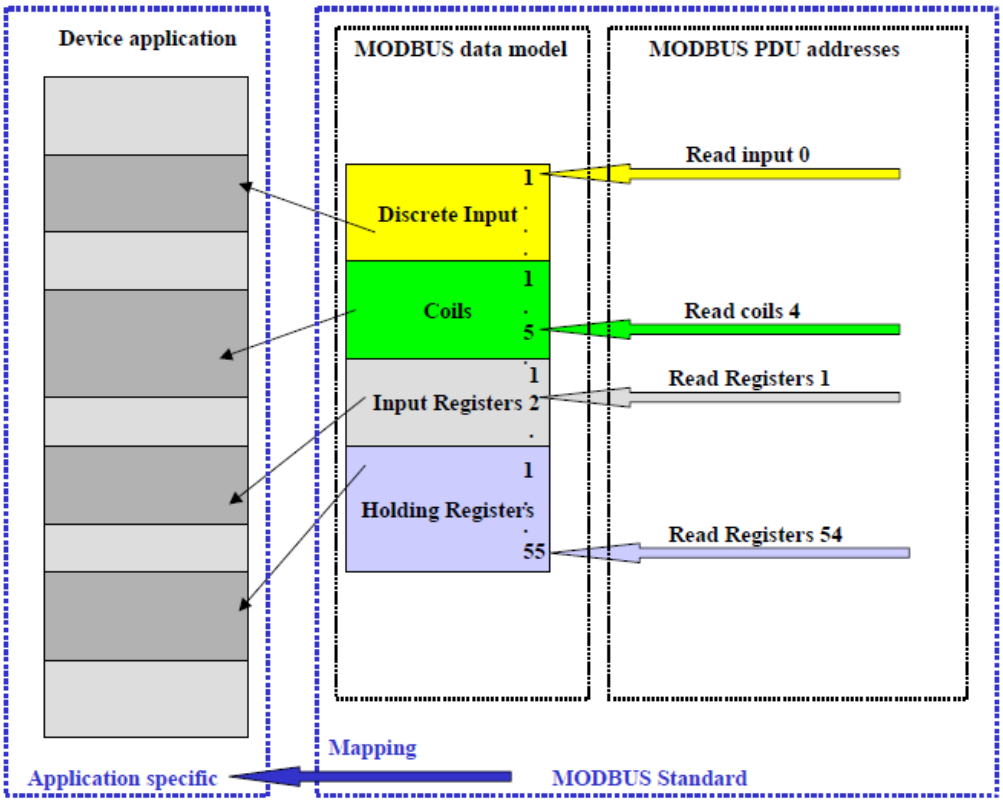

Figura 2.17 – Modelo de endereçamento MODBUS.

Um dado MODBUS numerado como X é endereçável em uma PDU MODBUS como (X – 1).

2.3.1.4 Códigos de Funções

Existem 3 categorias de funções MODBUS:

Funções públicas

- São funções bem definidas;
- Garantia de serem únicas;
- Validadas pela comunidade MODBUS-IDA.org;
- São publicamente documentadas.

Funções definidas pelo usuário

- Existem 2 faixas para esse tipo de função: 65 a 72 e 100 a 110 decimal;
- Usuário pode implementar uma função que não é suportada pela especificação;
- Não existe garantia de unicidade;

Funções reservadas

- Funções normalmente usadas em produtos de algumas empresas onde não é permitido o uso público.

A Figura 2.18 apresenta as categorias das funções assim como suas respectivas faixas de códigos de funções.

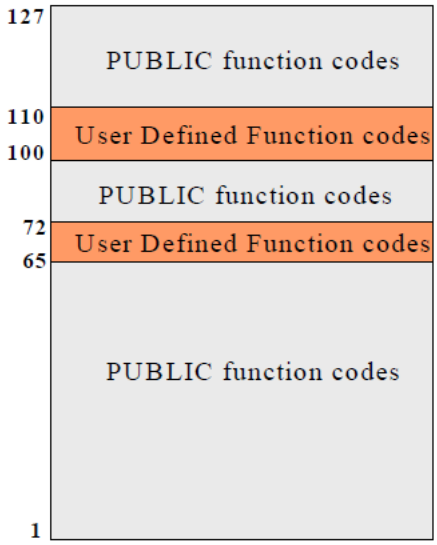

Figura 2.18 – Categorias das funções MODBUS.

Os principais códigos de funções MODBUS estão listados na Tabela 2.5.

Tabela 2.5 – Principais códigos de funções MODBUS.

Códigos de Funções		Descrição		
Código Decimal	**Código Hex**			
02	02	Leitura de Discrete Inputs	**Acesso a bit**	**Dados**
01	01	Leitura de Coils		
05	05	Escrita em uma única Coil		
15	0F	Escrita em múltiplas Coils		
04	04	Leitura de Input Registers	**Acesso a word (16 bits)**	
03	03	Leitura de Holding Registers		
06	06	Escrita em um único Holding Register		
16	10	Escrita em múltiplos Holding Registers		
07	07	Leitura de Exception status	**Diagnósticos**	
08	08	Várias funções de diagnóstico		

2.3.1.5 Exception Responses

Quando um cliente envia uma requisição para um servidor, 4 eventos podem ocorrer:

- Se o servidor receber a requisição sem nenhum erro de comunicação e manipular o pedido normalmente, ele retorna uma resposta normal;
- Se o servidor não receber a requisição devido a um erro de comunicação, nenhuma resposta é retornada. O programa cliente pode eventualmente determinar uma condição de timeout para não esperar a resposta indefinidamente;
- Se o servidor receber a requisição, mas detectar algum erro de comunicação (paridade, CRC ou LRC), nenhuma resposta é retornada. O programa cliente pode eventualmente determinar uma condição de timeout para não esperar a resposta indefinidamente;
- Se o servidor receber a requisição sem nenhum erro de comunicação, mas não poder manipular o pedido (requisição para ler uma coil não existente, por exemplo), o servidor retorna uma exception response para informar ao cliente a natureza do erro. Tal evento é ilustrado na Figura 2.15.

2.3.2 Protocolo MODBUS em Linha Serial

O protocolo MODBUS em linha serial é um protocolo mestre/escravo. Ele está posicionado na camada 2 (Enlace) do modelo OSI, como mostra a Tabela 2.3.

Um sistema mestre/escravo possui um nó mestre que emite comandos para um ou mais nós escravos e processa as respostas. Nós escravos normalmente não transmitem dados sem uma requisição do nó mestre e não se comunicam com outros nós escravos.

Como camada física, pode-se utilizar RS-232, RS-422 ou RS-485. A interface RS-485 de 2 fios é a mais comum. A interface RS-232 pode ser utilizada para comunicação ponto a ponto de curta distância.

No protocolo MODBUS em linha serial, o papel do cliente é proporcionado pelo mestre do barramento serial e os dispositivos escravos atuam como servidores.

2.3.2.1 Princípio do Protocolo MODBUS Mestre/Escravo

Somente um mestre pode estar conectado no barramento e um ou vários (máximo de 247) escravos podem também estar conectados na mesma linha serial. Uma comunicação MODBUS sempre é iniciada pelo mestre. Os escravos nunca transmitem sem receber uma requisição do mestre e também nunca se comunicam entre si. O mestre sempre inicia

somente uma transação MODBUS e aguarda o térmico desta para, então, iniciar uma nova transação.

O mestre emite requisições para os escravos de 2 maneiras:

- Modo unicast: o mestre endereça um escravo individualmente. Depois de receber e processar a requisição, o escravo retorna uma resposta para o mestre. Nesse modo, uma transação MODBUS consiste em 2 mensagens: uma requisição do mestre e uma reposta do escravo. Cada escravo possui um único endereço (1 a 247) para que possa ser endereçável independentemente dos outros escravos;

- Modo broadcast: o mestre envia uma requisição para todos os escravos. Nenhuma resposta é retornada, pois as requisições são necessariamente de escrita de comandos. Todos os dispositivos aceitam a mensagem broadcast. O endereço 0 (zero) é reservado para identificar uma troca broadcast.

O modo unicast e o modo broadcast são mostrados na Figura 2.19 e na Figura 2.20 respectivamente.

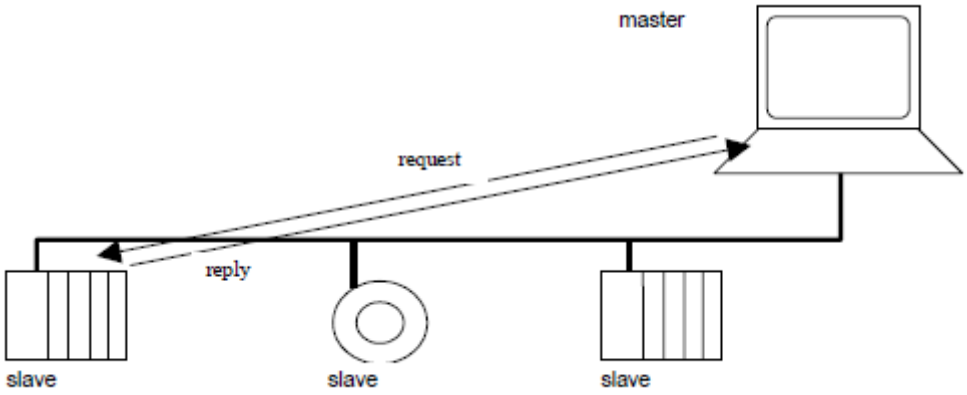

Figura 2.19 – Modo unicast.

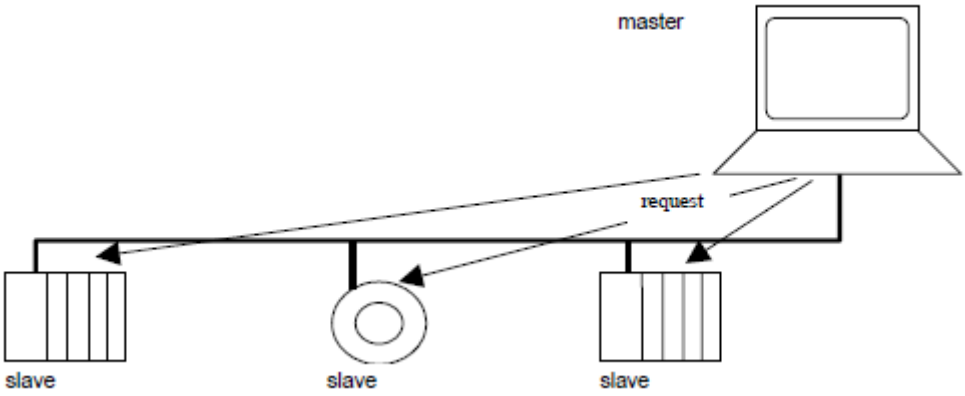

Figura 2.20 – Modo broadcast.

2.3.2.2 Regras do Endereçamento MODBUS

O espaço de endereços MODBUS compreende 256 diferentes endereços, como é mostrado na Tabela 2.6.

Tabela 2.6 – Espaço de endereços MODBUS.

0	1 ... 247	248 ... 255
Endereço broadcast	Endereços individuais dos escravos	Reservado

O endereço 0 (zero) é reservado como endereço broadcast. Todos os escravos devem reconhecer o endereço broadcast.

O mestre MODBUS não possui endereço específico. O endereço de cada escravo deve ser único em um barramento serial MODBUS.

O mestre endereça um escravo colocando seu respectivo endereço no campo endereço da ADU. Quando este escravo retorna a resposta, ele coloca seu próprio endereço no campo de endereço da resposta fazendo com que o mestre saiba qual escravo está respondendo.

2.3.2.3 Diagrama de Estados de Escravos

O diagrama de estados de escravos MODBUS é apresentado na Figura 2.21.

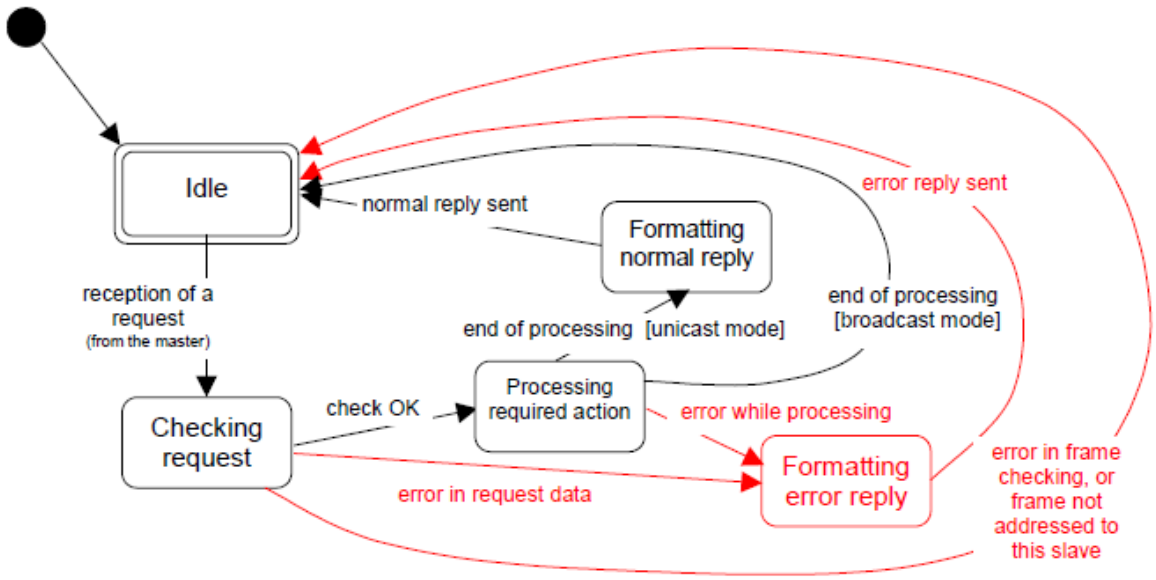

Figura 2.21 – Diagrama de estados de escravos MODBUS.

Da análise do diagrama da Figura 2.21, percebe-se:

- O estado Idle corresponde a nenhuma requisição pendente. Esse é o estado inicial após a inicialização do dispositivo;

- Quando uma requisição é recebida, o escravo checa os dados recebidos. Se ocorrer um erro de comunicação (paridade, CRC ou LRC) ou a requisição não é endereçada para ele, nenhuma resposta é enviada ao mestre. Na ocorrência de outros erros, uma resposta deve ser enviada ao mestre informando a natureza do erro;

- Uma vez completada a ação requerida, uma mensagem unicast é retornada ao mestre.

2.3.2.4 Diagrama de Tempo da Comunicação Mestre/Escravo

A Figura 2.22 mostra o diagrama de tempo de 3 casos típicos em comunicações mestre/escravo.

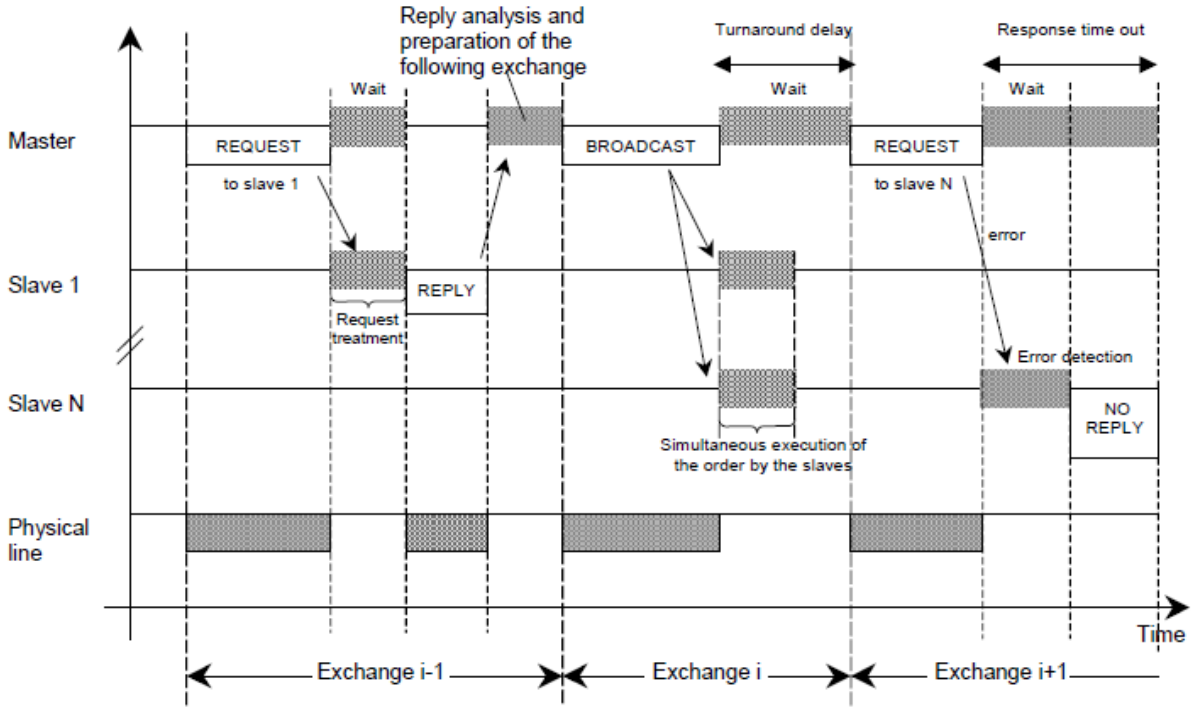

Figura 2.22 – Diagrama de tempo em comunicações mestre/escravo.

2.3.2.5 Modo de Transmissão RTU

Quando dispositivos se comunicam em uma linha serial MODBUS usando o modo de transmissão Remote Terminal Unit (RTU), cada byte da mensagem contém 2 caracteres hexadecimais de 4 bits cada. Esse modo de transmissão permite uma maior densidade de caracteres se comparado com o outro modo de transmissão, o American Standard Code for Information Interchange (ASCII), para o mesmo baud rate.

O formato de 11 bits para cada byte no modo RTU é:

- 1 start bit + 8 bits de dados + 1 bit de paridade + 1 stop bit.

Paridade par é necessária, mas outros modos (paridade ímpar ou sem paridade) também podem ser utilizados. Para garantir a compatibilidade com outros produtos, é recomendado suportar também o modo sem paridade.

Cada caractere ou byte é transmitido:

- Bit menos significativo ... bit mais significativo.

A Figura 2.23 e a Figura 2.24 mostra 2 casos de transmissão no modo RTU.

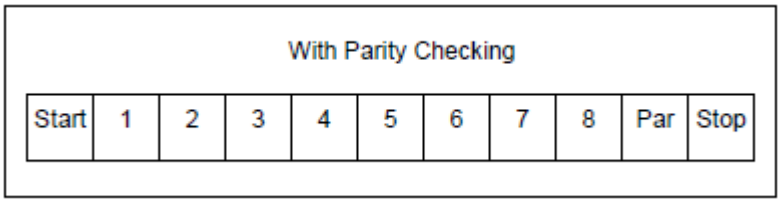

Figura 2.23 – Sequência de bits no modo RTU.

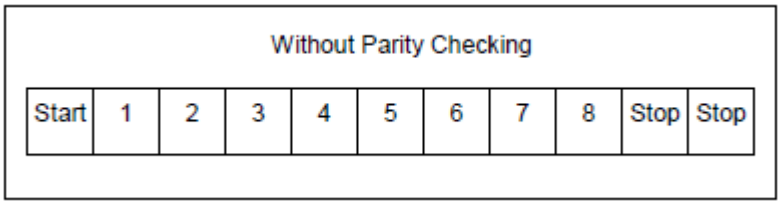

Figura 2.24 – Sequência de bits no modo RTU, caso sem paridade.

Nota-se que no caso sem paridade, tem-se 2 stop bits.

Já para o modo de transmissão ASCII, cada byte da mensagem é enviado como 2 caracteres ASCII. Exemplo: o byte 5B hexadecimal é enviado em 2 caracteres, 35 hexadecimal e 42 hexadecimal (0x35 = 5 e 0x42 = B em ASCII). O modo de transmissão ASCII é menos eficiente que o RTU, pois cada byte necessita de 2 caracteres.

2.3.2.6 Frame MODBUS no modo RTU

Uma mensagem MODBUS é colocada pelo transmissor em um frame que possui pontos de início e fim bem definidos. Assim, os dispositivos que recebem mensagens podem detectar o início das mensagens e quando elas são completadas.

No modo RTU, os frames das mensagens são separados por um silêncio na linha de no mínimo 3,5 caracteres. Um frame de mensagem MODBUS no modo RTU assim como o silêncio na linha são mostrados na Figura 2.25.

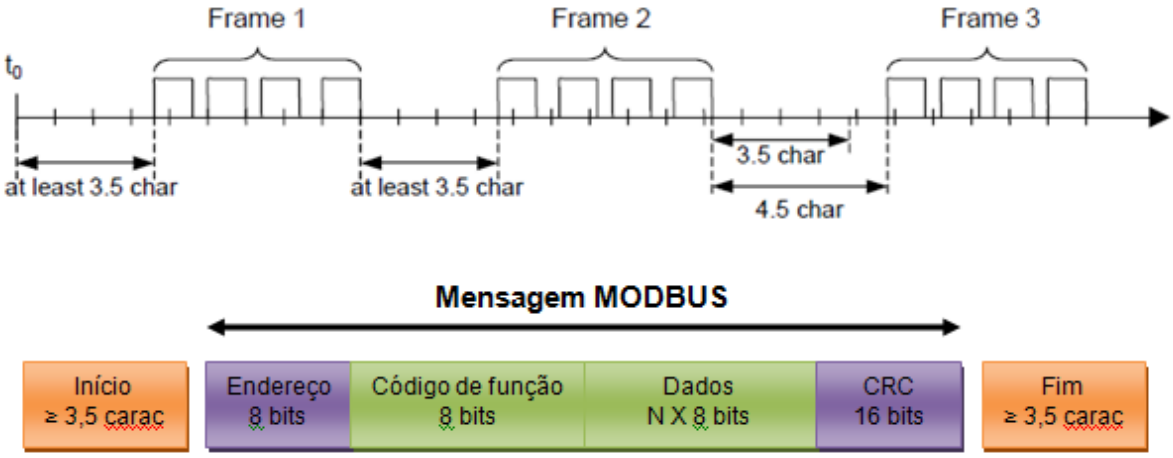

Figura 2.25 – Silêncio e mensagem MODBUS no modo RTU.

Se um silêncio maior que 1,5 caracteres ocorrer entre 2 caracteres, o frame de mensagem é considerado incompleto e deve ser descartado pelo receptor. Tal fato é ilustrado na Figura 2.26.

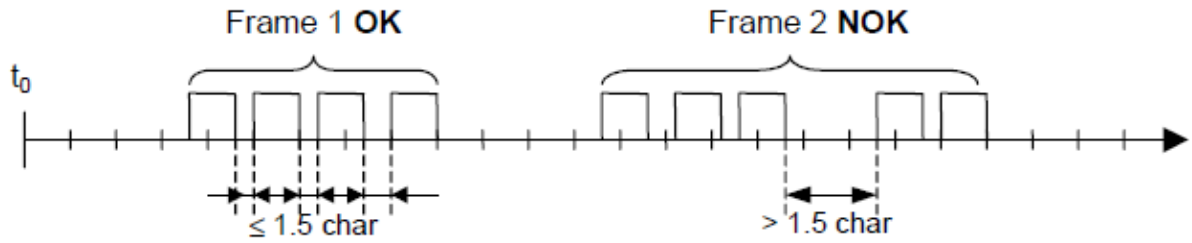

Figura 2.26 – Silêncio entre caracteres.

O diagrama de estados da transmissão RTU é mostrado na Figura 2.27. Ambos pontos de vista, do mestre e do escravo, são expressados.

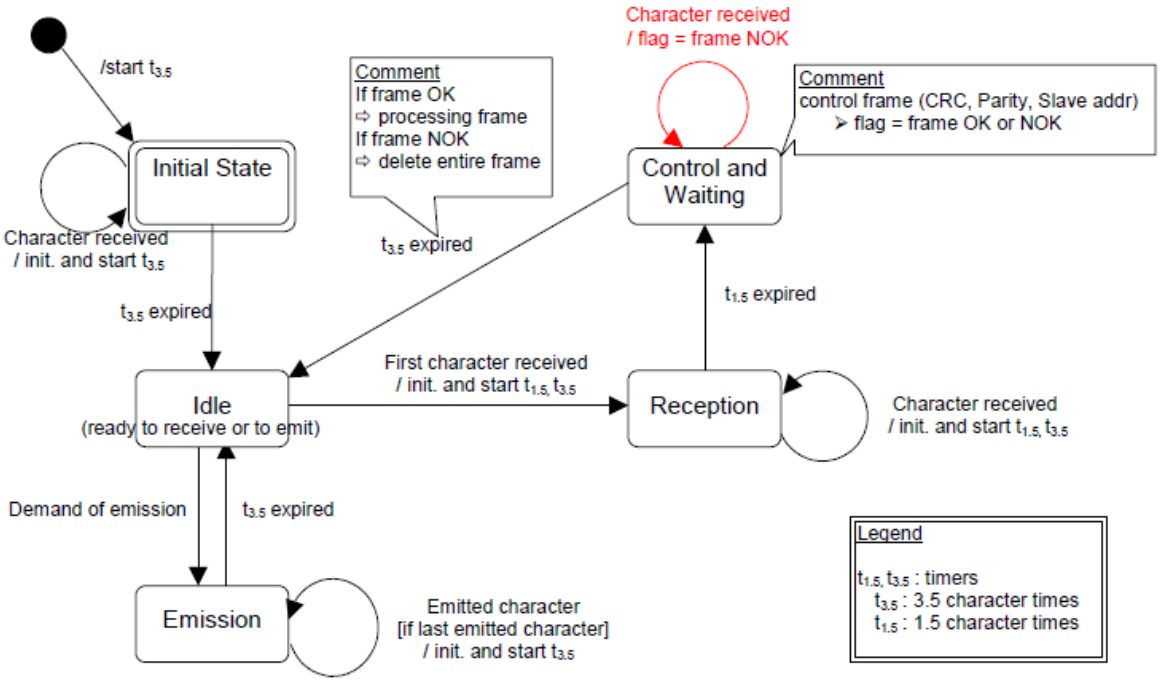

Figura 2.27 – Diagrama de estados da transmissão RTU.

Da análise do diagrama da Figura 2.27, percebe-se:

- A transação entre o Initial State para o Idle necessita de $t_{3,5}$ timeout;
- O estado Idle é o estado normal quando nenhuma emissão ou recepção está ativa;
- No modo RTU, a linha de comunicação é declarada no estado Idle quando não existem transmissões ativas depois de um intervalo de tempo maior ou igual a 3,5 caracteres;
- Quando a linha está no estado Idle, cada caractere transmitido detectado é identificado como início de um frame. A linha passa então para o estado ativo. O final do frame é identificado quando não existem mais caracteres para serem transmitidos depois de um intervalo de tempo de 3,5 caracteres;
- Depois do recebimento completo do frame, o cálculo do Cyclic Redundancy Check (CRC) é feito. Da mesma forma, o campo de endereço é analisado. Se o frame não é endereçado para o dispositivo, ele é descartado. Para reduzir o tempo de processamento da recepção, o campo de endereço pode ser analisado assim que ele for recebido, sem esperar o final do frame.

2.3.2.7 Checagem CRC

O modo RTU inclui um campo de checagem de erros baseado no CRC. O campo de CRC checa o conteúdo total da mensagem. Ele é aplicado sem considerar o bit de paridade dos caracteres da mensagem.

O campo de CRC está posicionado no final da mensagem e contém um valor de 16 bits divido em 2 bytes. O byte de mais alta ordem é transmitido por último.

O valor do CRC é calculado pelo dispositivo transmissor. O dispositivo receptor recalcula o CRC baseado nos bytes recebidos e o compara com o valor recebido no campo de CRC. Se esses valores não são iguais, um erro é gerado.

O processo para geração do CRC é citado abaixo:

1. Carregar um registrador de 16 bits com FFFF hexadecimal (todos bits em 1) chamado de registrador de CRC;

2. Realizar um OU EXCLUSIVO dos 8 bits da mensagem com os 8 bits menos significativos do registrador de CRC e colocar o resultado no registrador de CRC;

3. Deslocar com carry o registrador de CRC em 1 bit no sentido dos bits menos significativos e colocar 0 (zero) no bit mais significativo do registrador de CRC. Examinar o carry do deslocamento;

4. Se o carry é 0 (zero): Pular para o passo 5. Se o carry é 1: Realizar um OU EXCLUSIVO entre o registrados de CRC e o polinômio gerador de CRC A001 hexadecimal (1010 0000 0000 0001) e colocar o resultado no registrador de CRC;

5. Repetir os passos 3 e 4 até que 8 deslocamentos tenham sido feitos. Dessa forma, um byte completo é processado;

6. Repetir os passos 2 a 5 para o próximo byte da mensagem. Continuar até que todos os bytes tenham sido processados;

7. O conteúdo final do registrador de CRC é o valor do CRC;

8. Transmitir o byte de mais baixa ordem primeiro.

A Figura 2.28 mostra o fluxograma de um algoritmo que realiza o cálculo do CRC.

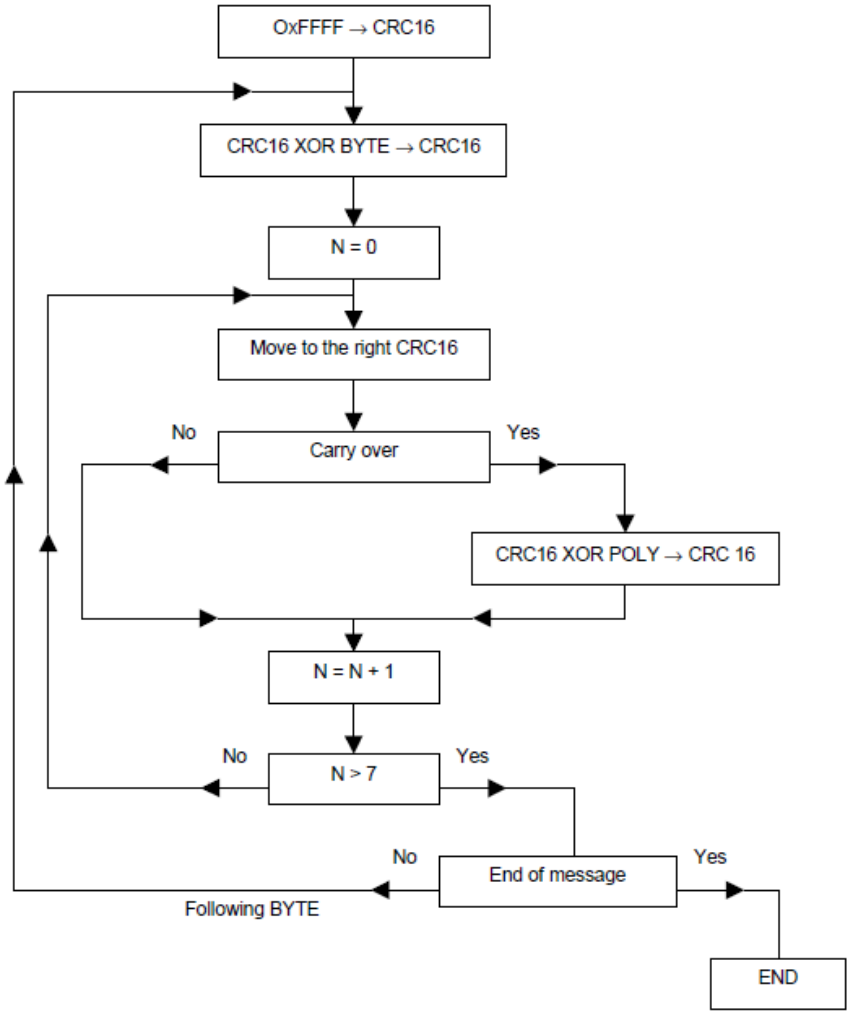

Figura 2.28 – Fluxograma para o cálculo do CRC.

Na Figura 2.28:

- XOR = OU EXCLUSIVO;
- N = número de deslocamentos;
- POLY = polinômio gerador de CRC = 1010 0000 0000 0001.

2.3.3 Padrão RS-232 Aplicado ao MODBUS

O padrão RS-232 é muito popular e tipicamente utilizado para conectar 2 equipamentos de comunicação de dados. Esse padrão cobre as características mecânicas, elétricas e funcionais da interface serial assíncrona.

Um sinal é considerado marca quando a tensão na linha é mais negativa que -3 V e é considerado espaço quando a tensão é mais positiva que +3 V. A região entre +3 V e -3 V é denominada região de transição e é considerada como nível inválido. Tensões acima de +15

V ou abaixo de -15 V também são consideradas inválidas. Tais faixas de tensão são ilustradas na Figura 2.29.

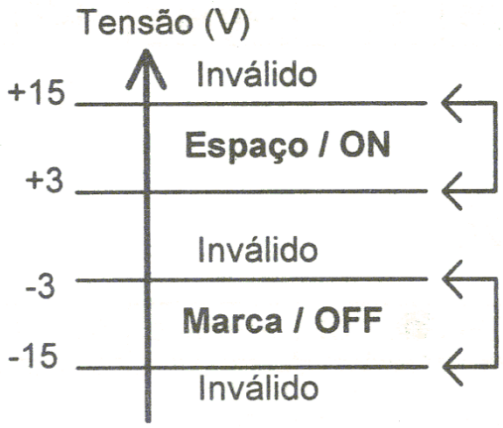

Figura 2.29 – Níveis de tensão válidos.

Durante uma transmissão de dados, a marca (-12 V) é usada para caracterizar o estado binário 1. Já o espaço (+12 V) é usado para caracterizar o estado binário 0 (zero). A Figura 2.30 mostra os níveis de tensão utilizados para marca e espaço de 2 pontos de vista diferentes, Transistor-Transistor Logic (TTL) e RS-232.

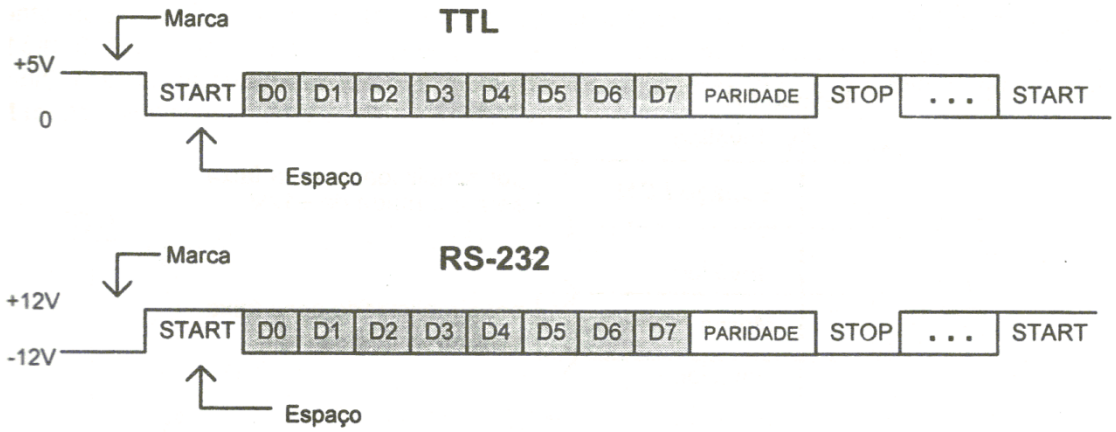

Figura 2.30 – Níveis tensão para TTL e RS-232.

Na grande maioria das aplicações práticas, é necessário utilizar circuitos que convertam os níveis TTL para os níveis exigidos pelo padrão RS-232 e vice-versa. Um dos circuitos mais populares desenvolvido para essa função de conversão de níveis lógicos é o MAX232, fabricado por empresas como Texas Instruments e MAXIM - Dallas Semiconductor.

O padrão define também 2 tipos de equipamentos:

- Data Terminal Equipment (DTE): equipamento terminal de dados, ou seja, está localizado nos extremos da comunicação. Os dados são gerados ou terminam num DTE;

- Data Communication Equipment (DCE): equipamento comunicador de dados, ou seja, dispositivo por onde passam os dados.

Os sinais presentes no padrão RS-232 na visão de um equipamento DTE são mostrados na Tabela 2.7.

Tabela 2.7 – Descrição dos sinais RS-232 na ótica de um DTE.

Pino Conector DB9	Sinal	Direção
3	TD: Transmitted Data	Output
2	RD: Received Data	Input
7	RTS: Request to Send	Output
8	CTS: Clear to Send	Input
6	DSR: Data Set Ready	Input
5	GND: Ground	-
1	CD: Carrier Detect	Input
4	DTR: Data Terminal Ready	Output
9	RI: Ring Indicator	Input

O protocolo MODBUS em linha serial exige 3 dos 9 sinais padronizados pelo RS-232:

- TD, RD e GND;

Os outros sinais são opcionais. O sinal TD de um dispositivo deve ser conectado ao sinal RD do outro dispositivo e vice-versa. Além disso, cabos tendo menos de 20 m de comprimento devem ser usados nas conexões.

3 ESPECIFICAÇÃO E ANÁLISE DE ALTERNATIVAS

Neste capítulo é realizada a especificação do módulo de I/O remoto MODBUS e também é feita a avaliação das possíveis alternativas técnicas para os diversos componentes utilizados no produto ou em seu desenvolvimento.

O capítulo inicia com a especificação do produto. Tal especificação tenta ser a mais abrangente possível, cobrindo os principais aspectos do módulo. Em seguida, a análise de alternativas é feita. Por fim, os componentes/dispositivos escolhidos são apresentados em maiores detalhes.

3.1 Especificação do Produto

O módulo de I/O remoto MODBUS deve:

- Atuar como um dispositivo escravo;
- Possuir, no mínimo: 2 entradas digitais, 2 saídas digitais, 2 entradas analógicas e 2 saídas analógicas que correspondem, respectivamente, a 2 Discrete Inputs, 2 Coils, 2 Input Registers e 2 Holding Register, como mostra a Tabela 2.4;
- Possuir modelo de dados com 4 blocos separados, como ilustra a Figura 2.16;
- Suportar, no mínimo, as funções: 01 (Leitura de Coils), 02 (Leitura de Discrete Inputs), 03 (Leitura de Holding Registers), 04 (Leitura de Input Registers), 05 (Escrita em uma única Coil) e 06 (Escrita em um único Holding Register);
- Reconhecer e responder a mensagens broadcast;
- Possuir endereço escravo configurável na faixa de 1 a 247 decimal;
- Realizar transmissão serial no modo RTU, com paridade par, a uma taxa de 9600 bps;
- Utilizar como meio físico o padrão RS-232 sendo um equipamento DCE.

A especificação do módulo é resumida na Tabela 3.1.

Tabela 3.1 – Especificação do produto.

Características	
Endereços	Escravo: Configurável entre 1 e 247
Broadcast	Sim
Baud rate	9600 bps
Paridade	Par
Modo	RTU
Meio físico	RS-232
Tipo de conector	DB9 fêmea

3.2 Análise de Alternativas

3.2.1 Protocolo de Comunicação

Existem diversos protocolos utilizados em automação industrial. Alguns deles são mostrados na Tabela 2.1.

Por motivos óbvios, os protocolos fechados tiveram seu uso logo descartado. O protocolo MODBUS foi escolhido pelos motivos citados abaixo:

- Foi padronizado em 1979, o que o torna uma solução bastante conhecida e consolidada no mercado;
- Possui fácil operação e manutenção;
- É uma solução de baixo custo;
- É um dos protocolos industriais de maior utilização no mercado mundial.

Outro fator importante na escolha do protocolo MODBUS se refere a pouca exigência de hardware necessária para sua implementação; com apenas um microcontrolador e uma interface serial RS-232 pode-se construir um dispositivo capaz de se comunicar utilizando o protocolo.

3.2.2 Microcontrolador

Como o módulo deve possuir uma interface serial e realizar a leitura de determinados sinais elétricos de um sistema e gerar outros sinais elétricos para tal sistema, sendo um dispositivo inteligente, o uso de um microcontrolador se mostrou adequado.

Existem atualmente no mercado 3 principais fabricantes: a Freescale Semiconductor, responsável pela fabricação de microcontroladores da família HCS08; a Microchip

Technology, responsável pela fabricação de microcontroladores da família PIC e a Atmel Corporation, responsável pela fabricação de microcontroladores da família 8051. A Tabela 3.2 mostra um comparativo entre microcontroladores das empresas acima citadas.

Tabela 3.2 – Comparativo entre microcontroladores.

	Freescale MC9S08JM60	**Microchip PIC18F4550**	**Atmel AT89C5132**
Memória Flash (kB)	60	32	64
Memória RAM (kB)	4	2	2
USB RAM (Bytes)	256	1024	256
EEPROM (Bytes)	-	256	2048
I/O	Até 51	Até 35	Até 44
ADC	Até 12 canais de 12 bits	Até 13 canais de 10 bits	Até 2 canais de 10 bits
SPI	2	1	1
I²C	Sim	Sim	Sim
USB	Device 2.0	Device 2.0	Device 1.1
MIPS	Até 12	Até 10	Até 1,66

O microcontrolador da Freescale foi escolhido por ter um elevado número de Millions of Instructions Per Second (MIPS) e por possuir canais de Analog to Digital Converter (ADC) de 12 bits.

Além disso, o ambiente de desenvolvimento fornecido pela Freescale, o CodeWarrior, oferece uma interface de fácil utilização para o usuário. Com o uso da ferramenta de design rápido de aplicação chamada de Device Initialization, é possível configurar todos os periféricos do microcontrolador de uma maneira gráfica além de gerar código para a tabela de vetores de interrupção. A Figura 3.1 mostra uma tela do CodeWarrior com o Device Initialization em primeiro plano.

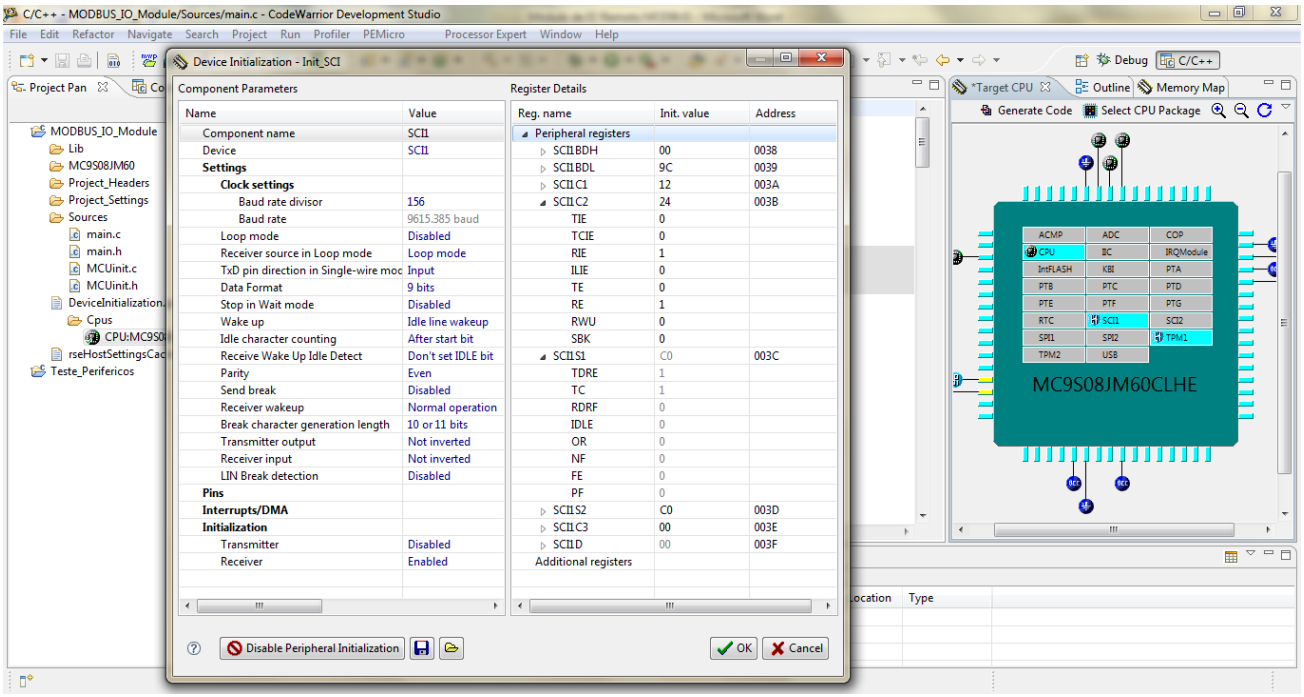

Figura 3.1 – CodeWarrior com a ferramenta Device Initialization.

3.2.3 Kit de Desenvolvimento

Realizada a escolha do microcontrolador, buscou-se uma solução para o hardware do produto. Foram selecionados 2 kits de desenvolvimento: o kit MagicFlexis, fabricado pela Microgenios e o kit DEMOJM, fabricado pela própria Freescale.

O kit MagicFlexis foi escolhido por proporcionar uma solução completa de hardware, não sendo necessário o desenvolvimento de hardware adicional durante a realização do trabalho. Outro diferencial do kit MagicFlexis é o gravador e depurador microBDM que acompanha o kit. Através dele, é possível gravar os projetos no microcontrolador assim como realizar a depuração in circuit. A Figura 3.2 mostra uma foto do gravador microBDM e a Figura 3.3 traz uma foto do kit MagicFlexis.

Figura 3.2 – Gravador e depurador microBDM.

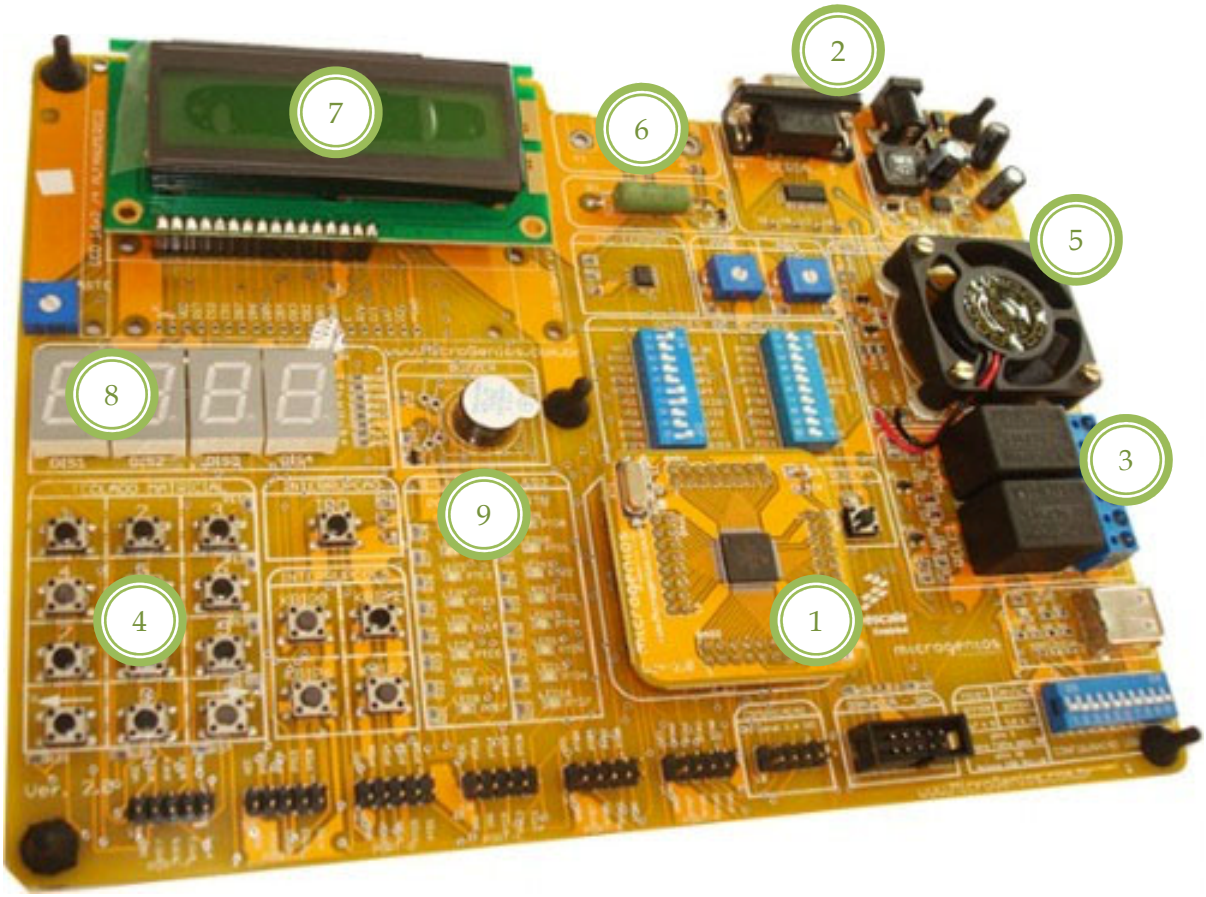

Figura 3.3 – Kit MagicFlexis.

O kit MagicFlexis possui, conforme Figura 3.3:

1. Microcontrolador MC9S08JM60CLH cujas características se encontram na Tabela 3.2;
2. Interface serial RS-232;
3. Relés;
4. Teclado matricial de 12 teclas;
5. Motor DC acionado por Pulse Width Modulation (PWM) + sensor ótico que gera um pulso a cada volta do motor;
6. Resistência acionada por PWM + sensor de temperatura MCP9700;
7. Liquid Crystal Display (LCD) 16x2;
8. Displays de 7 segmentos;
9. Barras de Light Emitting Diode (LED).

4 DESENVOLVIMENTO DO PROJETO

Neste capítulo são mostradas em detalhes todas as etapas do desenvolvimento do projeto.

O capítulo inicia com o estudo do hardware utilizado. Na sequência, todos os aspectos relativos ao firmware do produto são apresentados. Em seguida, a identificação dos sistemas presentes no kit MagicFlexis é feita. Por fim, o projeto dos controladores de temperatura e velocidade é realizado.

4.1 Estudo do Kit MagicFlexis

Após o recebimento do kit de desenvolvimento, deu-se início o seu estudo com o intuito de conhecer melhor seu hardware. A partir dos esquemáticos fornecidos pela Microgenios, mostrados na Figura 4.1, na Figura 4.2, na Figura 4.3, na Figura 4.4, na Figura 4.5 e na Figura 4.6, foi montada a Tabela 4.1, que mostra a alocação dos pinos do microcontrolador.

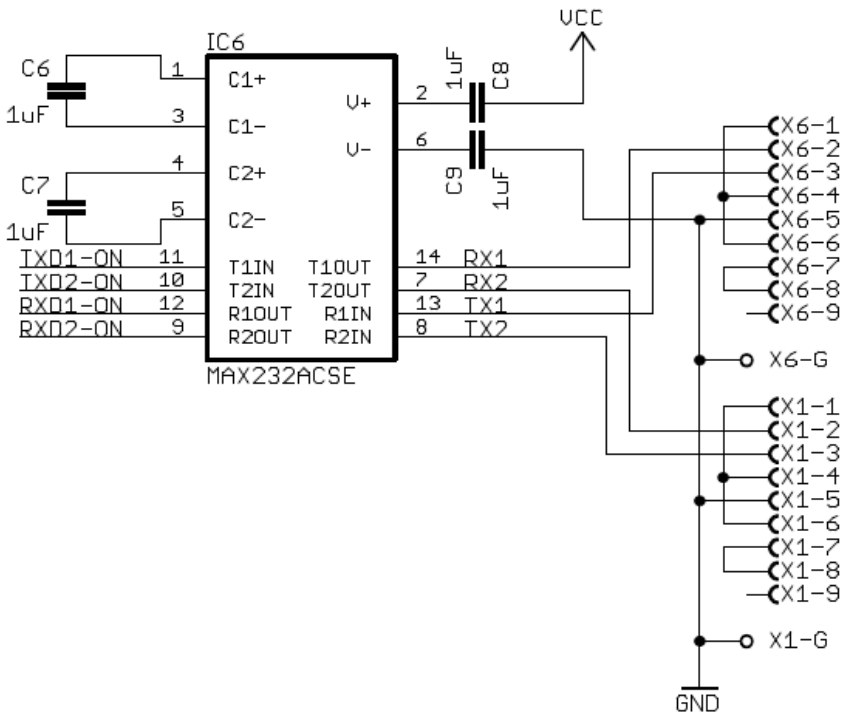

Figura 4.1 – Esquemático interface serial.

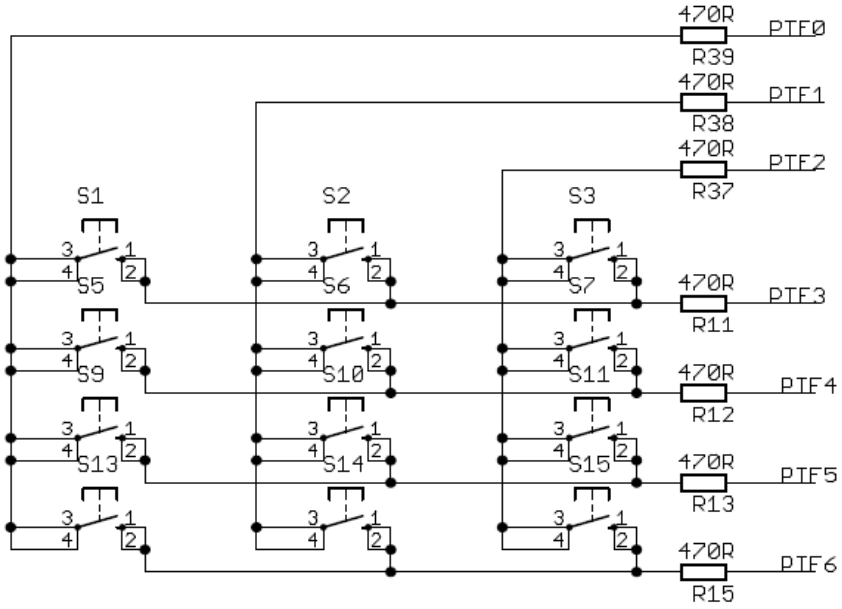

Figura 4.2 – Esquemático teclado matricial.

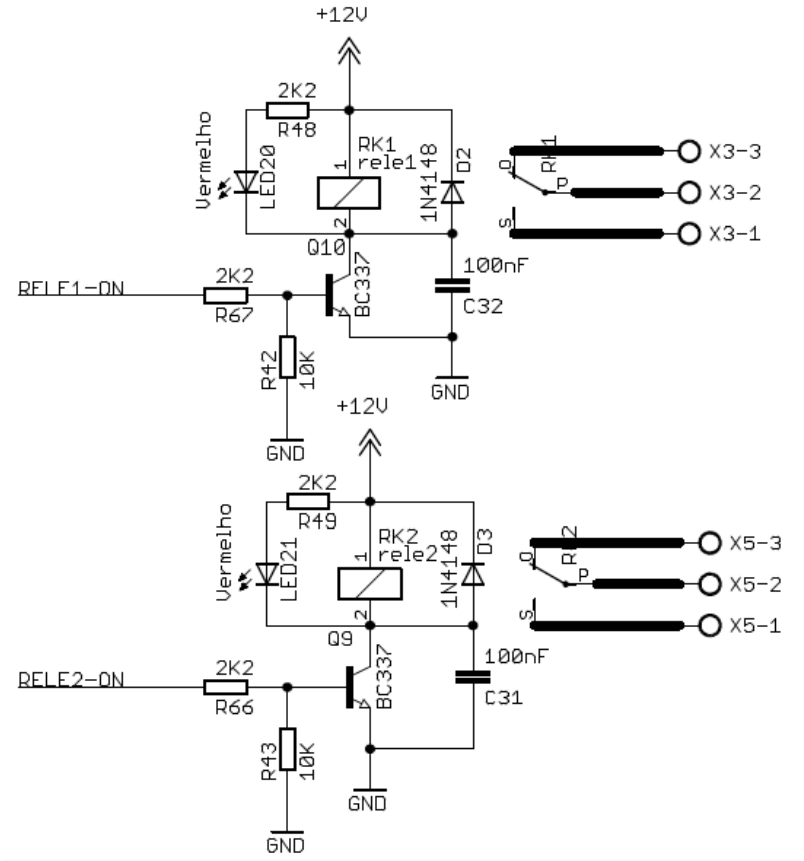

Figura 4.3 – Esquemático relés.

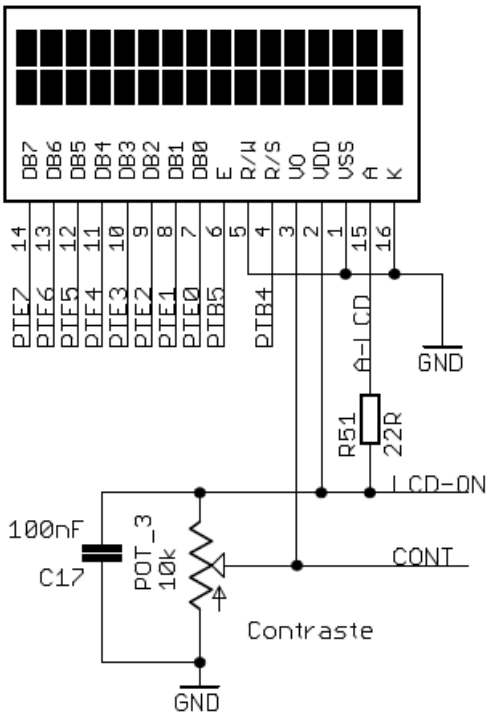

Figura 4.4 – Esquemático LCD.

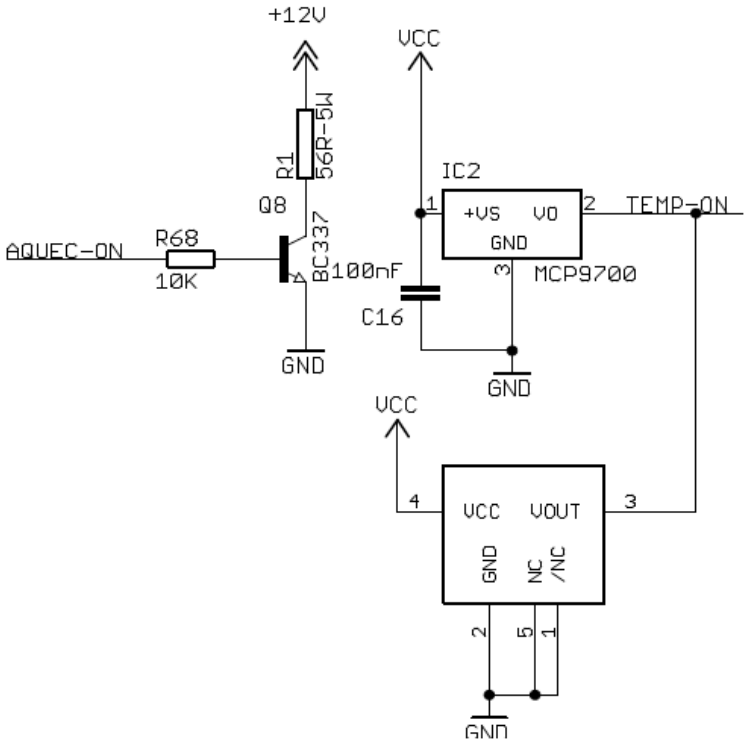

Figura 4.5 – Esquemático sensor de temperatura.

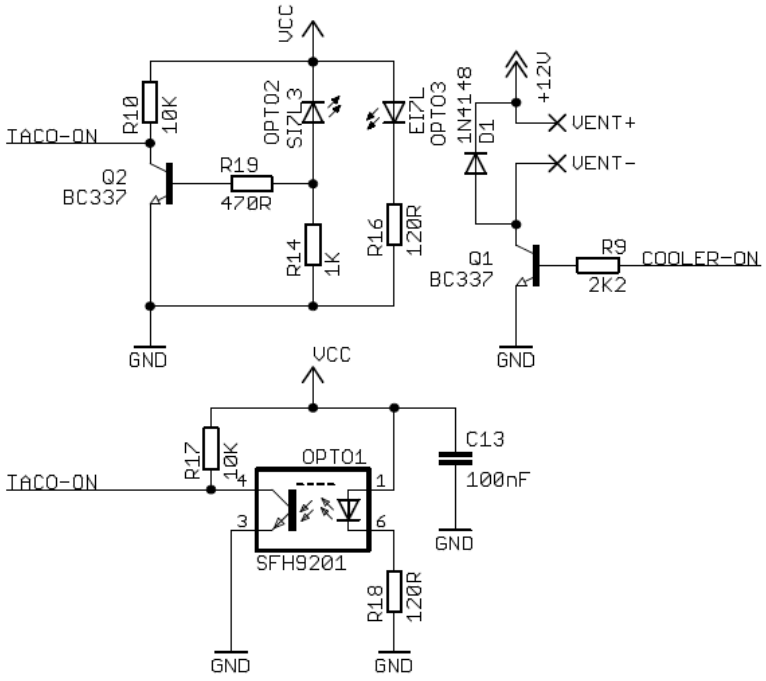

Figura 4.6 – Esquemático sensor de velocidade.

Tabela 4.1 – Alocação dos pinos.

Porta	Pino	Sinal	Direção	Lógica	Descrição / Observação
PTA0 - PTA5	28 - 33				Alguns sinais do USB
PTB0	34	AN0	IN	+	Conectado ao canal 0 do ADC
PTB1	35	AN1	IN	+	Conectado ao canal 1 do ADC
PTB2	36	SENSOR_TEMP	IN	+	Conectado ao canal 2 do ADC
PTB4	38	LCD_CONF	OUT	+	Configuração = 0, caractere = 1
PTB5	39	LCD_DADO_OK	OUT	+	LCD_DADO_OK
PTC0	60	SCL_EEPROM	OUT	+	Clock I²C
PTC1	61	SDA_EEPROM	BI	+	Data I²C
PTC2	62	DISP1_ON	OUT	+	DISP1_ON
PTC3	63	DISP2_ON	OUT	+	DISP2_ON
PTC4	1	DISP3_ON	OUT	+	DISP3_ON
PTC5	64	DISP4_ON	OUT	+	DISP4_ON
PTD0	42	LED9n	OUT	-	LED9
PTD1	43	LED10n	OUT	-	LED10
PTD2	48	LED11n	OUT	-	LED11
PTD3	49	LED12n	OUT	-	LED12
PTD4	50	LED13n	OUT	-	LED13
PTD5	51	LED14n	OUT	-	LED14
PTD6	52	LED15n	OUT	-	LED15
PTD7	53	LED16n	OUT	-	LED16
PTD0	42	RELE1	OUT	+	RELE1

PTD1	43	RELE2	OUT	+	RELE2
PTE0	13	LED1n	OUT	-	LED1
PTE1	14	LED2n	OUT	-	LED2
PTE2	15	LED3n	OUT	-	LED3
PTE3	16	LED4n	OUT	-	LED4
PTE4	17	LED5n	OUT	-	LED5
PTE5	18	LED6n	OUT	-	LED6
PTE6	19	LED7n	OUT	-	LED7
PTE7	20	LED8n	OUT	-	LED8
PTE0	13	DISP_A	OUT	+	DISP_A
PTE1	14	DISP_B	OUT	+	DISP_B
PTE2	15	DISP_C	OUT	+	DISP_C
PTE3	16	DISP_D	OUT	+	DISP_D
PTE4	17	DISP_E	OUT	+	DISP_E
PTE5	18	DISP_F	OUT	+	DISP_F
PTE6	19	DISP_G	OUT	+	DISP_G
PTE7	20	DISP_PT	OUT	+	DISP_PT
PTE0	13	TXD1	OUT	+	TXD1
PTE1	14	RXD1	IN	+	RXD1
PTED	13 - 20	LCD_DATA	OUT	+	D7 - D0 Dado escrito no LCD Ao utilizar LCD no modo 4 bits, PTE0 – PTE3 ficam livres para uso
PTF0	4	AQUEC	OUT	+	Conectado ao canal 2 do TPM1
PTF1	5	MOTOR	OUT	+	Conectado ao canal 3 do TPM1
PTF4	8	SENSOR_MOTOR	IN	-	Conectado ao canal 0 do TPM2
PTF5	11	BUZZER	OUT	+	BUZZER
PTF0	4	TECLADO_COL1	OUT	+	Coluna 1 do teclado
PTF1	5	TECLADO_COL2	OUT	+	Coluna 2 do teclado
PTF2	6	TECLADO_COL3	OUT	+	Coluna 3 do teclado
PTF3	7	TECLADO_LIN1n	IN	-	Linha 1 do teclado (sem R pull-up)
PTF4	8	TECLADO_LIN2n	IN	-	Linha 2 do teclado (sem R pull-up)
PTF5	11	TECLADO_LIN3n	IN	-	Linha 3 do teclado (sem R pull-up)
PTF6	12	TECLADO_LIN4n	IN	-	Linha 4 do teclado (sem R pull-up)
PTG0	26	KBIP0n	IN	-	Conectado KBIP0 (sem R pull-up)
PTG1	27	KBIP1n	IN	-	Conectado KBIP1 (sem R pull-up)
PTG2	54	KBIP2n	IN	-	Conectado KBIP6 (sem R pull-up)
PTG3	55	KBIP3n	IN	-	Conectado KBIP7 (sem R pull-up)
IRQ	2	IRQn	IN	-	IRQ
RESET	3	RESETn	IN	-	RESET
BKGD	56	BKGD	BI	+	Sinal de background para debugging

Da análise da Figura 4.1, da Figura 4.2, da Figura 4.3, da Figura 4.4, da Figura 4.5, da Figura 4.6 e da Tabela 4.1, percebe-se que:

- O pino de transmissão da interface serial está conectado ao pino 2 do conector DB9 fêmea presente no kit, assim como o pino de recepção está conectado ao pino 3 do conector DB9, tornando o módulo de I/O um dispositivo DCE;
- Tanto a resistência quanto o motor DC presentes no kit estão prontos para serem acionados pelos canais de PWM do microcontrolador;
- Existem muitos dispositivos conectados em determinadas portas do microcontrolador, como na porta E, onde estão conectados 8 LED, 8 pinos de dados do LCD e ainda os 2 sinais da interface serial.

Após o conhecimento inicial a respeito do hardware utilizado, alguns testes foram realizados para comprovar se os dispositivos estavam funcionando adequadamente. Não foi constatada nenhuma anomalia e, portanto, pôde-se passar para a próxima etapa do projeto: o desenvolvimento do firmware que controla todo o funcionamento do módulo de I/O remoto MODBUS.

4.2 Desenvolvimento do Firmware

Como já descrito na seção 3.2.2, o ambiente de desenvolvimento utilizado foi o CodeWarrior, oferecido pela fabricante do microcontrolador, a Freescale. A Ediçao Especial do ambiente de desenvolvimento é grátis e permite projetos de até 32 kB de código. Além disso, a ferramenta de design rápido Device Initialization foi utilizada para configurar os periféricos do microcontrolador e para gerar código para a tabela de vetores de interrupção. A linguagem de programação utilizada foi a linguagem C.

Primeiramente, funções básicas de leitura e escrita em alguns dos periféricos no kit foram desenvolvidas. Tais funções são descritas nas 2 seções subsequentes.

4.2.1 Funções de Escrita no LCD

O LCD é largamente utilizado em diversos aparelhos eletrônicos com a finalidade de mostrar resultados preliminares ou informações relevantes ao usuário.

Para que ele funcione corretamente, é necessário primeiro configurá-lo, informando ao display como será a transferência de dados (8 ou 4 bits), quantas linhas serão utilizadas, se a mensagem deverá ficar fixa ou rolar e se a escrita será da esquerda para direita ou da direita para esquerda. Outro detalhe importante ao se trabalhar com este tipo de display é a temporização.

A descrição dos pinos do LCD 16x2 presente no kit e mostrado na Figura 4.4 é listada abaixo:

- Pinos de dados (DB0 – DB7): são usados para enviar as configurações e os caracteres a serem escritos no LCD;
- Pinos de controle (E, R/S e R/W): o pino E informa ao LCD quando os dados presentes nos pinos de dados estão prontos para serem lidos. O pino R/S informa ao LCD se os dados presentes nos pinos de dados são de configuração ou correspondem a um caractere a ser escrito no display. O pino R/W informa operação de escrita ou leitura ao LCD. Nota-se que o pino R/W está conectado ao GND do kit, sendo possível somente escrever caracteres no display;
- Pinos de alimentação (VDD e VSS);
- Pino de controle de contraste (VO): permite alterar o contraste do display;
- Pinos de iluminação do fundo (A e K).

O diagrama de tempo para a escrita de comandos no LCD é mostrado na Figura 4.7.

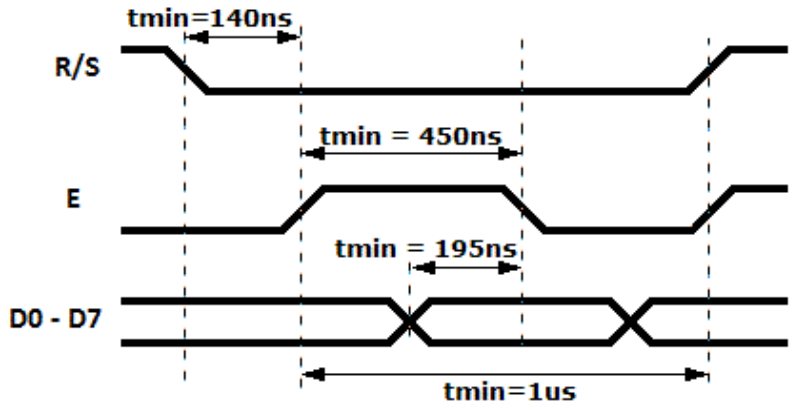

Figura 4.7 – Diagrama de tempo para escrita de comandos.

O diagrama de tempo para escrita de um caractere no display é muito semelhante ao diagrama de escrita de comandos, mas com o sinal R/S em nível lógico 1.

Foram desenvolvidas 3 funções para a escrita no LCD:

```
void EscreveLCDByte(UINT8 Dado, UINT8 Conf, UINT8 Tempo);

      Retorno: vazio
      Parâmetros:
            Dado – 8 bits de dados que correspondem a uma configuração ou a
            um caractere a ser escrito no display
            Conf – 0 indica configuração
                   1 indica caractere
            Tempo – tempo entre 2 bordas de subida do sinal E
      Resumo: escreve uma configuração ou um caractere no display
      utilizando o modo 8 bits e aguarda determinado tempo
```

```
void EscreveLCDNibble(UINT8 Dado, UINT8 Conf, UINT8 Tempo);

      Retorno: vazio
      Parâmetros:
            Dado – 8 bits de dados que correspondem a uma configuração ou a
            um caractere a ser escrito no display
            Conf – 0 indica configuração
                   1 indica caractere
            Tempo – tempo entre 2 bordas de subida do sinal E
      Resumo: escreve uma configuração ou um caractere no display
      utilizando o modo 4 bits e aguarda determinado tempo

void EscreveLCDString(UINT8 *String);

      Retorno: vazio
      Parâmetros:
            *String – endereço inicial da string a ser escrita no display
      Resumo: utiliza a função EscreveLCDNibble() para escrever uma string
      no display
```

No modo 4 bits, o LCD utiliza os pinos DB4 – DB7 para receber dados de 8 bits (4 bits de cada vez) liberando assim os pinos PTE0 – PTE3 do microcontrolador para uso geral. O modo 8 bits foi utilizado somente para realizar a configuração inicial do display, pois, conforme a Tabela 4.1, os pinos PTE0 e PTE1 precisam estar livres para uso da interface serial.

Conforme testes realizados, o tempo necessário para limpar o display é de 1,6 ms e o tempo necessário para os demais comandos (configuração e escrita de caracteres) é de 200 µs.

4.2.2 Função de Leitura do Teclado Matricial

O esquemático do teclado matricial de 12 teclas presente no kit é mostrado na Figura 4.2. Os pinos do microcontrolador conectados às linhas do teclado foram configurados como entradas e com R de pull-up. Já os pinos conectados às colunas foram configurados como saídas.

O processo de varredura do teclado é feito da seguinte maneira:

- Aplicar às colunas 011 e monitorar as linhas. Nível lógico 0 corresponde a botão pressionado;
- Aplicar às colunas 101 e monitorar as linhas. Nível lógico 0 corresponde a botão pressionado;
- Aplicar às colunas 110 e monitorar as linhas. Nível lógico 0 corresponde a botão pressionado;

Foi desenvolvida 1 função para a leitura das teclas do teclado:

```
UINT8 LerTeclado(void);

      Retorno: tecla pressionada no formato ASCII
      Parâmetros: vazio
      Resumo: realiza o processo de varredura citado acima e retorna a
      tecla pressionada no formato ASCII
```

4.2.3 Estrutura do Firmware

A estrutura da função `main()`do firmware do módulo de I/O remoto MODBUS é mostrada abaixo:

```
void main(void)
{
      MCU_init(); /* call Device Initialization */
      /* include your code here */

      InicializacaoModulo();

      for(;;)
      {
            ManipuladorAplicacao();
            ManipuladorTransmissaoRTU();
            ManipuladorEscravo();

            __RESET_WATCHDOG();
      } /* loop forever */
      /* please make sure that you never leave main */
}
```

Portanto, o módulo após ser alimentado ou sofrer um reset:

- Chama a função `MCU_init()`, onde são configurados os periféricos do microcontrolador;
- Chama a função `InicializacaoModulo()`, onde são configurados os periféricos do kit MagicFlexis, como o LCD, e onde o endereço MODBUS escravo é atribuído ao módulo pelo usuário;
- Chama ciclicamente as funções `ManipuladorAplicacao()`, `ManipuladorTransmissaoRTU()` e `ManipuladorEscravo()` e realiza o reset dos contadores do Watchdog.

As funções acima citadas serão mostradas com maiores detalhes nas próximas seções.

4.2.4 Estruturas de Dados

Para a correta comunicação entre as diversas máquinas de estados da aplicação e para o correto funcionamento do produto, foram criadas algumas estruturas de dados:

```
#define UINT8 unsigned char
#define UINT16 unsigned int
#define UINT32 unsigned long int

typedef struct status_transmissao_RTU
{
      UINT32
            /* MSB */
            bFrameNOK               :1,
            bFrameOK                :1,
            bErroParidade           :1,
            u9ContadorEmissao       :9,
            u9NoCaracEmissao        :9,
            u9ContadorRecepcao      :9;
            /* LSB */
} status_trans_RTU;

typedef struct status_escravo
{
      UINT8
            /* MSB */
            bFlagRequisicao         :1,
            bFlagEmissao            :1,
            u3CodigoErro            :3;
            /* LSB */
} status_esc;

typedef struct status_sensor_velocidade
{
      UINT8
            /* MSB */
            bFlagHabilitacao        :1,
            bFlagLeituraOK          :1;
            /* LSB */
} status_vel;

typedef enum estados_transmissao_RTU
{
      InitializationTrans = 0,
      Initial,
      IdleTrans,
      Emission,
      Reception,
      Control_Waiting
} estado_trans_RTU;

typedef enum estados_escravo
{
      InitializationEsc = 0,
      IdleEsc,
      Processing_Request,
      Formating_Normal_Reply,
      Formating_Error_Reply
} estado_esc;

typedef enum estados_inicializacao
{
      InicializacaoInit = 0,
      IdleInit,
      Debouncing,
      Controle,
```

```
      Gera_Endereco
} estado_init;

typedef enum estados_aplicacao
{
      InicializacaoApp = 0,
      OperacionalApp
} estado_app;

typedef enum estados_saidas_analogicas
{
      InicializacaoSaidas = 0,
      Pre_OperacionalSaidas,
      OperacionalSaidas
} estado_saidas;

typedef enum estados_entradas_analogicas
{
      InicializacaoEntradas = 0,
      OperacionalEntradas
} estado_entradas;

typedef struct entradas_digitais
{
      UINT8
            /* MSB */
            bEntrada                :1;
            /* LSB */
} entradas_dig;

typedef struct saidas_digitais
{
      UINT8
            /* MSB */
            bSaida                  :1;
            /* LSB */
} saidas_dig;
```

4.2.5 Variáveis Globais

As variáveis globais presentes no firmware do módulo de I/O remoto MODBUS são listadas abaixo:

```
UINT8 u8EnderecoModulo;
UINT16 u16Contador200us;

UINT8 u8BufferRecepcao[256];
UINT8 u8BufferEmissao[256];
status_trans_RTU stStatusTrans;

UINT8 u8BufferRequisicao[254];
status_esc stStatusEscravo;

entradas_dig edEntradasDigitais[NO_ENTRADAS_DIGITAIS];
saidas_dig sdSaidasDigitais[NO_SAIDAS_DIGITAIS];
UINT16 u16EntradasAnalogicas[NO_ENTRADAS_ANALOGICAS];
UINT16 u16SaidasAnalogicas[NO_SAIDAS_ANALOGICAS];
```

```
UINT16 u16BufferContadorVelocidade[2];
status_vel stStatusSensorVelocidade;
```

4.2.6 Interrupções

Entre as diversas fontes de interrupção do microcontrolador, algumas foram utilizadas pelo módulo de I/O remoto MODBUS:

- Estouro do contador do Timer 1: incrementa a variável global `u16Contador200us` a cada 200 μs. Essa variável é utilizada por diversas outras funções como base temporal;
- Indicação de evento externo do canal 0 do Timer 2: utilizada para obter o valor da velocidade do motor DC;
- Indicação de recepção da interface serial 1: utilizada em conjunto com a função `ManipuladorTransmissaoRTU()` para recebimento de dados;
- Indicação de pronto para transmitir na interface serial 1: utilizada em conjunto com a função `ManipuladorTransmissaoRTU()` para transmissão de dados.

4.2.7 Função `MCU_init()`

A função `MCU_init()` é gerada pela ferramenta Device Initialization do CodeWarrior e configura os diversos periféricos do microcontrolador. Os principais periféricos utilizados no módulo de I/O remoto MODBUS assim como suas respectivas configurações são listadas abaixo:

- Clock do barramento interno: 24 MHz;
- Watchdog: período de 256 ms;
- Interface serial 1: baud rate de 9600 bps, 9 bits de dados, paridade par;
- Timer 1: período de 200 μs, canais 2 e 3 habilitados como PWM;
- Timer 2: período de 349,325 ms, canal 0 habilitado como Input Capture;
- ADC: modo 12 bits, conversão única, habilitação por software, função comparação desabilitada;
- PTB: Pinos 4 e 5 habilitados como saídas, slew rate control e drive strength habilitados;
- PTD: Pinos 0 e 1 habilitados como saídas, slew rate control e drive strength habilitados.

4.2.8 Função `InicializacaoModulo()`

Primeiramente, o LCD recebe as seguintes configurações:

- Modo 4 bits, 2 linhas;
- Escrita da direita para a esquerda, scroll desligado;
- Display ON, cursor desligado.

Após, a mensagem "Módulo de I/O Remoto MODBUS" é mostrada no LCD durante 4 s. O LCD é limpo e a mensagem "End. MODBUS escravo:" é escrita, agora com o cursor ligado e piscando, indicando que o usuário deve digitar o endereço MODBUS escravo para o módulo. Então a função `ManipuladorInicalizacao()` é chamada.

4.2.8.1 Função `ManipuladorInicalizacao()`

O diagrama de estados mostrado na Figura 4.8 foi implementado.

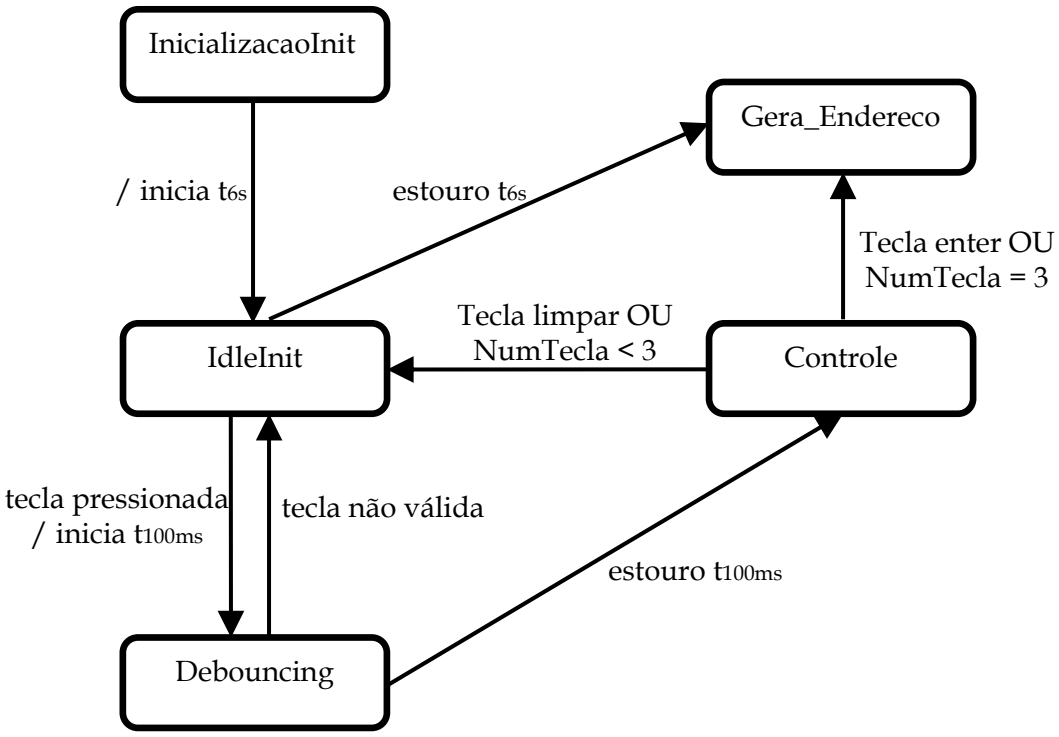

Figura 4.8 – Diagrama de estados da inicialização.

A finalidade do diagrama de estados da Figura 4.8 é realizar a leitura das teclas do teclado e gerar o endereço MODBUS escravo para o módulo após a tecla enter ser pressionado ou após se passarem 6 s. Endereços válidos se encontram na faixa de 1 a 247 decimal. Caso um endereço inválido seja pressionado ou 6 s se passarem sem nenhuma ação do usuário, a máquina de estados da inicialização atribui ao módulo o endereço escravo padrão definido como sendo o endereço 1.

A estrutura da função `ManipuladorInicalizacao()` é mostrada abaixo:

```
void ManipuladorInicalizacao(void)
{
	static estado_init eiEstadoInit = InicializacaoInit;
	static UINT16 u16TempoGeralUltimaLeitura;
	static UINT16 u16TempoCaracUltimaLeitura;
	static UINT8 u8CaracUltimaLeitura;
	static UINT8 u8ContCarac;
	static UINT8 u8Endereco[4] = "000";
	UINT16 u16Endereco;

	switch (eiEstadoInit)
	{

	case InicializacaoInit:

		u16TempoGeralUltimaLeitura = u16Contador200us;
		u16TempoCaracUltimaLeitura = 0;
		u8CaracUltimaLeitura = 0xFF;
		u8ContCarac = 0;

		eiEstadoInit = IdleInit;

	break;

	case IdleInit:

		if(/* condições */)
		{
			/* operações e atribuições */

			eiEstadoInit = Estado_X;
		}

	break;

	case Estado_X:

		/* continua */

	break;

	}
}
```

O modificador de acesso `static` é utilizado nas variáveis cujo valor deve ser mantido entre uma chamada e outra da função.

Todas as demais funções descritas neste trabalho que contenham `Manipulador` em seu nome utilizam a estrutura acima mostrada. A palavra `Manipulador` ainda indica que a função implementa um diagrama de estados.

Após o módulo ter seu endereço atribuído, ele o mostra no LCD durante 4 s. Então, o dispositivo entra em modo normal de operação, chamando ciclicamente as 3 funções descritas nas 3 seções subsequentes.

4.2.9 Função `ManipuladorTransmissaoRTU()`

O diagrama de estados mostrado na Figura 2.27 foi implementado. A variável global `u16Contador200us` é usada como base temporal para a máquina de estados. A variável global `stStatusTrans` tem seu conteúdo alterado para informar para outras funções do módulo os status da transmissão.

4.2.10 Função `ManipuladorEscravo()`

O diagrama de estados mostrado na Figura 2.21 foi implementado. A variável global `stStatusEscravo` tem seu conteúdo alterado para informar para outras funções do módulo os status do escravo.

As funções MODBUS 01 (Leitura de Coils), 02 (Leitura de Discrete Inputs), 03 (Leitura de Holding Registers), 04 (Leitura de Input Registers), 05 (Escrita em uma única Coil) e 06 (Escrita em um único Holding Register) são chamadas nesse manipulador e foram implementadas de acordo com os fluxogramas mostrados na Figura 4.9, na Figura 4.10, na Figura 4.11, na Figura 4.12, na Figura 4.13 e na Figura 4.14.

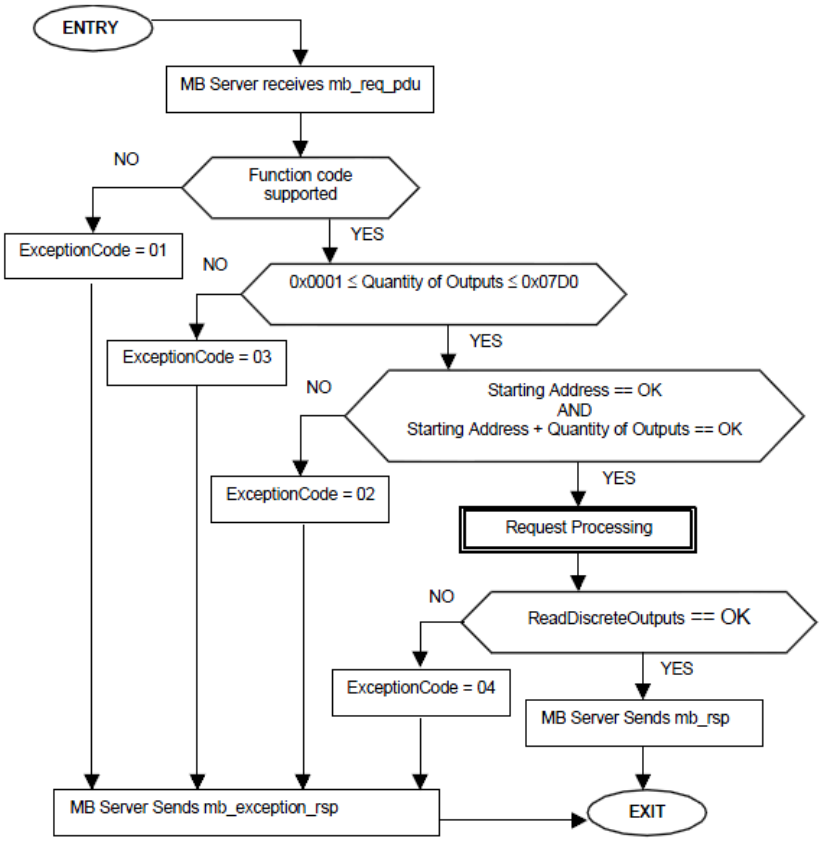

Figura 4.9 – Fluxograma função 01 (Leitura de Coils).

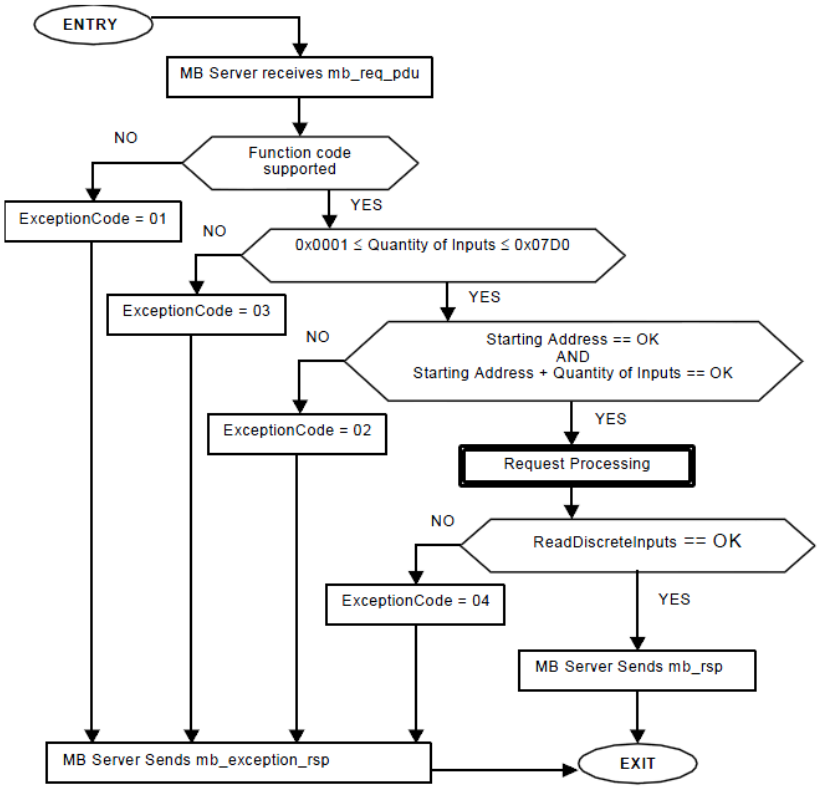

Figura 4.10 – Fluxograma função 02 (Leitura de Discrete Inputs).

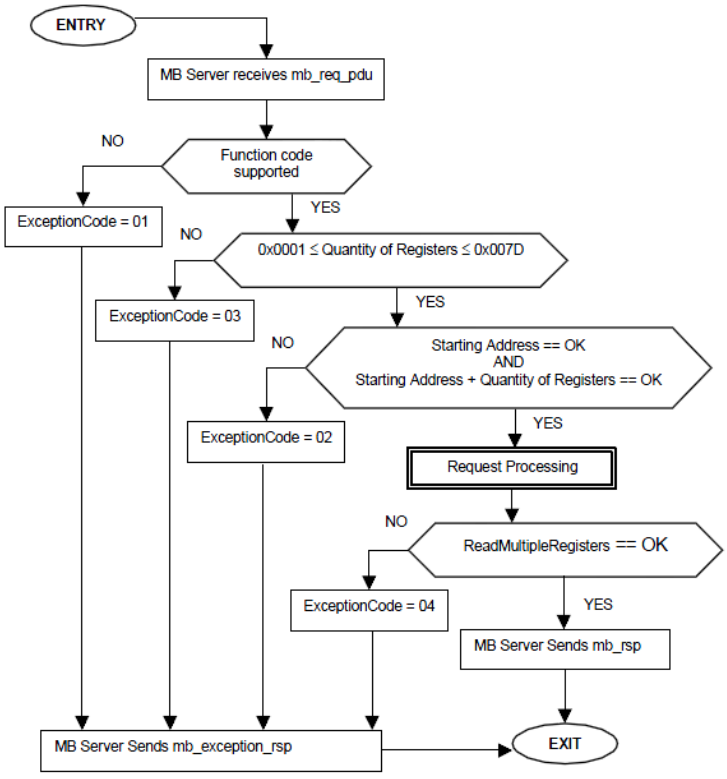

Figura 4.11 – Fluxograma função 03 (Leitura de Holding Registers).

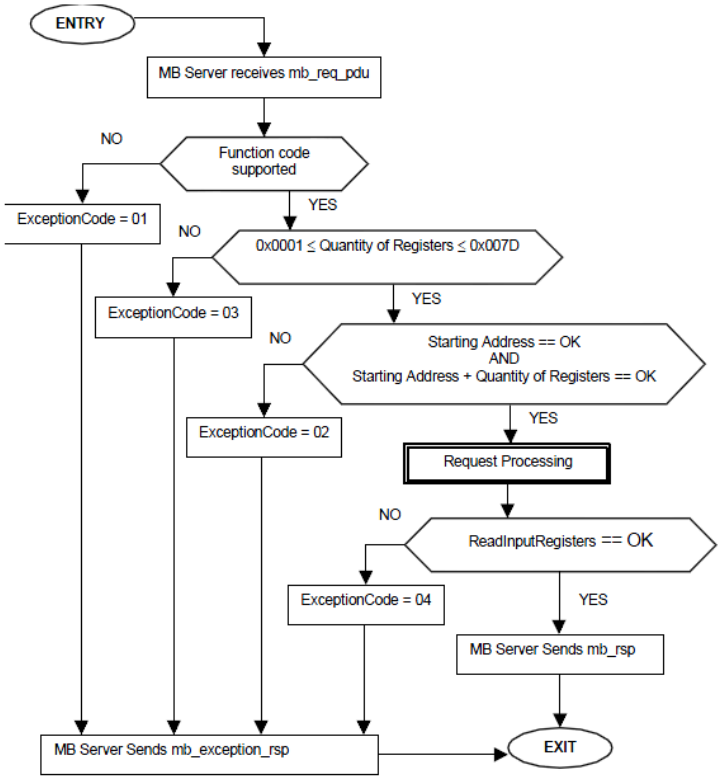

Figura 4.12 – Fluxograma função 04 (Leitura de Input Registers).

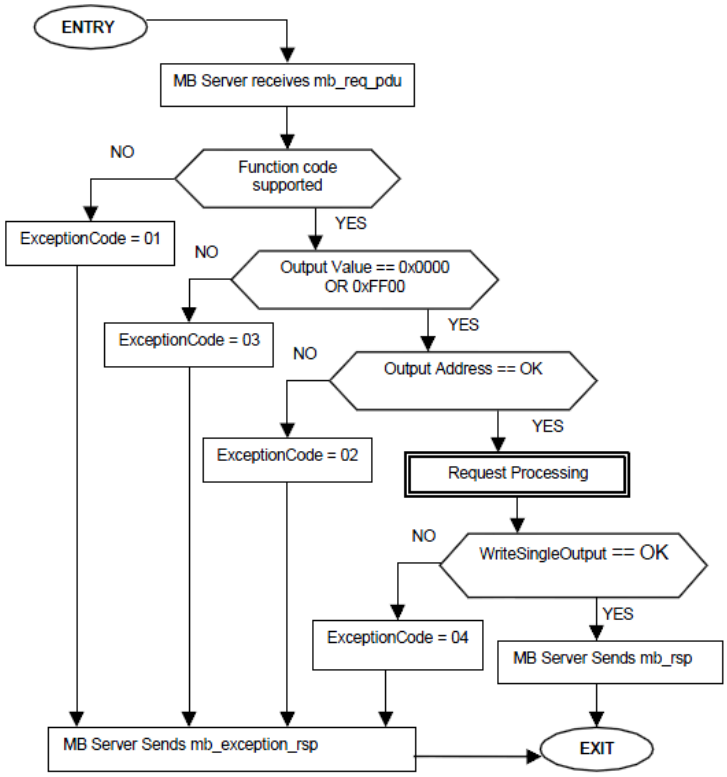

Figura 4.13 – Fluxograma função 05 (Escrita em uma única Coil).

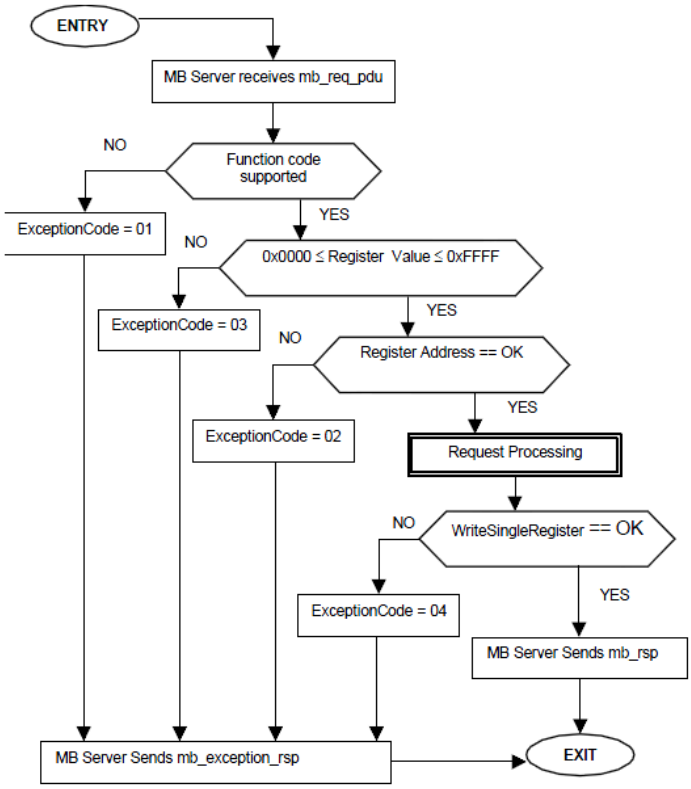

Figura 4.14 – Fluxograma função 06 (Escrita em um único Holding Register).

É através das funções MODBUS que o protocolo de comunicação tem acesso aos dados da aplicação, lendo ou escrevendo dados nas tabelas MODBUS, que são descritas na Tabela 2.4.

4.2.11 Função `ManipuladorAplicacao()`

Tem-se para o módulo de I/O remoto MODBUS:

- 2 entradas digitais que correspondem a 2 botões do teclado matricial e que equivalem a 2 Discrete Inputs do modelo de dados MODBUS. Discrete Inputs são do tipo somente leitura e são lidas pelo protocolo MODBUS através da função 02 (Leitura de Discrete Inputs);

- 2 saídas digitais que correspondem a 2 relés e que equivalem a 2 Coils do modelo de dados MODBUS. Coils são do tipo leitura e escrita; são lidas pelo protocolo MODBUS através da função 01 (Leitura de Coils) e são escritas através da função 05 (Escrita em uma única Coil);

- 2 entradas analógicas que correspondem a temperatura da resistência em ºC e a velocidade do motor em rpm e que equivalem a 2 Input Registers do modelo de dados MODBUS. Input Registers são do tipo somente leitura e são lidos pelo protocolo MODBUS através da função 04 (Leitura de Input Registers);

- 2 saídas analógicas que correspondem ao ciclo de trabalho para os canais PWM que acionam a resistência e o motor e que equivalem a 2 Holding

Registers do modelo de dados MODBUS. Holding Registers são do tipo leitura e escrita; são lidos pelo protocolo MODBUS através da função 03 (Leitura de Holding Registers) e são escritas através da função 06 (Escrita em um único Holding Register).

A função `ManipuladorAplicacao()` trata do I/O do módulo. É ela quem lê ou escreve na mesma área de memória utilizada pelos dados do protocolo MODBUS.

4.2.11.1 Entradas Digitais

A função simplesmente realiza a leitura de 2 botões do teclado matricial e copia os valores da leitura (botão pressionado = 1) para a variável global `edEntradasDigitais[NO_ENTRADAS_DIGITAIS]`. É dessa variável que a função 02 (Leitura de Discrete Inputs) buscará seus dados quando chamada.

4.2.11.2 Saídas Digitais

A função simplesmente aciona os relés em função da variável global `sdSaidasDigitais[NO_SAIDAS_DIGITAIS]`. É nessa variável que a função 05 (Escrita em uma única Coil) escreve seus dados.

4.2.11.3 Saídas Analógicas

A função copia os valores contidos na variável global `u16SaidasAnalogicas[NO_SAIDAS_ANALOGICAS]` para os registradores do Timer 1, que possui os canais 2 e 3 configurados como PWM. Essa ação modifica os ciclos de trabalhos dos PWM usados para acionar a resistência e o motor. A Figura 4.15 mostra o sinal PWM gerado pelos Timers do microcontrolador.

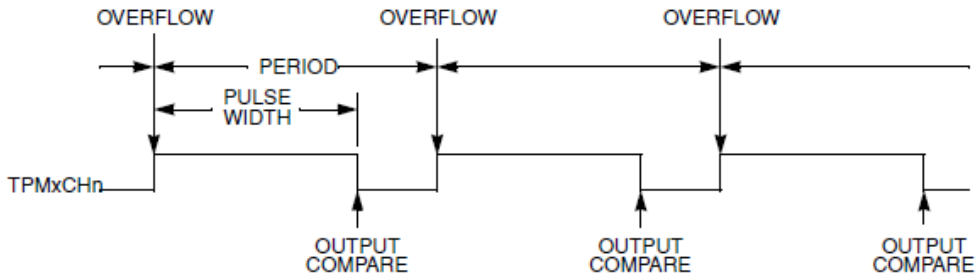

Figura 4.15 – Sinal PWM gerado pelo microcontrolador.

Um período de 200 µs foi configurado, o que equivale a uma frequência de 5 kHz. Valores válidos para os registradores se encontram na faixa de 0 a 4800 decimal. Escrever no registrador o valor 0 equivale a 0% do ciclo de trabalho e consequentemente a 0 V aplicado à

resistência e ao motor. Escrever no registrador 4800 equivale a 100% do ciclo de trabalho e consequentemente a 12 V aplicados à resistência e ao motor.

É na variável `u16SaidasAnalogicas[NO_SAIDAS_ANALOGICAS]` que a função 06 (Escrita em um único Holding Register) escreve seus dados.

4.2.11.4 Entradas Analógicas

Foi realizada a interface com os 2 sensores presentes no kit, obtendo assim a temperatura da resistência e a velocidade do motor.

Temperatura

O sensor de temperatura utilizado é o MCP9700, cujas características se encontram na Tabela 4.2.

Tabela 4.2 – Características do sensor de temperatura MCP9700.

Faixa de medida	-40 ºC a +125 ºC
Exatidão	± 4 ºC
Sensibilidade	10 mV/ºC
Faixa de alimentação	3,1 V a 5,5 V
Corrente	6 µA (típica)

A função de transferência do sensor de temperatura é mostrada na Equação 1.

$$V_{out} = 0{,}01T + 0{,}5 \quad (1)$$

Onde V_{out} é a tensão de saída do sensor em V e T é a temperatura ambiente em ºC.

A tensão V_{out} é convertida em um número de 12 bits pelo ADC do microcontrolador. Como a tensão de alimentação do ADC é 5 V, pode-se escrever:

$$T = 100(V_{out} - 0{,}5) = 100\left(\frac{5}{4095}Num - 0{,}5\right) \quad (2)$$

Onde Num é o número de 12 bits correspondente a tensão de saída do sensor de temperatura.

A função que realiza a leitura da temperatura faz a média dos últimos 8 valores obtidos do ADC e aplica a Equação 2. A média dos últimos 8 valores do ADC é feita para reduzir o erro introduzido pelo ruído. Além disso, a função copia o valor da temperatura para a variável global `u16EntradasAnalogicas[NO_ENTRADAS_ANALOGICAS]` e atualiza o display do módulo a cada 1 s com o valor atual da temperatura da resistência.

É na variável `u16EntradasAnalogicas[NO_ENTRADAS_ANALOGICAS]` que a função 04 (Leitura de Input Registers) buscará seus dados quando chamada.

Velocidade

O sensor de velocidade é composto por um LED emissor de infravermelho e por um fototransistor. Esse conjunto é posicionado de forma a gerar um pulso de 5 V a cada volta do motor.

A estratégia adotada foi a seguinte: obter o tempo para cada volta do motor e a partir dele gerar a velocidade em rpm. Para isso, o Timer 2 teve seu canal 0 habilitado como Input Capture. Nesse modo, o valor dos contadores é capturado a cada evento externo, no caso, o pulso gerado pelo sensor de velocidade.

A velocidade do motor em rpm é dada por:

$$v_{rpm} = \frac{60}{t_{volta}} = \frac{60}{nT_{timer2}} = \frac{60}{n} f_{timer2} \quad (3)$$

Onde t_{volta} é o tempo para 1 volta do motor em s, n é o valor capturado pelo Timer 2, T_{timer2} é o período do Timer 2 em s e f_{timer2} é a frequência do Timer 2 em Hz.

A frequência do Timer 2 foi configurada para 187,5 kHz possibilitando a seguinte faixa de medição de velocidade:

$$172\ rpm < v_{rpm} < 11 \times 10^6\ rpm \quad (4)$$

A função que realiza a leitura da velocidade obtêm 2 valores capturados no Timer 2, realiza a diferença entre eles e aplica a Equação 3. A cada volta do motor uma interrupção é gerada. Além disso, a função copia o valor da velocidade para a variável global `u16EntradasAnalogicas[NO_ENTRADAS_ANALOGICAS]` e atualiza o display do módulo a cada 1 s com o valor atual da velocidade do motor.

É na variável `u16EntradasAnalogicas[NO_ENTRADAS_ANALOGICAS]` que a função 04 (Leitura de Input Registers) buscará seus dados quando chamada.

4.3 Consistência de Dados

Em muitas aplicações onde um protocolo de comunicação é implementado, tem-se o seguinte cenário: uma mesma área de memória é utilizada pelo protocolo de comunicação e pela aplicação. A Figura 4.16 ilustra o fato acima citado para o caso do módulo de I/O remoto MODBUS.

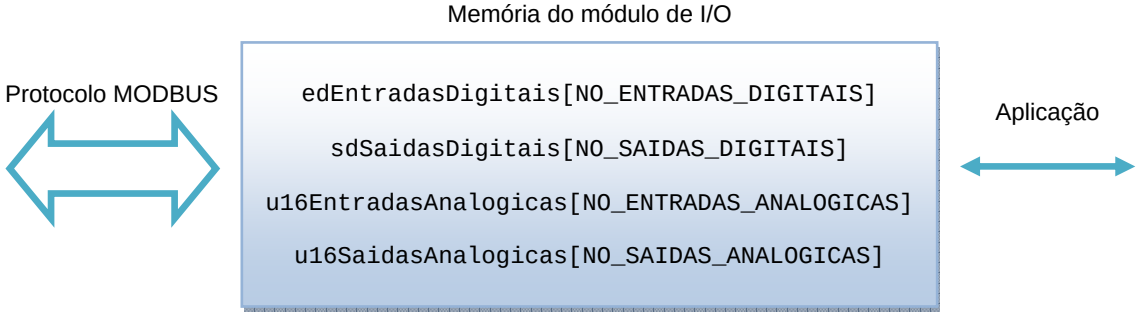

Figura 4.16 – Área de memória compartilhada.

Deve-se evitar que o protocolo MODBUS, através de suas funções, e a aplicação acessem a área de memória compartilhada simultaneamente. Como o acesso do protocolo MODBUS é realizado pela função `ManipuladorEscravo()` e o acesso da aplicação é realizado pela função `ManipuladorAplicacao()` em pontos distintos de código, evita-se acessos simultâneos.

4.4 Portabilidade de Código

Com o intuito de tornar o código adaptável para outros sistemas, onde a necessidade do número de pontos I/O muda, o número de entradas digitais, de saídas digitais, de entradas analógicas e de saídas analógicas foi definido como:

```
#define NO_ENTRADAS_DIGITAIS 2
#define NO_SAIDAS_DIGITAIS 2
#define NO_ENTRADAS_ANALOGICAS 2
#define NO_SAIDAS_ANALOGICAS 2
```

Para alterar o número de I/O do módulo, basta modificar um ou alguns dos defines acima conforme necessidade. Automaticamente a área de memória alocada para o I/O é recalculada e todas as funções MODBUS implementadas passam a aceitar o novo número de I/O para suas funções de leitura e escrita.

4.5 Camada de Enlace

Percebe-se claramente que as funções `ManipuladorTransmissaoRTU()` e `ManipuladorEscravo()` juntamente com códigos presentes nos vetores de interrupção da interface serial tratam do protocolo MODBUS enquanto a função `ManipuladorAplicacao()` trata do I/O do módulo. Portanto, pode-se dividir a camada de enlace de dados em 3 subcamadas que são mostradas na Figura 4.17 (em azul).

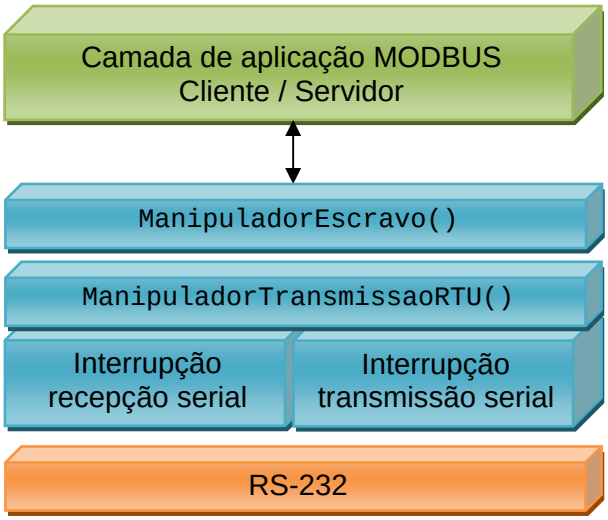

Figura 4.17 – As 3 subcamadas de enlace (em azul).

A subcamada mais baixa está diretamente em contato com o meio físico ao passo que a subcamada mais alta realiza a interface com a aplicação, chamando as funções MODBUS e lendo e/ou escrevendo na área de memória compartilhada do módulo.

4.6 Identificação dos Sistemas

Para que se possa realizar o controle em malha fechada da temperatura da resistência e da velocidade do motor, é necessário ter conhecimento a respeito da dinâmica dos sistemas.

O software Modbus Poll, um simulador de mestre MODBUS, oferece a funcionalidade de fazer requisições utilizando a função 04 (Leitura de Input Registers) e gerar um arquivo .xls com os dados obtidos.

A Figura 4.18 mostra uma tela do Modbus Poll. Em destaque pode-se perceber: Tx, número de requisições feitas pelo mestre; Err, número de erros da comunicação MODBUS; ID, endereço do dispositivo escravo; F, código da função MODBUS e SR, taxa com que o mestre realiza as requisições (no caso, a cada 30 ms).

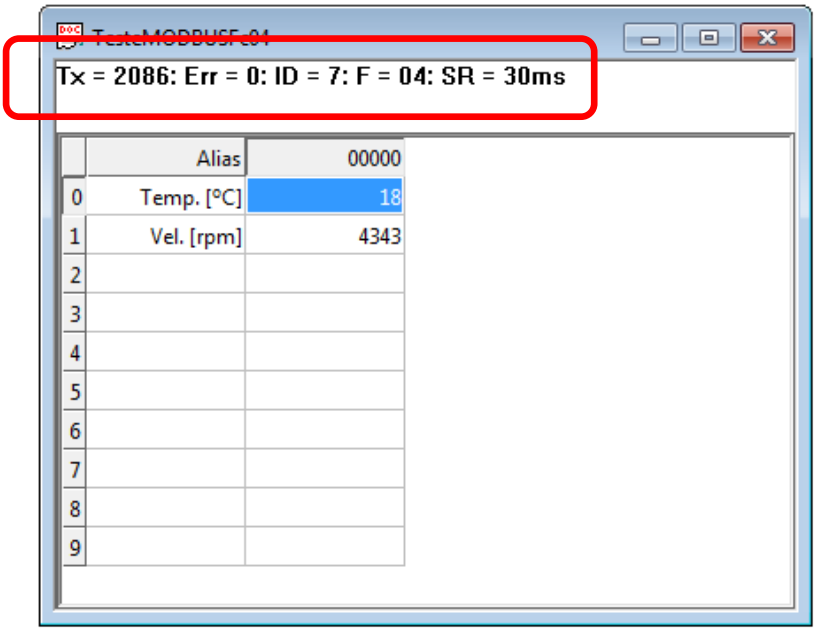

Figura 4.18 – Tela do Modbus Poll.

Foram realizados ensaios ao salto nos 2 sistemas presentes no kit. Os primeiros 5 valores da velocidade obtidos para o ensaio de 80% do ciclo de trabalho (registrador PWM = 3840) são mostrados na Tabela 4.3.

Tabela 4.3 – Ensaio ao salto, 5 primeiros valores.

Leitura de Entradas Analógicas (Fc04) Poll definition: ID = 7, Function = 04, Address = 1, ScanRate = 30		
	Tempo [ms]	Velocidade [rpm]
06/26/11 22:52:15:636	0	0
06/26/11 22:52:15:667	30	623
06/26/11 22:52:15:698	60	800
06/26/11 22:52:15:729	90	1043
06/26/11 22:52:15:760	120	1226

O seguinte script do MATLAB foi desenvolvido para gerar os gráficos para os ensaios ao salto.

```
clear all
clc

%%%%%%%%%%%%%%%%%%%%%% C O N T R O L E   T E M P E R A T U R A

[A,B]=xlsread('D:\UFRGS e
Profissional\Dropbox\TCC\Identificação\Identificacao_Temp01.xls','Plan1','B
3:C1004');
N=length(A);
t_temp=A(1:N,1);
Temp=A(1:N,2)-20;

figure(1);
plot(t_temp,Temp,'r-','LineWidth',1,'DisplayName','Dados ensaio');
hold on;

GTemp=tf([0.00677],[150 1])
numGTemp=[0.00677];
denGTemp=[150 1];
[Temp_sim,t_temp_sim]=step(3840*GTemp,0:1:1000);

plot(t_temp_sim,Temp_sim,'k-','LineWidth',2,'DisplayName','Dados simulados
GTemp');

title('Resposta ao salto de 80% do ciclo de trabalho (registrador PWM =
3840)');
xlabel(B(1));
ylabel(B(2));
legend('show');
grid on;

%%%%%%%%%%%%%%%%%%%%%% C O N T R O L E   V E L O C I D A D E

[A,B]=xlsread('D:\UFRGS e
Profissional\Dropbox\TCC\Identificação\Identificacao_Vel01.xls','Plan1','B3
:C204');
N=length(A);
t_vel1=A(1:N,1);
Vel1=A(1:N,2);
```

```
figure(2);
plot(t_vel1,Vel1,'r-','LineWidth',1,'DisplayName','Dados ensaio');
hold on;

GVel=tf([1.21],[0.730 1])
numGVel=[1.21];
denGVel=[0.730 1];
[Vel1_sim,t_vel1_sim]=step(3840*GVel,0:0.03:6);

plot(t_vel1_sim.*1000,Vel1_sim,'k-','LineWidth',2,'DisplayName','Dados
simulados GVel');

title('Resposta ao salto de 80% do ciclo de trabalho (registrador PWM =
3840)');
xlabel(B(1));
ylabel(B(2));
legend('show');
grid on;
```

A partir da análise gráfica dos resultados, aproximou-se o comportamento dos sistemas como sendo de uma função de transferência de primeira ordem sem atraso de transporte, conforme Equação 5.

$$G(s) = \frac{K}{\tau s+1} \tag{5}$$

Onde: $K = \frac{saída\ do\ sistema\ em\ regime\ permanente}{amplitude\ do\ degrau\ de\ entrada}$ e τ é a constante de tempo do sistema em s.

As funções de transferência identificadas para o sistema de temperatura e para o sistema de velocidade são mostradas na Equação 6 e na Equação 7 respectivamente.

$$G_{Temp}(s) = \frac{0{,}00677}{150s+1} \tag{6}$$

$$G_{Vel}(s) = \frac{1{,}21}{0{,}73s+1} \tag{7}$$

A Figura 4.19 e a Figura 4.20 mostram os gráficos obtidos para os ensaios de temperatura e de velocidade e também trazem a simulação dos sistemas descritos na Equação 6 e na Equação 7.

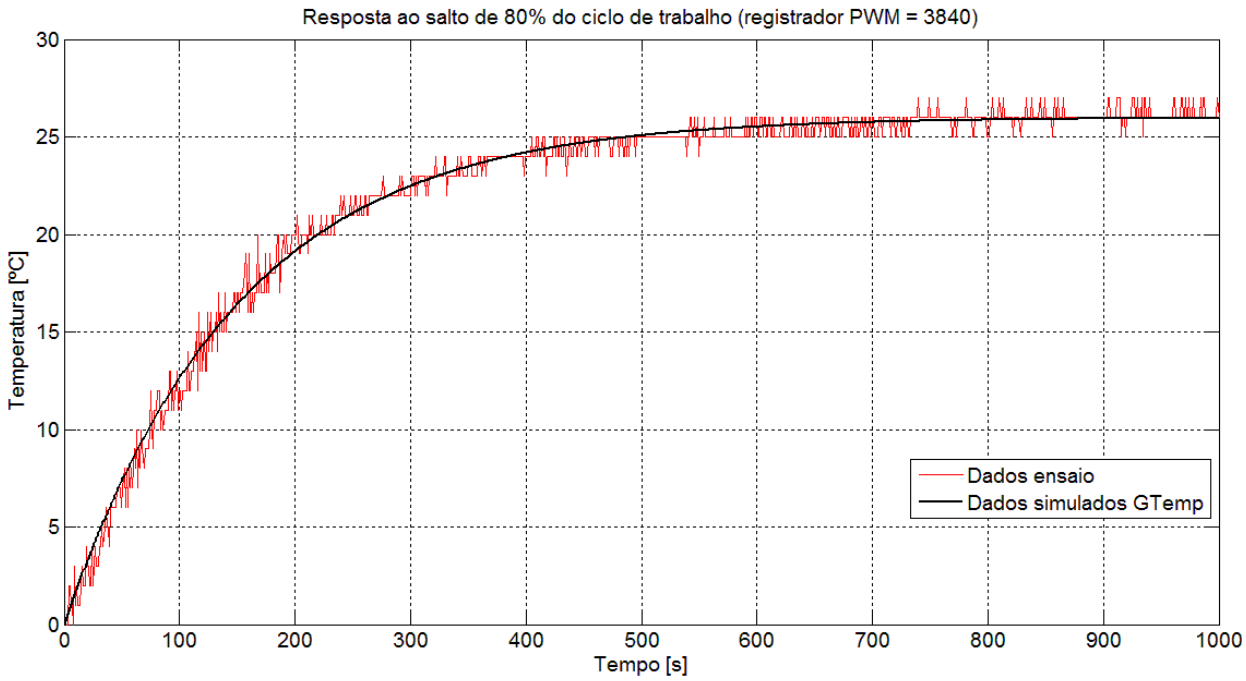

Figura 4.19 – Ensaio ao salto, temperatura.

Em função da sensibilidade do sensor de temperatura, um ruído de amplitude de 10 mV na entrada do ADC causa uma variação de 1 ºC na medida da temperatura. Tal fato pode explicar as variações nos dados obtidos na Figura 4.19. Percebe-se também que nenhum tipo de filtro é utilizado entre o pino de saída do sensor de temperatura e o pino do ADC do microcontrolador para eliminar possível ruído, conforme Figura 4.5.

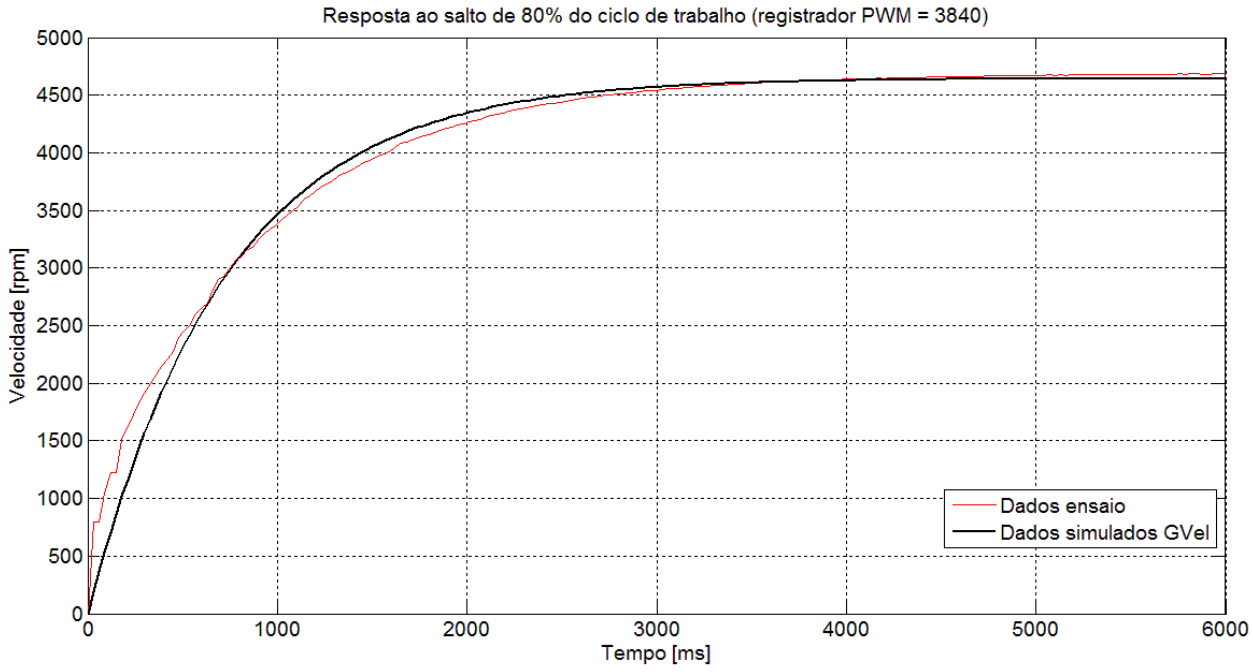

Figura 4.20 – Ensaio ao salto, velocidade.

4.7 Projeto dos Controladores

De posse da Equação 6 e da Equação 7 para os sistemas, foi utilizado o MATLAB/Simulink para o projeto dos controladores de temperatura e de velocidade. A estrutura escolhida dos controladores é proporcional-integral-derivativo (PID).

O modelo Simulink mostrado na Figura 4.21 foi utilizado tanto para o projeto do controlador de temperatura, quanto para o projeto do controlador de velocidade.

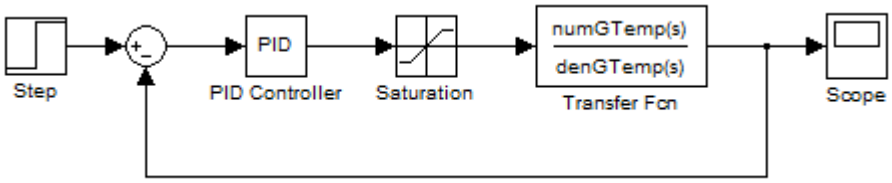

Figura 4.21 – Modelo Simulink para projetos dos controladores.

O bloco de saturação foi introduzido devido à saturação nos atuadores dos sistemas. O valor máximo permitido para os registradores dos PWM é 4800, o que corresponde a 100% do ciclo de trabalho.

4.7.1 Controlador de Temperatura

Diversos valores para os parâmetros do controlador PID foram simulados. Os parâmetros que apresentaram uma resposta com menor tempo de acomodação e menor sobrepasso foram:

$$K_p = 1000\ ;\ T_i = 2{,}5\ ;\ T_d = 0$$

Portanto, um controlador PI.

A Figura 4.22 mostra a simulação do sistema em malha fechada a um salto de 20 ºC.

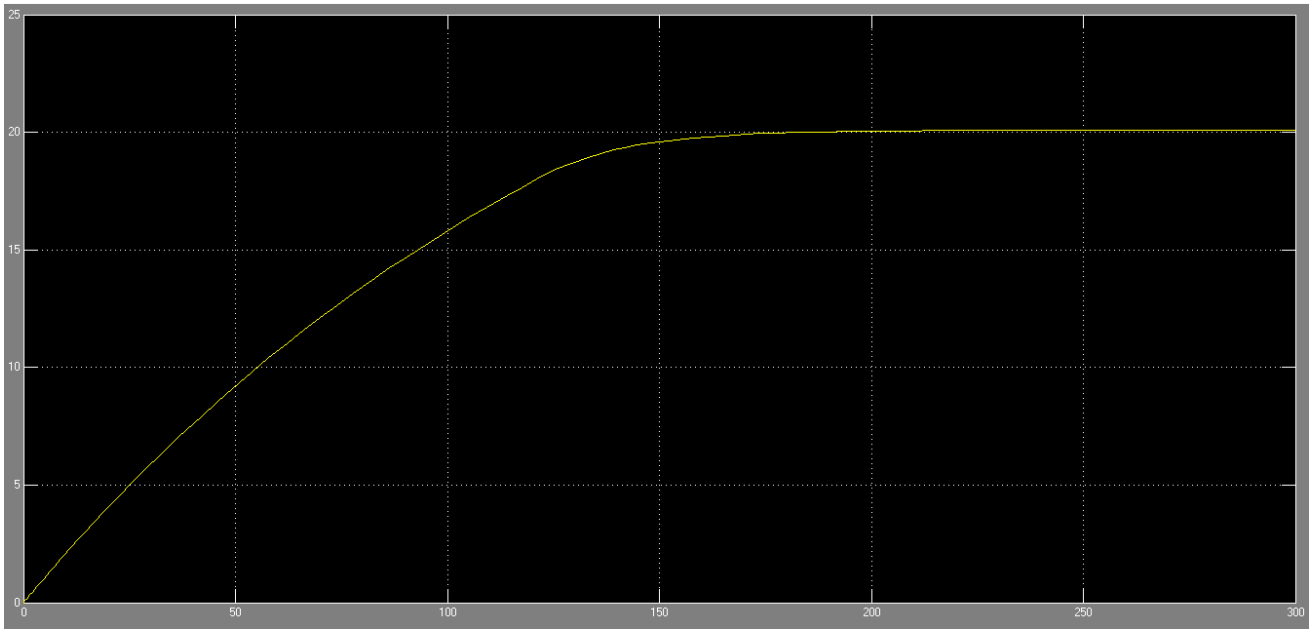

Figura 4.22 – Simulação em malha fechada, temperatura.

4.7.2 Controlador de Velocidade

Diversos valores para os parâmetros do controlador PID foram simulados. Os parâmetros que apresentaram uma resposta com menor tempo de acomodação e menor sobrepasso foram:

$$K_p = 6\ ;\ T_i = 2{,}5\ ;\ T_d = 0$$

Portanto, um controlador PI.

A Figura 4.23 mostra a simulação do sistema em malha fechada a um salto de 4000 rpm.

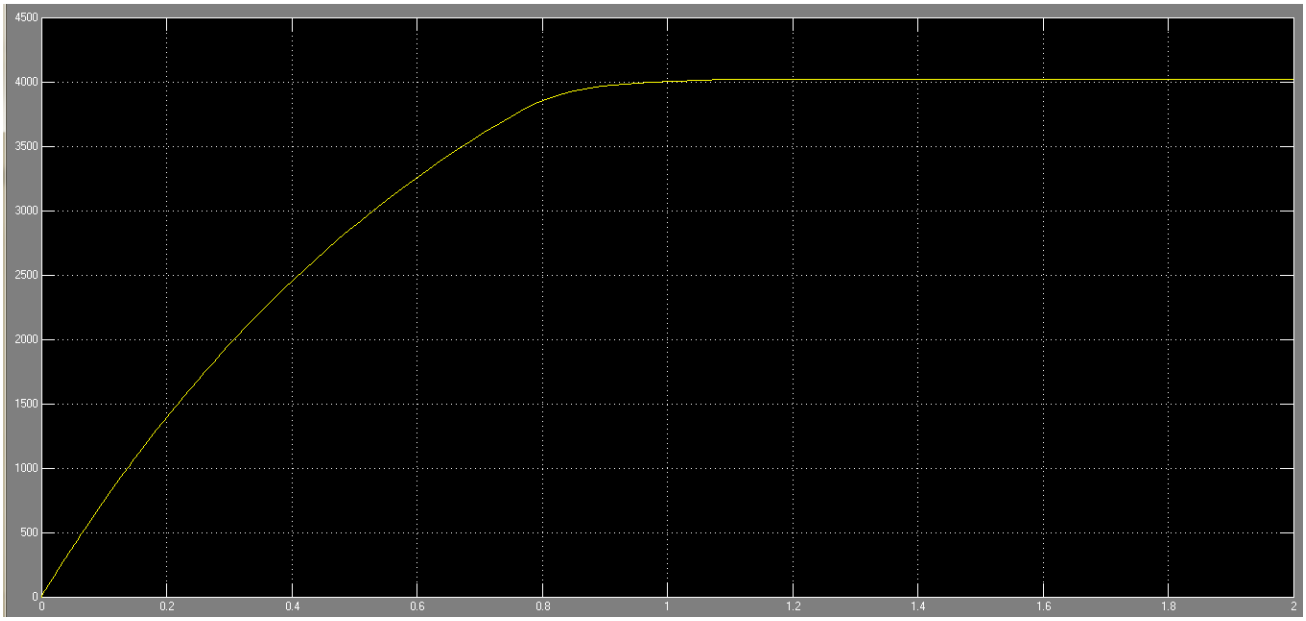

Figura 4.23 – Simulação em malha fechada, velocidade.

5 RESULTADOS E CONCLUSÕES

Neste capítulo são mostrados os resultados obtidos e suas respectivas conclusões. Propostas de melhoria para o produto são igualmente exibidas.

O capítulo inicia com testes realizados para comprovar o funcionamento do protocolo MODBUS. Em seguida, resultados práticos e uma análise teórica do desempenho da comunicação são expostos. Na sequência, resultados práticos do controle de temperatura e de velocidade em malha fechada são apresentados. Por fim, as conclusões e as propostas de melhoria para o produto são exibidas.

5.1 Testes MODBUS

Com o intuito de testar o funcionamento do protocolo MODBUS, foram realizados alguns testes utilizando o software Modbus Poll. Uma tela do programa é mostrada na Figura 4.18. O meio físico utilizado é o RS-232, realizando conexão ponto a ponto. Além disso, uma conexão de 9600 bps com paridade par foi configurada.

A Figura 5.1 mostra as respostas das requisições feitas pelo mestre (Modbus Poll) para o dispositivo escravo (módulo de I/O remoto MODBUS) para as funções 01, 02, 03 e 04.

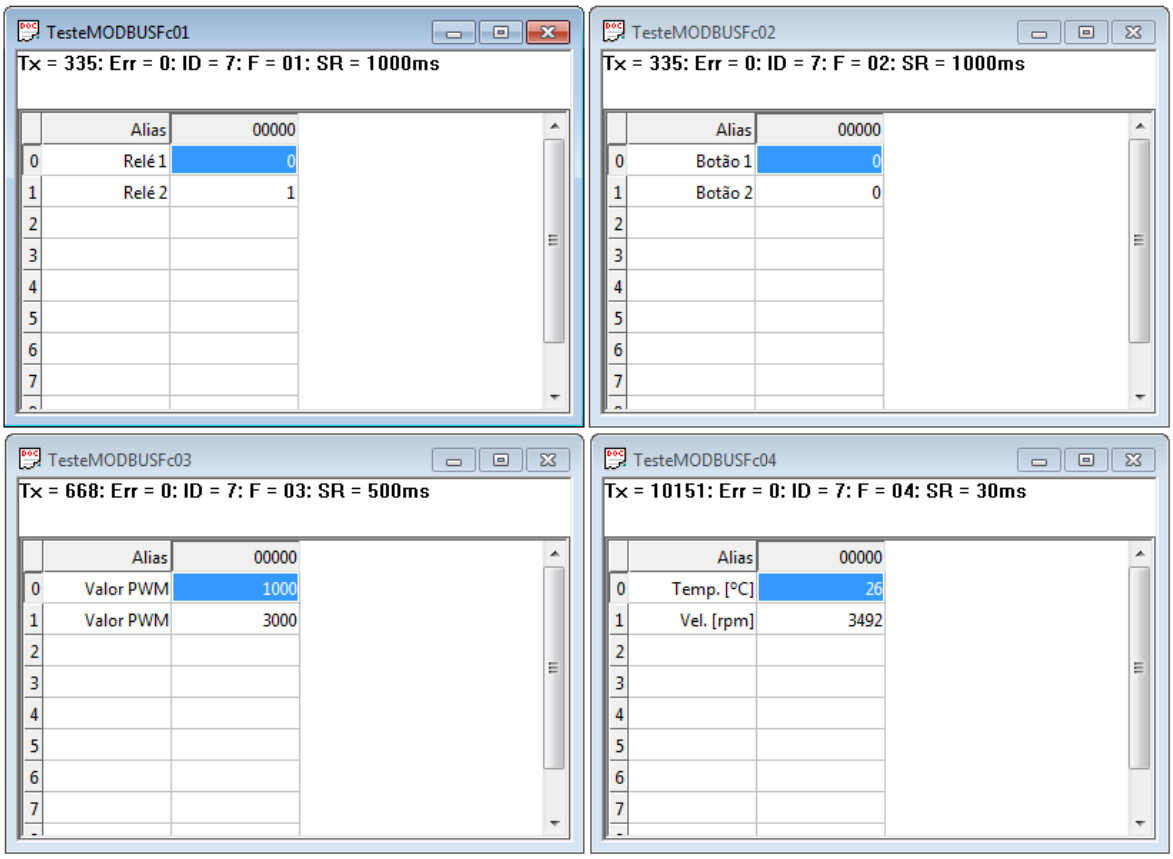

Figura 5.1 – Teste de comunicação MODBUS, funções 01, 02, 03 e 04.

Percebe-se que para a função 04, o mestre requisitou mais de 10000 vezes o dispositivo escravo e este respondeu a todas às requisições com sucesso. A taxa de requisição utilizada para a função 04 foi de 1 requisição a cada 30 ms, conforme Figura 5.1.

O teste para as funções de escrita 05 e 06, é mostrado na Figura 5.2. Percebe-se resposta OK nos 2 casos.

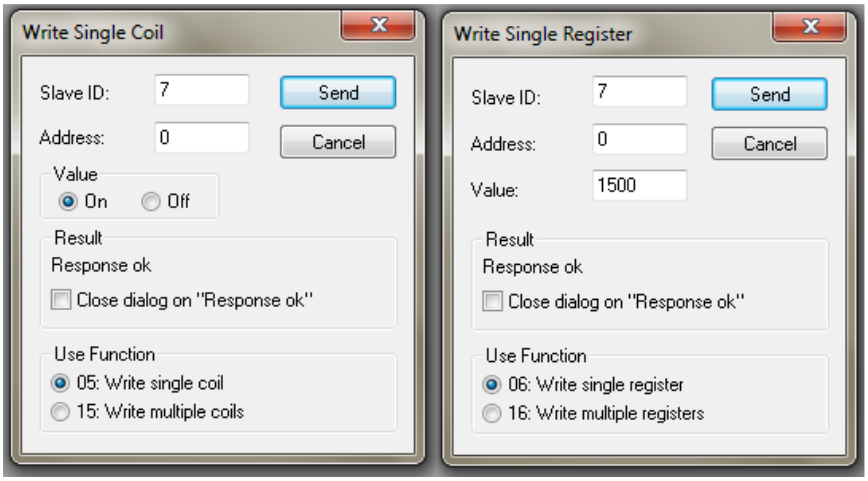

Figura 5.2 – Teste de comunicação MODBUS, funções 05 e 06.

Para testar as exception responses, foram criadas propositalmente algumas situações de erro e os resultados se encontram na Figura 5.3.

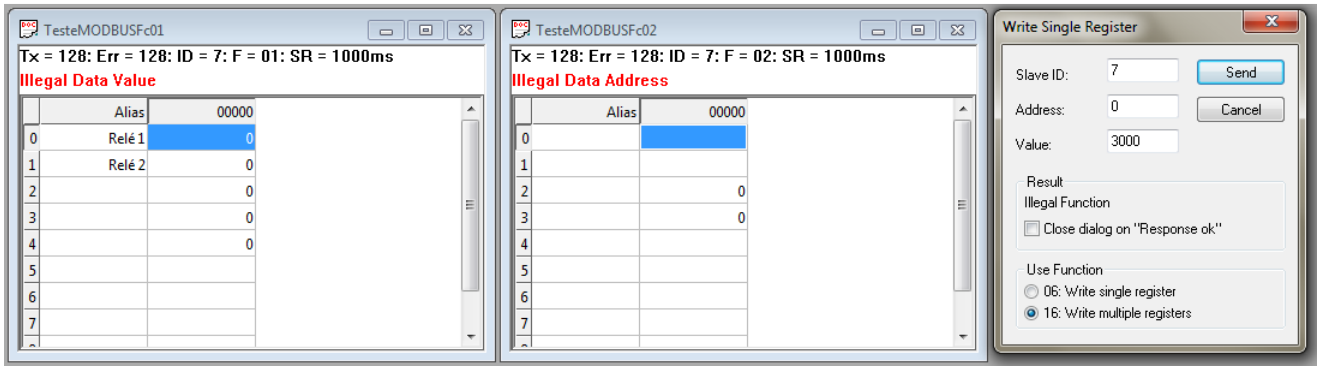

Figura 5.3 – Teste de comunicação MODBUS, exception responses.

As exception responses 01 (ILLEGAL FUNCTION), 02 (ILLEGAL DATA ADDRESS) e 03 (ILLEGAL DATAVALUE) foram corretamente geradas e enviadas pelo módulo.

Algumas outras situações de erro foram testadas e os resultados se encontram na Figura 5.4 e na Figura 5.5.

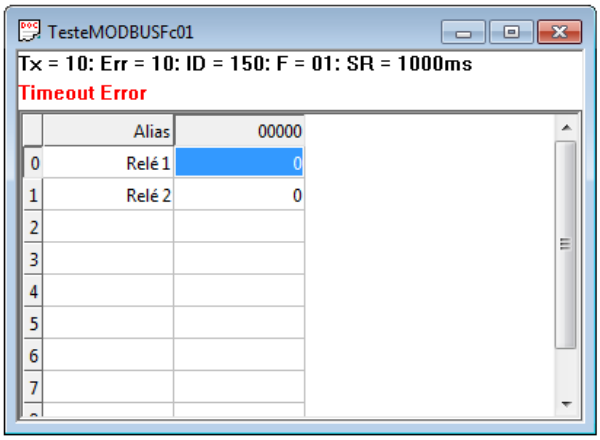

Figura 5.4 – Teste de comunicação MODBUS, endereço incorreto.

O mestre requisitou a leitura de 2 coils para o dispositivo de endereço 150. Como o módulo de I/O remoto MODBUS estava configurado com o endereço 7, ele não respondeu à requisição e o mestre acusou um erro de timeout.

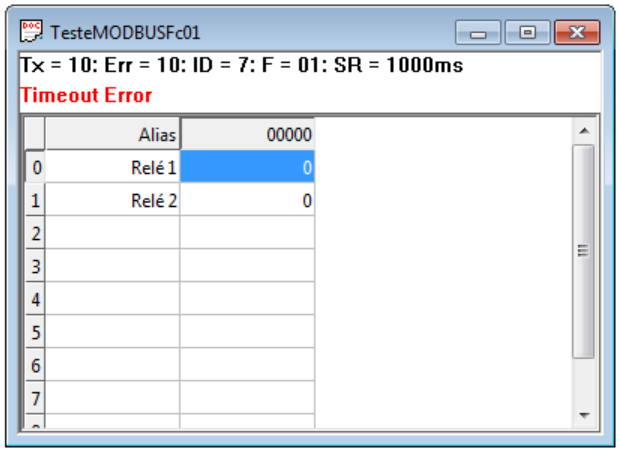

Figura 5.5 – Teste de comunicação MODBUS, paridade incorreta.

A conexão foi configurada no mestre como tendo paridade ímpar e o módulo de I/O remoto MODBUS suporta somente paridade par. Portanto, o módulo detectou erro de paridade na requisição e nenhuma resposta foi enviada ao mestre, que acusou um erro de timeout.

5.2 Teste de Desempenho

Testes demonstraram que a taxa máxima de requisições feitas pelo mestre é de 1 requisição a cada 30 ms. Uma análise teórica comprova tal resultado:

Tem-se uma taxa de transmissão de 9600 bps. Isso faz com que 1 dígito binário leve 104,17 µs para ser transmitido. Cada caractere MODBUS é composto por 11 dígitos binários (1 start + 8 bits de dados + 1 paridade + 1 stop). Portanto, cada byte leva 1,15 ms para ser transmitido. O silêncio na linha de 3,5 caracteres é de aproximadamente 4 ms.

No caso de uma requisição de leitura de 1 input register, tem-se a sequência mostrada na Figura 5.6.

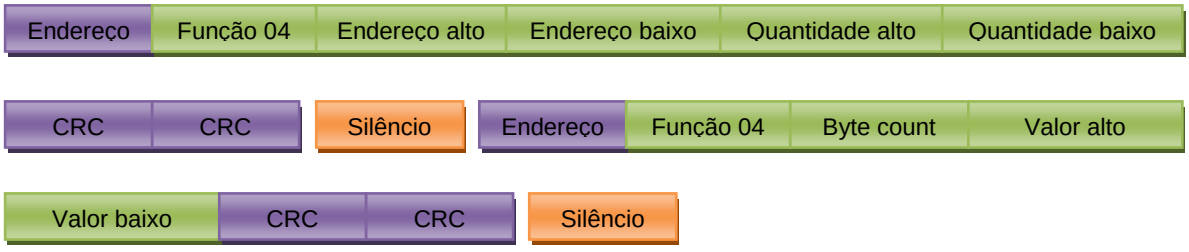

Figura 5.6 – Sequência dos caracteres em uma requisição.

Portanto, o tempo total para o mestre enviar uma requisição de leitura de 1 input register e receber o dado solicitado é de:

$$15 \times 1{,}15\ ms + 2 \times 4\ ms + t_{escravo} + t_{mestre} = 25{,}25\ ms + t_{escravo} + t_{mestre} \qquad (8)$$

Onde $t_{escravo}$ é o tempo de processando da requisição no dispositivo escravo em s e t_{mestre} é o tempo de processando da resposta no mestre em s.

A análise realizada acima supõe o melhor caso de transmissão, onde não existe atraso entre envios de caracteres.

5.3 Controle em Malha Fechada

Com o uso de um PLC da série Nexto da empresa Altus Sistemas de Informática S/A, foram realizadas 2 leis de controle em malha fechada, obtendo o controle de temperatura e de velocidade para os sistemas presentes no módulo de I/O remoto MODBUS.

A porta COM1 de uma CPU NX3010 foi configurada para utilizar o meio físico RS-232, realizando conexão ponto a ponto com o módulo de I/O. Além disso, uma conexão de 9600 bps com paridade par foi configurada.

5.3.1 Controle de Temperatura

Foram configuradas 2 relações MODBUS:

- Leitura do Input Register correspondente à temperatura da resistência, escrevendo o dado lido na variável IEC de endereço %IW0, com um polling de 30 ms;
- Escrita no Holding Register correspondente ao valor do PWM para acionar a resistência, lendo o dado a ser escrito da variável IEC de endereço %QW0, com um polling de 30 ms.

O programa mostrado na Figura 5.7 foi desenvolvido em linguagem Structured Text, uma das 5 linguagens definidas pela norma IEC 61131-3, e realiza o cálculo do valor a ser aplicado ao PWM que aciona a resistência em função do valor da temperatura e do setpoint.

As variáveis do tipo WORD Temperatura e ValorPWMTemp possuem seus endereços definidos em função das relações MODBUS citadas acima.

As funções WORD_TO_REAL() e REAL_TO_WORD() são usadas para realizar conversão entre variáveis do tipo WORD e do tipo REAL.

Os parâmetros usados para o controlador PID são os obtidos na seção 4.7.1:

$$K_p = 1000\ ;\ T_i = 2{,}5\ ;\ T_d = 0$$

A Figura 5.8 mostra o comportamento da temperatura para um setpoint de 45 ºC e para um setpoint de 55 ºC.

Percebe-se que o resultado prático é bem semelhante ao esperado, conforme simulação trazida na Figura 4.22.

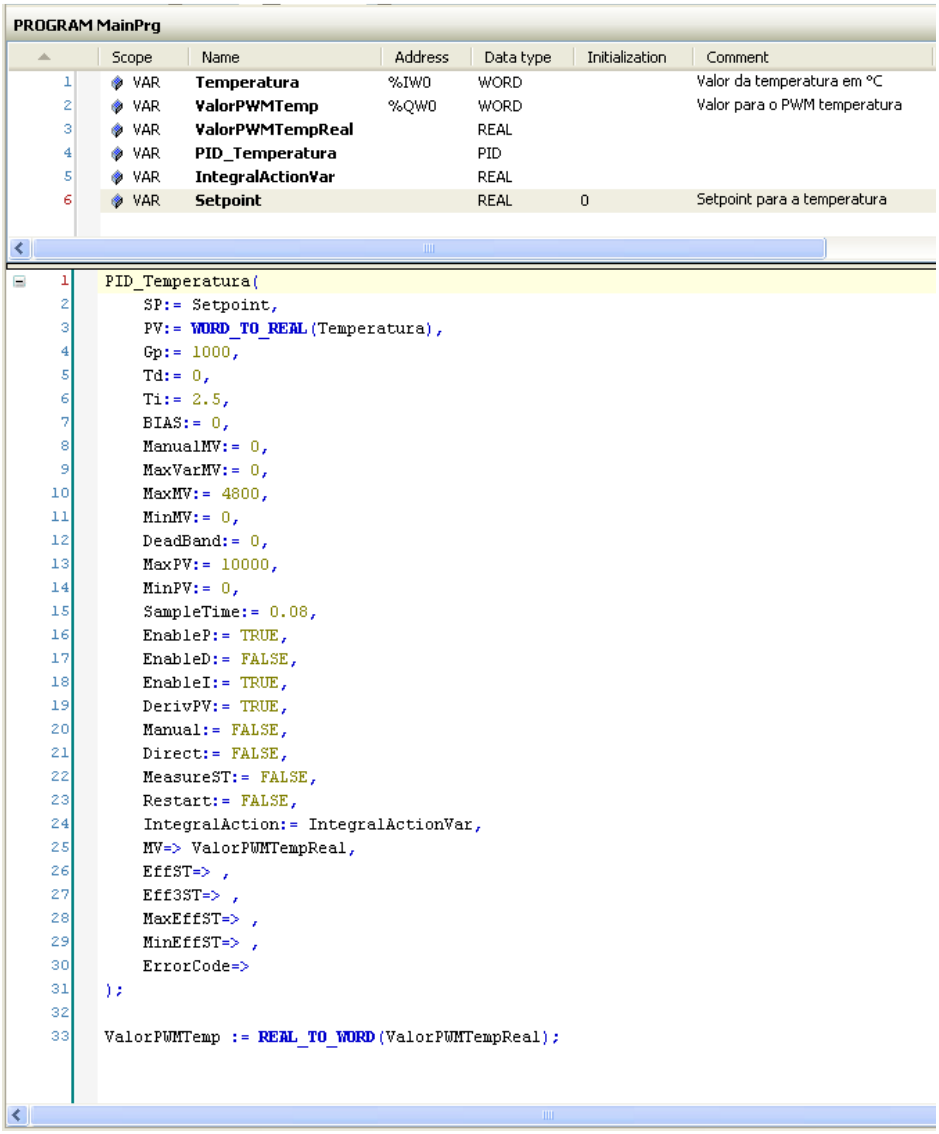

Figura 5.7 – Programa que implementa o controlador PID de temperatura.

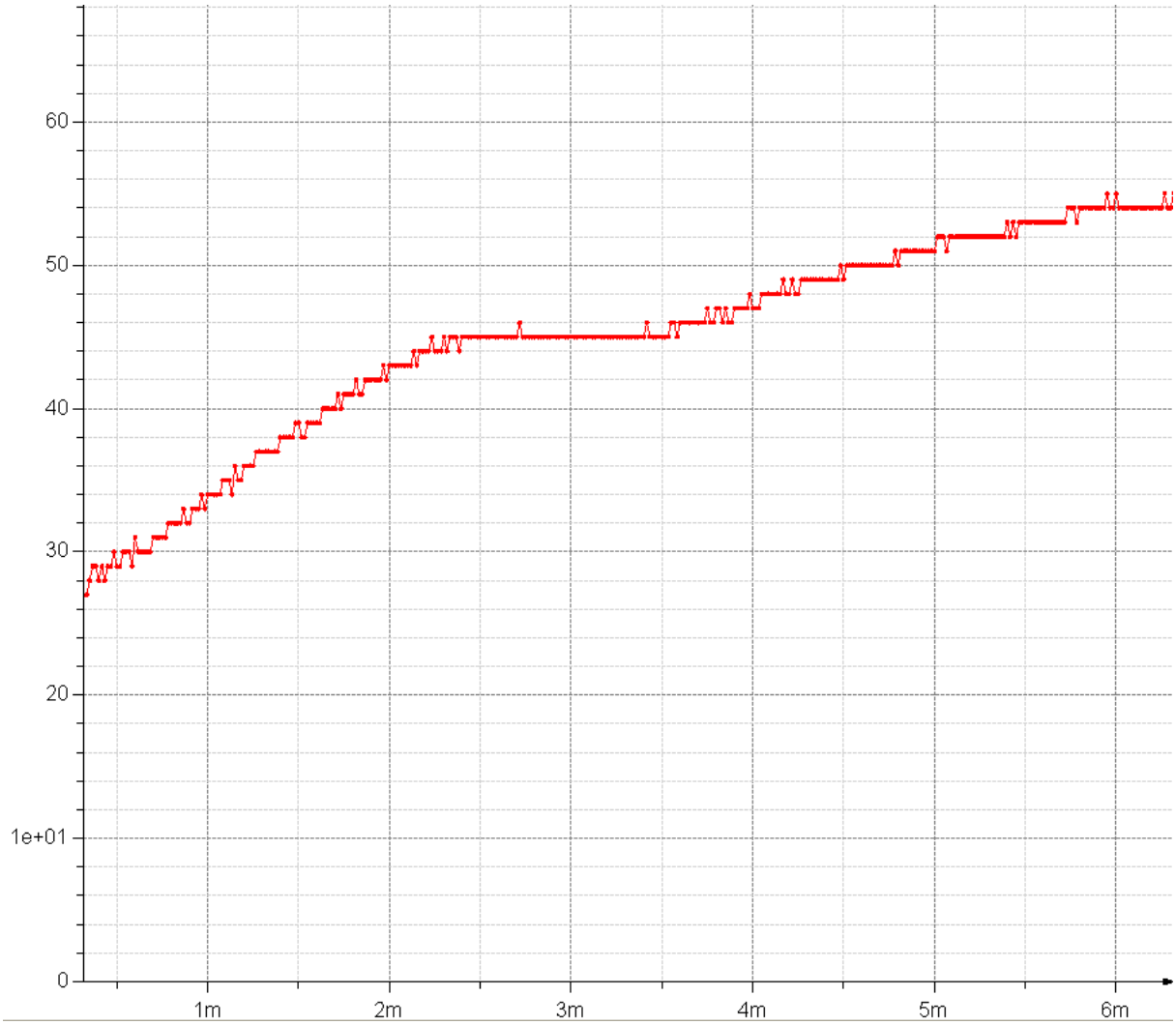

Figura 5.8 – Temperatura utilizando controle em malha fechada.

5.3.2 Controle de Velocidade

Foram configuradas 2 relações MODBUS:

- Leitura do Input Register correspondente à velocidade do motor, escrevendo o dado lido na variável IEC de endereço %IW0, com um polling de 30 ms;
- Escrita no Holding Register correspondente ao valor do PWM para acionar o motor, lendo o dado a ser escrito da variável IEC de endereço %QW0, com um polling de 30 ms.

O programa mostrado na Figura 5.9 foi desenvolvido em linguagem Structured Text, uma das 5 linguagens definidas pela norma IEC 61131-3, e realiza o cálculo do valor a ser aplicado ao PWM que aciona o motor em função do valor da velocidade e do setpoint.

As variáveis do tipo WORD Velocidade e ValorPWMVel possuem seus endereços definidos em função das relações MODBUS citadas acima.

As funções WORD_TO_REAL() e REAL_TO_WORD() são usadas para realizar conversão entre variáveis do tipo WORD e do tipo REAL.

Os parâmetros usados para o controlador PID são os obtidos na seção 4.7.2:

$$K_p = 6\ ;\ T_i = 2{,}5\ ;\ T_d = 0$$

A Figura 5.10 mostra o comportamento da velocidade para um setpoint de 4000 rpm e para um setpoint de 5500 rpm.

Percebe-se que o resultado prático é semelhante ao esperado, conforme simulação trazida na Figura 4.23. Tem-se para o primeiro setpoint um pequeno sobrepasso e uma pequena oscilação na resposta. A diferença entre o modelo proposto pela Equação 7 e o sistema real pode explicar essas diferenças entre a simulação e a prática.

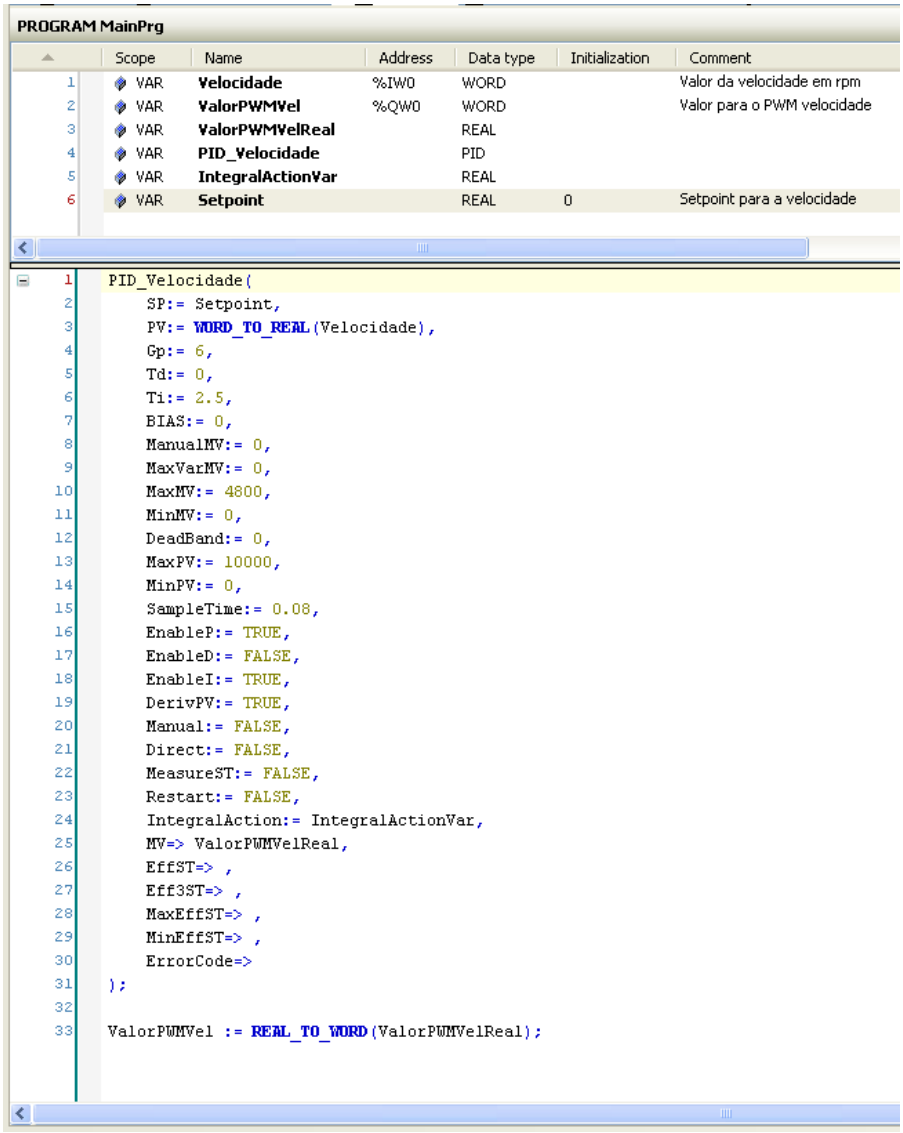

Figura 5.9 – Programa que implementa o controlador PID de velocidade.

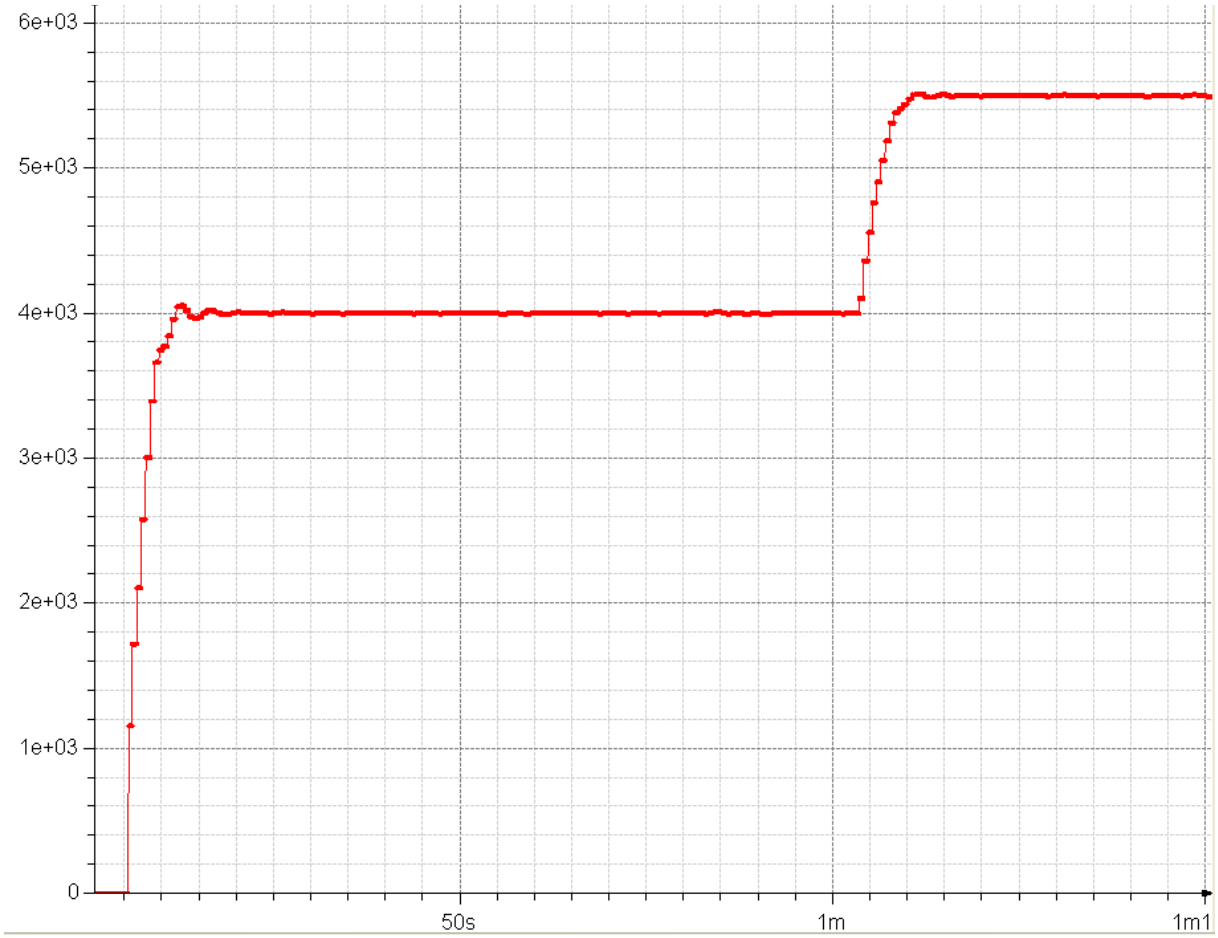

Figura 5.10 – Velocidade utilizando controle em malha fechada.

5.4 Conclusões

Os objetivos propostos na seção 1.2 foram todos atingidos com êxito.

O módulo de I/O remoto MODBUS cumpre todas as especificações feitas na seção 3.1.

O protocolo MODBUS se mostrou uma poderosa forma de comunicação serial entre dispositivos. A facilidade na operação, na configuração e a pouca exigência de hardware o tornam um dos protocolos mais utilizados em automação industrial. Além disso, o protocolo possui ótima documentação e é um protocolo aberto.

Os microcontroladores da Freescale em conjunto com o ambiente de desenvolvimento CodeWarrior são ótimas opções para projetistas de sistemas embarcados. A possibilidade de depuração in circuit proporcionada pelo gravador e depurador microBDM facilita muito o projeto e tornam simples a tarefa de localizar e corrigir erros de programação.

Em aplicações, onde o desempenho em regime transitório não seja fator preponderante, pode-se utilizar técnicas de identificação de sistemas menos sofisticadas,

como as utilizadas neste trabalho. Já o desempenho em regime permanente não é tão alterado ao realizar a identificação de um sistema através de um simples ensaio ao salto.

O módulo de I/O remoto MODBUS pode tranquilamente se tornar um produto comercial após passar por um projeto de hardware e algumas modificações de software sendo conectado em uma rede RS-485 realizando a troca de I/O com um PLC.

5.5 Propostas de Melhoria

As propostas de melhoria para o módulo de I/O remoto MODBUS são:

- Implementação de outras taxas de transmissão diferentes de 9600 bps;
- Implementação do modo sem paridade;
- Realização de mudanças de hardware para que o módulo possa utilizar o meio físico RS-485;
- Implementação das funções MODBUS 15 (Escrita em múltiplas Coils), 16 (Escrita em múltiplos Holding Registers) e 07 (Leitura de Exception status);
- Implementação de diagnósticos visuais, onde LED são usados para indicar comunicação, erro e dispositivo alimentado;
- Verificação da existência de ruído na entrada do ADC utilizado para a leitura da temperatura e implementação de um filtro para eliminá-lo;
- Realização da gravação do último endereço escravo MODBUS atribuído ao módulo na EEPROM do kit e da leitura desse valor na inicialização do módulo. Assim, na falta de alimentação, o módulo de I/O remoto MODBUS não precisaria ser reconfigurado e o sistema voltaria a funcionar corretamente após alimentação restabelecida.

REFERÊNCIAS BIBLIOGRÁFICAS

BAZANELLA, A. S.; SILVA Jr, J. M. G. **Sistemas de Controle: princípios e métodos de projeto**. Porto Alegre: Editora UFRGS, 2005. 297 p.

FREESCALE SEMICONDUCTOR. **MC9S08JM60 Data Sheet**. Rev. 3. 2009. 388 p. Disponível em: < http://www.freescale.com/files/microcontrollers/doc/data_sheet/MC9S08JM60.pdf >. Acesso em: 28/05/2011.

FREESCALE SEMICONDUCTOR. **HCS08 Peripheral Module Quick Reference**. Rev. 1. 2006. 116 p. Disponível em: < http://www.freescale.com/files/microcontrollers/doc/user_guide/HCS08QRUG.pdf >. Acesso em: 28/05/2011.

FREESCALE SEMICONDUCTOR. **CodeWarrior for Microcontrollers V10.x Quick Start**. 18 p. Disponível em: < http://cache.freescale.com/files/soft_dev_tools/doc/quick_ref_guide/926-77846.pdf?fpsp=1 >. Acesso em: 28/05/2011.

FREESCALE SEMICONDUCTOR. **CodeWarrior for Microcontrollers V10.x Getting Started Guide**. 2011. 70 p. Disponível em: < http://cache.freescale.com/files/soft_dev_tools/doc/user_guide/CW_MCU_10.1_UM.pdf?fpsp=1 >. Acesso em: 28/05/2011.

MICROCHIP TECHNOLOGY. **MCP9700 Data Sheet**. 2009. 24 p. Disponível em: < http://ww1.microchip.com/downloads/en/DeviceDoc/21942e.pdf >. Acesso em: 23/06/2011.

MODBUS-IDA. **MODBUS Application Protocol Specification V1.1b**. 2006. 51 p. Disponível em: < http://www.modbus.org/docs/Modbus_Application_Protocol_V1_1b.pdf >. Acesso em: 12/05/2011.

MODBUS-IDA. **MODBUS over Serial Line Specification and Implementation Guide V1.02**. 2006. 44 p. Disponível em: < http://www.modbus.org/docs/Modbus_over_serial_line_V1_02.pdf >. Acesso em: 12/05/2011.

ROSÁRIO, J. M. **Princípios da Mecatrônica**. São Paulo: Editora Pearson Prentice Hall, 2005.

SILVEIRA , P. R.; SANTOS , W. E. **Automação e Controle Discreto**. São Paulo: Editora Érica, 1998. 256 p.

ZELENOVSKY, R.; MENDONÇA, A. **PC: Um Guia Prático de Hardware e Interfaceamento**. 4ª edição. Rio de Janeiro: Editora MZ, 2006. 1175 p.