UNIVERSIDADE FEDERAL DE MINAS GERAIS – UFMG
ESCOLA DE ENGENHARIA

Rede de Instrumentação para Aplicação em Controle Embarcado

Tiago Amadeu Arruda

Belo Horizonte
Novembro de 2009

TIAGO AMADEU ARRUDA

Rede de Instrumentação para Aplicação em Controle Embarcado

Projeto de Final de Curso apresentado ao Colegiado de Engenharia de Controle e Automação da Universidade Federal de Minas Gerais como requisito para obtenção do título de Engenheiro de Controle e Automação.

Orientador: Prof. Guilherme Augusto Silva Pereira / DEE - UFMG

Supervisor: Prof. Leonardo Antonio Borges Torres / DELT - UFMG

Belo Horizonte
Novembro de 2009

Agradecimentos

Agradeço a Deus pelo dom da vida e por sua presença constante durante esta graduação. Pela saúde, proteção, ânimo e disposição sem limites que por Ele me foram dados e me permitiram chegar onde cheguei.

Agradeço minha família pelo apoio, pelo incentivo e pela coragem. Sem ela, eu não teria esta vitória. Obrigado por todos os momentos que compartilhamos por sorrirem comigo nas horas felizes e por oferecerem um ombro amigo nas horas difíceis.

Agradeço a todos os professores dos quais fui aluno, que não mediram esforços na transmissão do conhecimento e me forneceram a base para crescer no campo acadêmico, profissional e pessoal.

Em especial quero agradecer aos professores Guilherme Pereira e Leonardo Torres que se dispuseram a me auxiliar na elaboração e conclusão deste trabalho final de curso, fornecendo os recursos necessários e esclarecendo as muitas dúvidas que surgiram durante o desenvolvimento do mesmo.

Resumo

Este trabalho mostra a implementação completa de uma rede baseada em Modbus sobre a camada física RS-485 com o objetivo de substituir o modelo de comunicação via conexão USB, atualmente utilizado no carro autônomo desenvolvido pelo Grupo de Pesquisa e Desenvolvimento de Veículos Autônomos (PDVA) da UFMG. O controle de qualquer sistema é baseado em três blocos principais sendo eles o bloco sensor, o bloco atuador e o bloco controlador. Para extração do máximo potencial no uso destes blocos faz-se necessária a utilização de uma comunicação confiável e de alta velocidade entre eles. Após uma revisão geral sobre os protocolos de comunicação atualmente utilizados em automação, o protocolo Modbus sobre RS-485 mostrou-se o mais promissor no cumprimento dos requisitos exigidos neste trabalho: Comunicação determinística, alta taxa de rejeição a ruídos, baixo custo e escalabilidade. Os testes mostraram a viabilidade desta rede e seu potencial para substituir com muitas vantagens a comunicação USB. Sua implementação em microcontrolador utilizou 3% e 15% respectivamente da memória de programa e memória RAM do PIC utilizado. Em Modbus o envio e recepção de mensagens acontece nos modos *unicast* e *broadcast* entre os nós mestres e escravos. E a especificação Modbus/RS-485 permite o uso de até 32 nós na rede sem o uso de repetidores. O barramento polarizado em conjunto com os resistores de terminação de linha garantem a máxima rejeição a perturbação, podendo esta ainda ser maior com o uso de conectores e cabos blindados.

Palavras chave: Rede de Instrumentação; Modbus; Rede de Sensores; Rede Determinística; Sistemas Embutidos

Abstract

This work shows the complete implementation of a network solution based on Modbus over RS-485, in order to replace the present communication system based in USB connections in an autonomous car that has been developed by Grupo de Pesquisa e Desenvolvimento de Veículos Autônomos (PDVA) of UFMG. The control of any system is based on three main elements called sensor, actuator and controller. To get the maximum potential of this devices it is necessary a reliable and fast communication among them. After a overview of the most common networks actually available to be used in automation solutions, the Modbus over RS-485 showed to be the the most suitable choice once it matches all the requirements in this work: Deterministic communication, high disturbance rejection ratio, low cost and scalability. The test showed the feasibility of this network and its potential to replace with many advantages the USB communication. Its implementation in microcontroller used 3% and 15% of the PIC's Flash and RAM memory. Modbus protocol allows messages in unicast and broadcast modes between master and slaves nodes. And the Modbus/RS-485 specification allows until 32 nodes in the bus whitout use of repeater. The polarized cable and the line termination resistor allows a higher disturbance rejection and its use with shielded connectors and cables became this protection more effective.

Keywords: Instrumentation Network; Modbus; Sensor Network; Deterministic Network; Embedded Systems

Siglas e Abreviaturas

PDVA	Grupo de Pesquisa e Desenvolvimento de Veículos Autônomos
UFMG	Universidade Federal de Minas Gerais
CORO	Laboratório de Computação e Robótica
ADU	Do inglês, *Application Data Unit*
ASCII	Do inglês, *American Standard Code for Information Interchange*
ASI	Do inglês, *Actuator Sensor Interface*
CAN	Do inglês, *Controller-Area Network*
CLP	Controlador Lógico Programável
CRC	Do inglês, *Cyclic Redundancy Check*
CSMA/CA	Do inglês, *Carrier Sense Multiple Access with Collision Avoidance*
DIN	Do alemão, *Deutsches Institut für Normung*
EN	Do inglês, *European Standard*
FF	Do inglês, *Foundation Fieldbus*
IEC	Do inglês, *International Electrotechnical Commission*
PDU	Do inglês, *Protocol Data Unit*
RAM	Do inglês, *Random Access Memory*
RTU	Do inglês, *Remote Terminal Unit*
USB	Do inglês, *Universal Serial Bus*

Conteúdo

Lista de Figuras

Capítulo 1

Introdução

1.1 Apresentação do Laboratório

Este projeto foi realizado na Universidade Federal de Minas Gerais no Laboratório de Sistemas de Computação e Robótica (CORO) onde estão disponíveis os equipamentos, ferramentas e materiais necessários a execução do mesmo.

O CORO é um laboratório de pesquisa que conta com a participação de professores, mestrandos, alunos de graduação e estagiários técnicos, e tem projetos nas seguintes áreas:

- Desenvolvimento de Sistemas de Instrumentação, Navegação e Controle de Aeronaves;
- Desenvolvimento de Sistemas de Hardware e Software para localização, Navegação e Controle de Veículos Autônomos em Ambientes Externos;
- Localização e Identificação de um Mini-Helicóptero;
- Desenvolvimento de um Sistema de Localização e Reconstrução de Trajetórias para Veículos Terrestres;
- Comparação de Estriamentos de Projéteis Propelidos por Armas de Fogo Utilizando Função de Correlação Cruzada.

O Laboratório de Sistemas de Computação e Robótica do Departamento de Engenharia Elétrica se encontra na Av. Antônio Carlos 6627 Belo Horizonte-MG, CEP 31270-910, Prédio da Engenharia Sala 2212. Os responsáveis pelo laboratório é o professor Guilherme Augusto Silva Pereira.

1.2 Motivação

Em controle realimentado existem três blocos principais sem os quais o sistema não é funcional [4]. Estes blocos são definidos como sensor, controlador e atuador. O sensor é o responsável por informar o valor da variável de saída da planta e é utilizado para levantamento do modelo e posteriormente para o seu controle. O controlador, por sua vez, envia um sinal a planta de modo que a saída real se iguale a saída desejada. Entre o controlador e a planta situa-se o atuador que é uma interface entre sinais de comando e sinais de atuação.

Para sistemas industriais, existem diversas soluções já em uso que resolvem os problemas de comunicação entre os dispositivos dispostos ao longo da planta. Isto ainda é facilitado pois, via de regra, grande parte do equipamento e software de controle possui recursos de comunicação que permitem interligá-los e fazê-los trabalhar conjuntamente [29].

Este trabalho visou a pesquisa e implementação um protocolo que fosse útil para trafegar dados de diversos sensores, e/ou outros equipamentos, espalhados ao longo de um veículo autônomo. A rede utilizada atende especificações para garantir um sistema de comunicação confiável e determinístico de modo que a informação seja válida para estimação de modelos, levantamento de parâmetros e fornecimento de dados confiáveis ao bloco controlador do sistema em questão. No laboratório CORO existem placas de circuito eletrônicos para desenvolvimento de aplicações sobre a qual a rede foi implementada.

O Veículo autônomo em questão é um Chevrolet Astra 16V, câmbio automático, direção hidráulica e freios ABS mostrado na Figura 1.1. Ele está atualmente equipado com sistemas de controle de freio, de câmbio, direção e aceleração e possui quatro sensores sendo eles um GPS, um sensor de ângulo do volante, uma IMU e um sensor de velocidade das rodas. Todos estes equipamentos se comunicam via USB com um PC e a introdução da rede visa substituir este sistema, mantendo porém os nós e suas respectivas funções.

Figura 1.1: Veículo Autônomo desenvolvido pelo PDVA.

1.3 Escopo

O escopo deste projeto se restringiu inicialmente ao estudo das redes disponíveis e, obtida uma que seja compatível com a proposta, adaptá-la de modo a utilizá-la nos projetos do CORO. Este trabalho não visou a criação de um novo protocolo de rede.

1.4 Revisão Bibliográfica

Um computador, seja ele de uso geral ou dedicado, pode ser utilizado para a coleta, armazenamento e manipulação de dados. Para a obtenção destes dados, que geralmente se encontram em meio externo ao computador, faz-se necessário o uso de periféricos de entrada e saída [31] como porta serial, porta paralela ou porta USB. Porém, uma limitação inerente destes periféricos no seu uso em sistemas embarcados é a necessidade de pelo menos uma porta para cada dispositivo implicando em uso de extensores e adaptadores para sua interligação.

Outra opção é a utilização de uma rede para interligar todos os dispositivos, sendo assim necessário somente uma interface de comunicação entre o computador e o restante

dos elementos. Este tipo de abordagem permite que os sensores, atuadores ou outros dispositivos sejam conectados a um barramento, de maneira que a comunicação entre o computador e os nós (nome dado a um elemento genérico da rede) seja feita por algum protocolo específico.

Nesta revisão, realizou-se uma busca pela soluções existentes em redes de instrumentação para que, dentre as opções disponíveis, fosse escolhida uma que atendesse determinadas especificações necessárias para seu uso em um sistema embarcado. Dentre as principais características buscadas estão:

- Determinismo - O tempo de transporte dos dados é consistente, previsível e possui repetibilidade sob certas condições especificadas;
- Baixa latência - O tempo necessário para a comunicação está abaixo de certo parâmetro a ser determinado;
- Escalabilidade - Habilidade de manipular uma porção crescente de trabalho de forma uniforme, ou estar preparado para crescer [1].
- Rejeição a Ruídos - Os sinais trafegados na rede devem sofrer pouca ou nenhuma interferência externa em seus sinais para que haja integridade na comunicação.
- Fácil implementação em microcontrolador.

1.4.1 Principais Redes Utilizadas em Automação

Atualmente, existem muitas soluções industriais disponíveis para comunicação de dados entre dispositivos, sendo que algumas delas, dependendo de suas características [7], podem ou não atender ao objetivo buscado no desenvolvimento deste trabalho. Algumas destas soluções são listadas a seguir:

CAN:

O protocolo CAN *(Controller Area Network)* é uma rede multi-mestre com resolução de colisão baseada em prioridade de mensagens criada por Robert Bosh nos anos 80 [6][3]. Com vistas a suprir a necessidade de interconexão entre diversos dispositivos presentes em um veículo, como controle de tração, sistemas de freios ABS e demais funções, esta rede foi desenvolvida como sendo de tempo real, baixo custo, alta taxa de rejeição a ruídos e alta velocidade de transmissão de dados podendo chegar até

1Mbps. Devido a estas características esta rede foi largamente adotada tanto pela indústria automotiva quanto pela indústria de automação. A especificação CAN define os níveis físico e de link de dados (níveis 1 e 2) da camada ISO [34]. O protocolo CAN é baseado na troca de mensagens assíncronas entre os diversos nós, onde cada nó possui um endereço único na rede e um endereço compartilhado de *broadcast*. No barramento não são necessários árbitros ou um mestre, e a colisão é resolvida através do protocolo CSMA/CA, como indicado em [22]. Além destas características, o uso de filtros e máscaras como previsto na especificação [2] permite o uso desta rede para os mais diversos fins.

Os quadros de mensagens CAN são formados por um campo de endereço, um campo de função, um campo indicando o número de bytes a ser enviados e o campo com os bytes de dados que podem ter até 8 bytes. Além destes, há bytes de detecção de erro CRC e bytes de controle. O quadro CAN pode ser visto na Figura 1.2.

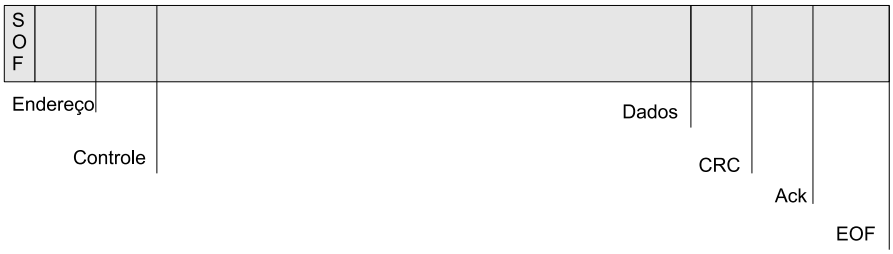

Figura 1.2: Quadro de Mensgem padrão do protocolo CAN adaptado de [2]

DeviceNet:

DeviceNet [21] é um exemplo de tecnologia baseada em especificação CAN [2] que recebeu considerável aceitação em aplicações no nível de dispositivos. Sua especificação é aberta [23] e gerenciada pela DeviceNet Foundation. Esta rede é determinística e permite a comunicação por meio do protocolo Produtor/Consumidor o qual pode emular qualquer outro tipo de comunicação como mestre-escravo, produção cíclica de dados e outras configurações. Apesar de derivar do protocolo CAN, esta rede possui características físicas adicionais que dificultam a sua implementação em sistemas embarcados, favorecendo o seu uso como rede industrial. Dentre as características no nível físico exigidas estão:

- Topologia física básica do tipo linha principal com derivações;

- Barramentos separados de par trançado para a distribuição de sinal e de alimentação (24VCC), ambos no mesmo cabo;
- Uso de terminadores de 121 ohms em cada fim de linha.

Este protocolo também permite a inserção de instrumentos a quente, ou seja, que podem ser introduzidos na mesma sem que a alimentação tenha que ser desligada.

Profibus:

Esta rede, como descrito em [10], foi concebida a partir de 1987 e adota um mecanismo híbrido de acesso com uma hierarquia mestre-escravo para troca de dados cíclicos. Um *token* é passado por entre os mestres para compartilhar o barramento de controle. A rede está padronizada através da norma DIN 19245 incorporada na norma européia Cenelec EN 50170 e é dividida em Profibus-PA para conexão com dispositivos como sensores e atuadores e Profibus-DP que possui comunicação mais rápidas próprias para transmissão de grandes massas de dados entre CLP(Controlador Lógico Programável)/Servidor ou Servidor/Servidor e aplicações de controle discreto operando sobre RS-485 com taxas de transmissão de até 12Mbps. O protocolo de comunicação utilizado nesta rede é do tipo mestre-escravo onde:

Dispositivo Mestre - É capaz de enviar mensagens independente de solicitações externas quando tiver a posse do token. São também chamados de estações ativas.

Dispositivos Escravos - Não possuem direito de acesso ao barramento e podem apenas confirmar o recebimento de mensagens ou responder a uma mensagem enviada por um mestre. São também chamadas de estações passivas. Sua implementação é mais simples e barata que a dos mestres.

Um *token* é passado para cada estação segundo sua posição no anel lógico (endereços crescentes) dentro de um tempo bem determinado. O tempo de retenção do *token* por cada mestre é determinado pelo tempo de rotação do mesmo, que é configurável. A comunicação em Profibus é independente de conexão, o que permite executar uma comunicação *broadcast* (uma estação envia uma mensagem sem reconhecimento para todas as demais, mestres ou escravos) ou *multicast* (uma estação ativa envia uma mensagem sem reconhecimento para um determinado grupo de estações mestre ou escravos). Sua especificação é aberta e pode ser encontrada em [26].

Foundation Fieldbus

A rede Foundation Fieldbus(FF) descrita em [33] é uma proposta especial para redes industriais de tempo real. Redes FF são geralmente caracterizadas pela necessidade de respeitar estritamente restrições temporais em instrumentos de campo. A FF é uma rede digital cuja padronização levou mais de dez anos para ser concluída. Existem duas redes FF, uma de baixa velocidade concebida para interligação de instrumentos (H1 - 31,25 Kbps) e outra de alta velocidade utilizada para integração das demais redes e para a ligação de dispositivos de alta velocidade como CLPs (HSE - 100 Mbps). Tanto a FF-H1 como a rede Profibus-PA têm sua camada física padronizada pela norma IEC 61158-2. Os sinais H1 são codificados utilizando codificação *Manchester* Bifase-L. Trata-se de uma comunicação síncrona que envia os sinais de dados combinados com o relógio. Este dispositivo utiliza-se do modelo mestre-escravo de comunicação e é determinístico com uma precisão de 1ms. Para este fim, os relógios dos elementos presentes na rede devem ser sincronizados periodicamente. Na FF o LAS (*Link Active Scheduler*) é o dispositivo que controla a comunicação no barramento e é também responsável pela sincronização no barramento.

Assim como o Profibus, a FF depende de cabeamento especial (polarizado) e fontes de alimentação que pode variar de 9 a 32V. As informações técnicas sobre a FF estão em [27].

Ethernet Industrial:

Recentemente, a Ethernet Industrial(uso de *switches* e *hubs* inteligentes em modo *full-duplex*) se tornou uma alternativa muito promissora para aplicações de tempo real na indústria, devido a eliminação das incertezas inerentes da Ethernet tradicional, como pode ser visto em [12]. O protocolo Ethernet tende a disputar os espaços hoje ocupados pelo Profibus e pela Foundation Fieldbus devido a sua simplicidade, alta velocidade e aceitação.

Modbus:

MODBUS é um protocolo de mensagens e provê uma comunicação cliente/servidor entre dispositivos conectados em diferentes tipos de barramentos ou redes [19]. Pode ser dividido em dois principais modos sendo eles:

- Protocolo de linha serial;
- Protocolo TCP/IP sobre Ethernet.

No modo serial a camada física compreende os protocolos seriais conhecidos RS-232, RS-422 e RS-485. No outro modo, como indicado, a camada física é a Ethernet. Este é talvez o protocolo de mais larga utilização em automação industrial, pela sua simplicidade e facilidade de implementação. Em Modbus um único dispositivo, o mestre, pode iniciar transações denominadas *queries*. O demais dispositivos da rede (escravos) respondem, suprindo os dados requisitados pelo mestre ou executando uma ação por ele comandada. Os papéis de mestre e escravo são fixos, quando se utiliza a comunicação serial, mas em outros tipos de rede, um dispositivo pode assumir ambos os papéis, embora não simultaneamente. Em Modbus serial existem dois modos de transmissão: ASCII (*American Code for Information Interchange*) e RTU (*Remote Terminal Unit*), que são selecionados durante a configuração dos parâmetros de comunicação.

No modo ASCII cada byte de mensagem é enviado como dois caracteres ASCII. Durante a transmissão, intervalos de até um segundo entre caracteres são permitidos, sem que a mensagem seja truncada. Algumas implementações fazem uso de tais intervalos de silêncio como delimitadores de fim de mensagem, em substituição à sequência `cr+lf` que consiste no envio dos bytes `0x0D` e `0x0A` ao fim de cada mensagem.

No modo RTU cada byte de mensagem é enviado como um byte de dados. A mensagem deve ser transmitida de maneira contínua, já que pausas maiores que 1,5 caracter provocam truncamento da mesma.

No modo Modbus/TCP o protocolo provê uma comunicação cliente/servidor entre os dispositivos na rede Ethernet. Há quatro mensagens neste modo sendo elas *Modbus request, Modbus confirmation, Modbus indication* e *Modbus response*, cada qual com uma função específica como pode ser verificado em [13].

ASI:

AS-i é um acrônomo para *Actuator Sensor Interface* (Interface sensor-atuador)[11]. A AS-i foi projetada como um padrão aberto pela associação alemã ASI *Association* adotando um padrão para ser utilizado em dispositivos de diversos países. O padrão ASI define a comunicação e um gerenciamento pertinente desta comunicação para os elementos da rede tais como controladores, sensores e atuadores[30].

Esta rede utiliza topologia mestre/escravo, é determinística e possui um tempo de ciclo de 5ms para leitura de até 31 dispositivos na rede. Possui uma série de parâmetros para comunicação inteligente entre o mestre o os escravos. O padrão de comunicação é

aberto, mas assim como em Profibus e FF, a rede ASI também depende de cabeamento e fonte de alimentação específica.

1.4.2 Conclusão Acerca das Redes

Uma análise das redes previamente especificadas no Capítulo 1 levaram à escolha do protocolo Modbus em modo serial RTU(*Remote Terminal Unit*) para ser a base no restante do desenvolvimento deste trabalho.

As redes *Devicenet, Profibus,Foundation Fieldbus* e *ASI* cumpriam aos requisitos de tempo real, porém, além de necessitar de conectores e cabos especiais, sua implementação em microcontrolador é dificultada pela gama de recursos oferecida nas mesmas.

A rede Ethernet Industrial foi descartada pela necessidade da adaptação de controladores de redes em todos os dispositivos, como mencionado anteriormente.

A rede CAN cumpre a todos os requisitos impostos, porém a mesma fornece funcionalidades extras, como possibilidade de nós autônomos ou rede multi-mestre, que não são necessárias neste projeto e dificultam a sua implementação em tempo hábil. Deste modo, a partir do próximo capítulo é descrito o protocolo escolhido, as funcionalidades oferecidas, suas principais características e o meio físico no qual o mesmo será implementado.

1.5 Objetivo

Este trabalho objetivou a obtenção de um sistema de comunicação em um Modbus sobre RS-485 e a adaptação do mesmo ao hardware existente no laboratório para utilizá-lo em um veículo autônomo. Deste modo, independente da função do nó, o mesmo poderá ser introduzido na rede com todas as funcionalidades que o protocolo permitir.

Assim pode-se enumerar a sequência de atividades realizadas para alcançar o objetivo final:

1. Estudo dos protocolos de redes disponíveis e suas aplicações;
2. Definição do protocolo de rede Modbus e da camada física RS-485 para implementação do projeto;
3. Construção do hardware do nó;

4. Construção do software a ser implementado num PC e nos nós micro controlados, respectivamente o mestre e os escravos;

5. Definição das funções executadas pelos nós de modo a facilitar a integração com os sistemas pré-existentes;

6. Testes da rede em bancada;

7. Análise dos resultados;

8. Teste da rede no veículo.

1.6 Estrutura da Monografia

Esta monografia é composta por cinco capítulos sendo o primeiro um resumo geral do projeto, sua proposta e sua base teórico-científica.

No Capítulo 2 é apresentada a rede escolhida e um detalhamento maior de sua especificação, suas limitações e o meio físico utilizado. Além de definir a sequência de procedimentos tomados para a implementação da mesma.

No terceiro capítulo são apresentados os testes iniciais em bancada com o sistema pronto e a partir dos resultados, descrever pequenas mudanças realizadas.

Executadas as modificações pertinentes, o Capítulo 4 apresenta os procedimentos realizados para testes em um sistema embarcado real (um veículo autônomo).

O último capítulo, Capítulo 5, contem as conclusões sobre o projeto e futuras implementações.

Capítulo 2

Descrição da Rede

2.1 Introdução

A partir da definição da rede utilizada neste trabalho, como pode ser visto no Capítulo 1, o estudo apresentado neste capítulo visou mostrar os detalhes da rede escolhida e da camada física da mesma, de modo tal que permitiu a sua implementação a partir do Capítulo 3.

2.2 Protocolo Modbus - Visão Geral

Modbus é um protocolo de mensagens que provê comunicação Cliente/Servidor, ou também chamada Mestre/Escravo entre dispositivos conectados em diferentes tipos de barramentos e redes, como mostrado na Figura 2.1. Este protocolo de oferece serviços especificados por códigos de funções que são componentes do Quadro de Mensagem. Sua descrição do ponto de vista do modelo OSI não fornece uma definição clara do nível ao qual o mesmo se encaixa. A referência [28] se refere ao mesmo como sendo da camada de enlace no nível 2 enquanto que a sua especificação [19] o coloca como sendo um protocolo da camada 7 de aplicação por conta de sua inter-operabilidade entre diferentes redes.

2.2.1 O Quadro Modbus

O protocolo MODBUS define um núcleo padrão para todas as suas mensagens, independente da camada de comunicação utilizada, denominada Unidade de Dados do Protocolo - PDU (*Protocol Data Unit*). O mapeamento do protocolo Modbus em um

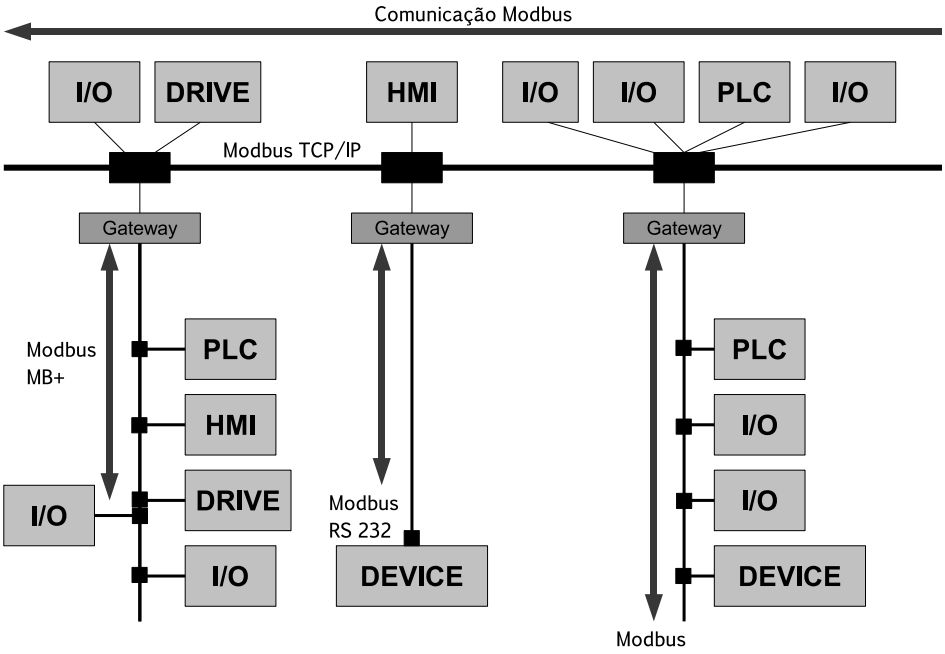

Figura 2.1: Comunicação Modbus Inter-redes.

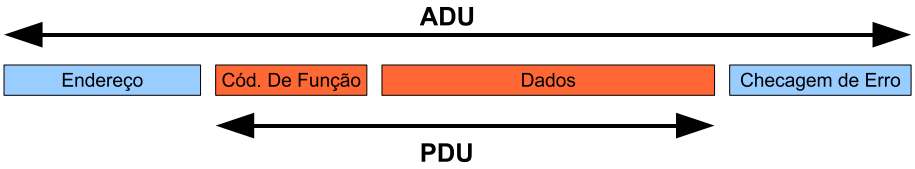

Figura 2.2: Quadro de Mensagens Padrão Modbus.

barramento ou rede específica pode adicionar alguns campos que juntos à PDU são denominamos Unidade de Dados de Aplicação - ADU (*Application Data Unit*), conforme mostrado na Figura 2.2.

A ADU Modbus é construída pelo Mestre/Cliente que inicia uma transação. O código de função indica ao Escravo/Servidor qual tipo de ação tomar. O protocolo de aplicação Modbus estabelece o formato de uma requisição iniciada pelo cliente.

O campo de função de uma unidade de dados Modbus é codificada em um byte. Os códigos válidos estão no intervalo de 1 a 255 decimal onde o intervalo 128 a 255 é reservado para uso em respostas a exceções. Quando uma mensagem é enviada de um Mestre para um dispositivo Escravo, o campo de função diz ao mestre qual tipo de ação exercer.

Opcionalmente, o campo de dados da mensagem enviada de um Mestre para um

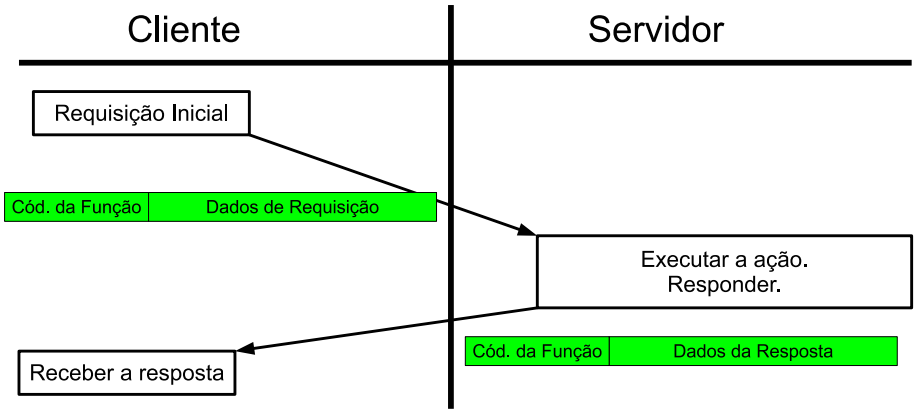

Figura 2.3: Transação Modbus livre de erros, adaptado de [19].

Escravo pode conter informações adicionais para auxiliar o processamento da ADU no mesmo. Isto pode incluir itens como o tamanho do quadro de mensagem, o número de bytes de dados ou até mesmo sub-funções, que são adicionadas ao campo de dados do quadro, para definir múltiplas ações sobre o mesmo código de função.

O campo de dados pode não existir (tamanho zero) em certas circunstâncias das requisições. Neste caso, o código da função sozinho especifica a ação por completo de modo que o Escravo não requer nenhuma informação adicional para processar a mensagem.

Se nenhum erro for encontrado na função requisitada, o receptor da ADU Modbus responderá a solicitação. Se houver uma ocorrência de erro, o campo da função será retornado pelo Escravo contendo o código de uma das exceções a que o sistema está sujeito. As Figuras 2.3 e 2.4 mostram as respectivas respostas do Escravo em caso normal e em caso de exceção.

O tamanho do PDU Modbus é limitado por herança pela primeira implementação do Modbus em Rede de Linha Serial. O seu tamanho máximo é calculado tomando-se os 256 bytes de dados máximo permitido para transmissão serial, e subtraindo-se os campos de Endereço do Mestre de 1 byte e do CRC(*Cyclical Redundancy Checking*) de 2 bytes. Deste modo, o PDU Modbus tem um tamanho máximo para comunicação de 253 bytes. Isto garante que os quadros transitando em RS232 ou RS485 tenham o tamanho máximo de 256 bytes.

A especificação do protocolo Modbus também divide as PDUs em três categorias diferentes para facilitar a classificação dos mesmos de acordo com os dados que trafe-

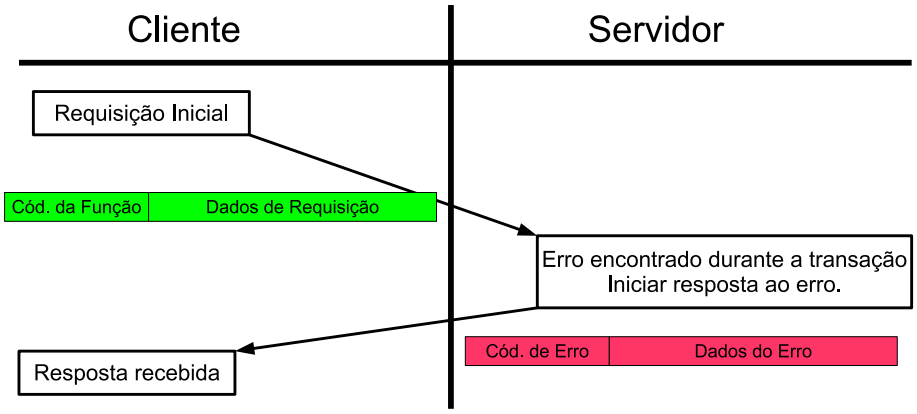

Figura 2.4: Transação Modbus - Ocorrência de exceção, adaptado de [19].

gam em seu interior. São eles:

- *Modbus Request* PDU(*mb_req_pdu*) - Possui os seguintes campos:
  - *function_code*: É o código da função Modbus requisitada pelo Mestre e ocupa um byte;
  - *request_data*: Este campo contém os dados que se deseja transmitir e pode ter de zero a 252 bytes. Este campo é dependente do campo *function_code* e usualmente contém informações como referências a variáveis, número de bytes, *offset* de dados, código de sub-funções etc.
- *Modbus Response* PDU(*mb_rsp_pdu*) - Possui os seguintes campos:
  - *function_code*: É o código da função Modbus retornada pelo Escravo e ocupa um byte;
  - request_data: Este campo é similar àquele anteriormente descrito.
- *Modbus Exception Response* PDU(*mb_excep_rsp_pdu*) - Possui os seguintes campos:
  - *exception_function_code*: É o código da função Modbus acrescido do valor 0x80. Este campo ocupa um byte;
  - *request_data*: Retorna os dados necessários para se identificar em qual situação a exceção se encontra. O código Modbus de exceção é definido na tabela *"MODBUS Exception Codes"* e está disponível na referência [19].

2.2.2 Funções e Representações de dados em Modbus

As funções definidas no protocolo Modbus podem ser classificados em três categorias:

- Funções Públicas:São as funções pré-definidas na padronização do protocolo MODBUS. Estas funções ocupam os intervalos de 01 a 65, de 72 a 100 e de 110 a 127.
- Funções Reservadas: Estão alocadas em meio as funções públicas e têm os endereços: 8, 9, 10, 13, 14, 41, 42, 90, 91, 125, 126 e 127.
- Funções Genéricas: São as funções definidas pelo usuário, e que não tem padronização na norma. Elas compreendem todos os outros endereços não mencionados anteriormente, exceto o "0", que é um código de função inválido.

Para o campo de dados da PDU Modbus é definida a representação *Big-Endian*. Isto significa que quando um determinado número, contido no campo de dados a ser transmitido, excede o tamanho de 1 byte, o byte mais signficativo é enviado primeiro. Assim se um determinado valor possui 16 bits e tem o valor de `0xABCD`, o primeiro byte enviado deve ser `0xAB` e só então `0xCD`.

A próxima seção descreve as particularidades do Modbus-RTU e na sequência sua implementado em meio físico RS-485.

2.3 O Modbus-RTU

A especificação do Modbus-RTU possui particularidades que o diferencia dos demais modos Modbus de transmissão. Nesta seção serão apresentadas as características desta rede e também os procedimentos necessários para correta implementação do mesmo em meio físico RS-485 de maneira a tornar a comunicação eficiente e com minimização de erros.

2.3.1 Comunicação Mestre/Escravo em Modbus-RTU

O protocolo de Linha Serial Modbus-RTU é do tipo Mestre Escravo[20], onde somente um Mestre por vez pode ser conectado no barramento, enquanto que até 247 nós Escravos podem ser ligados a este barramento. A comunicação Modbus-RTU é sempre

iniciada pelo Mestre, enquanto que os nós Escravos nunca transmitirão dados sem receber uma solicitação do nó Mestre. Devido a esta regra, os nós Escravos também nunca se comunicam um com o outro. Ao Mestre é permitido iniciar somente uma transação por vez, seja ela em qualquer um dos modos de transmissão.

Há dois tipos de transmissão. No modo *unicast* o Mestre endereça um nó Escravo e após o envio da requisição aguarda por um determinado tempo a resposta. No modo *broadcast* a requisição é enviada a todos os Escravos simultaneamente e nenhuma resposta é esperada. O endereço disponível para os escravos é de 1 a 247 sendo o endereço zero alocado para o *broadcast* e os endereços de 248 a 255 reservados para usos futuros, como pode ser visto na especificação em [20].

2.3.2 Modo de Transmissão RTU

No modo RTU(*Remote Terminal Unit*), cada byte da mensagem contém dois caracteres de 4-bits hexadecimal. A principal vantagem deste modo é que a grande densidade de caracteres permite um melhor *throughput* do que o modo ASC-II para a mesma taxa de transmissão. As mensagens são transmitidas em um fluxo contínuo de caracteres para assegurar uma correta transação, e como será visto a frente, o intervalo entre bytes para que a mensagem seja considerada incompleta ou perdida é determinado de acordo com a velocidade de transmissão escolhida. O formato define que para cada 8 bits de informação RTU, são utilizados 3 bits adicionais para formalizar a comunicação serial. Deste modo, cada byte RTU conterá em suma 11 bits:

- 1 start bit;
- 8 bits de dados (menos significativo enviado primeiro);
- 1 bit de paridade;
- 1 stop bit.

Isto garante um aproveitamento de 72% do quadro completo para transmissão da informação útil, sendo os outros 28% bytes de controle. Para efeito de comparação, este aproveitamento gira em torno de 40% para o modo de transmissão Modbus ASC-II. A paridade par é utilizada como padrão, mas outras variações como paridade ímpar e sem

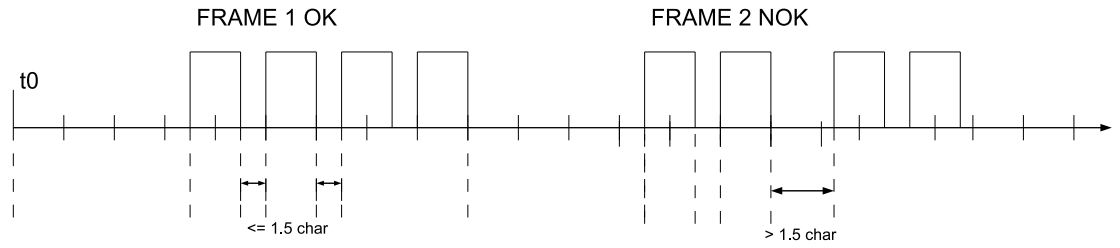

Figura 2.5: Exemplo de Intervalo entre bytes.

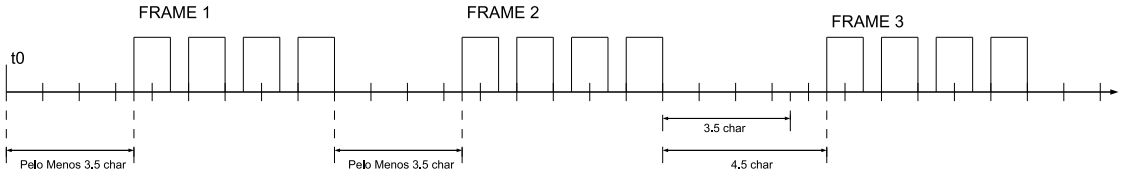

Figura 2.6: Exemplo de Intervalo entre quadros.

paridade podem ser implementadas. No envio sem paridade são necessários 2 stop-bits, de modo que não há mudança na relação de bits enviados por bits úteis.

2.4 O Quadro de Mensagem Modbus-RTU

Uma mensagem Modbus-RTU é colocada pelo dispositivo transmissor em um quadro que tem seu início e fim conhecidos. Isto permite aos dispositivos que recebem um novo quadro descobrir quando uma mensagem começa e quando ela é completada. No modo RTU, os quadros de mensagem são separados por um silêncio de pelo menos 3,5 caracteres de tempo. Porém, para evitar carregamento desnecessário da CPU, estes valores devem ser respeitados somente para transmissões de baixa velocidade, de até 19200bps. Para valores maiores é recomendado pelo protocolo que se utilize valores fixos de 750μs para tempo entre caracteres($t_{1.5}$) e 1,750ms ($t_{3.5}$) para tempo entre mensagens. A Figura 2.5 contém uma sequência de quadros transmitidos onde um deles não seguiu a regra tempo entre bytes e por isto foi descartado. Já a Figura 2.6 mostra três quadros de mensagem que respeitam o tempo entre quadros definidos na especificação.

2.4.1 Detecção de Erro

No modo RTU a detecção de erro é baseada na detecção do Erro de Redundância Cíclica CRC (*Cyclical Redundancy Checking*)[5]. O CRC checa o conteúdo da mensagem inteira para constatar se houve erro no envio. O campo do CRC na ADU ocupa 16 bits e após o seu cálculo o mesmo é adicionado ao fim do quadro. Em caso de erro de CRC quando da leitura do quadro, uma mensagem indicando o erro é retornada ao dispositivo Mestre de modo que, ao receber esta mensagem, seja possibilitado a ele reenviar a mensagem possibilitando assim uma comunicação com entrega da mensagem garantida.

2.5 A Camada Física RS-485 para Modbus

A implementação física do Modbus em RS-485 faz-se utilizando uma série de normas e padrões estabelecidos tanto pela EIA/TIA/RS-485 quanto pelos padrões exigidos numa comunicação Modbus para alcançar os requisitos necessários e relevantes para um bom funcionamento da rede como especificado em [20]. Assim, como todo sistema baseado em barramento, há diversas maneiras de se ligar um dispositivo a rede. De acordo com a norma EIA/TIA/RS-485 o modelo 485 pode ser implementado em 4 configurações diferentes sendo elas *Daisy chain*, Árvore, Estrela e *Branch* mostradas na Figura 2.7.

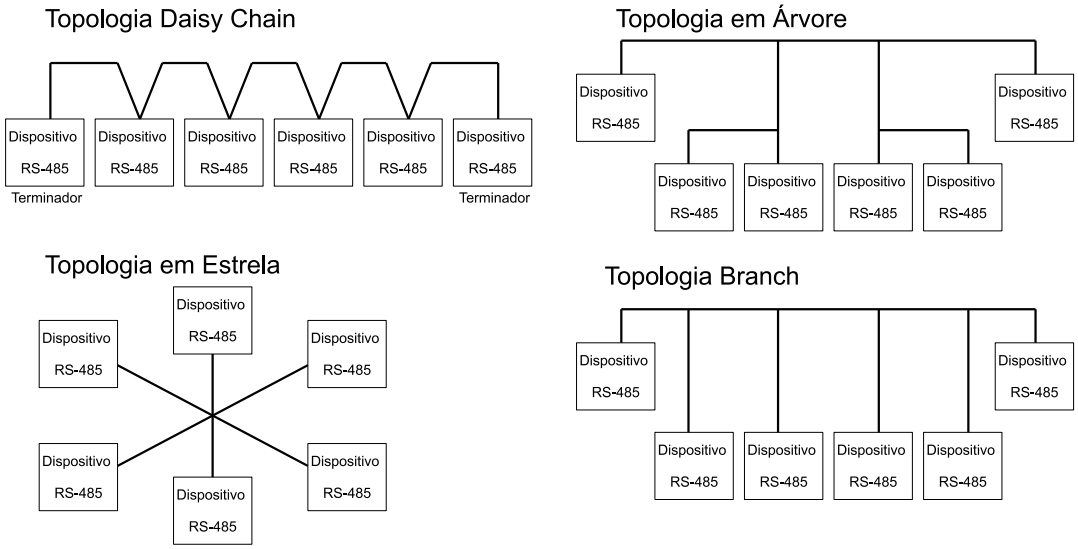

Figura 2.7: Topologia Modbus.

A topologia *Daisy Chain* é a configuração fortemente recomendada pela especifi-

cação [20] dado que nessa configuração efeitos de reflexão, isolamento de terra e interferências diversas são minimizados. Todas as outras três possibilidades são desaconselhadas por conter sérias limitações quando o número de dispositivos adicionados à rede é incrementado. Portanto, neste trabalho, é abordado unicamente o modo de construção *Daisy Chain*.

A comunicação é padrão em 19,2Kbps, porém o protocolo permite que a comunicação se dê em velocidades que variam de 9,6Kbps a 10Mbps, ficando esta escolha a cargo do usuário. Uma vez que se desejou alta velocidade para comunicação de dados na rede de instrumentação concebida, optou-se por utilizar a taxa de transmissão a 115,2Kbps, dado que esta era a maior velocidade disponível no microcontrolador PIC utilizado.

2.5.1 Interfaces Elétricas

A ligação dos dispositivos ao barramento dentro da configuração *Daisy Chain* pode ser feito de três maneiras possíveis sendo elas:

- Uso de derivação ativa (*active tap*)- Contêm em si o transceiver de recepção em modo RS-485;
- Uso de derivações passivas (*passive tap*) - O transceiver é implementado no nó;
- Ligação direta no fio principal (*trunk*) - Os fios são ligados diretamente ao nó.

A Figura 2.8 mostra uma ligação Modbus/485 utilizando derivação ativa (AUI), derivação passiva (IUD) e truncamento (ITr).

Por questões de conveniência e visando o melhor aproveitamento do nó utilizado para construção da rede deste trabalho, deu-se preferência pela utilização da ligação direta no fio principal.

As altas velocidade e grandes distâncias possíveis de serem alcançadas com a especificação EIA/TIA-485 se deve principalmente ao modo de transmissão diferencial adotado para tráfego dos sinais seriais, no qual o valor lógico de um bit é determinado com base na diferença entre os sinais presentes nos dois fios de transmissão de dados, denominados D0 e D1. Para a ligação dos Mestres e Escravos na rede existem duas configurações possíveis no sistema Modbus/485 sendo elas:

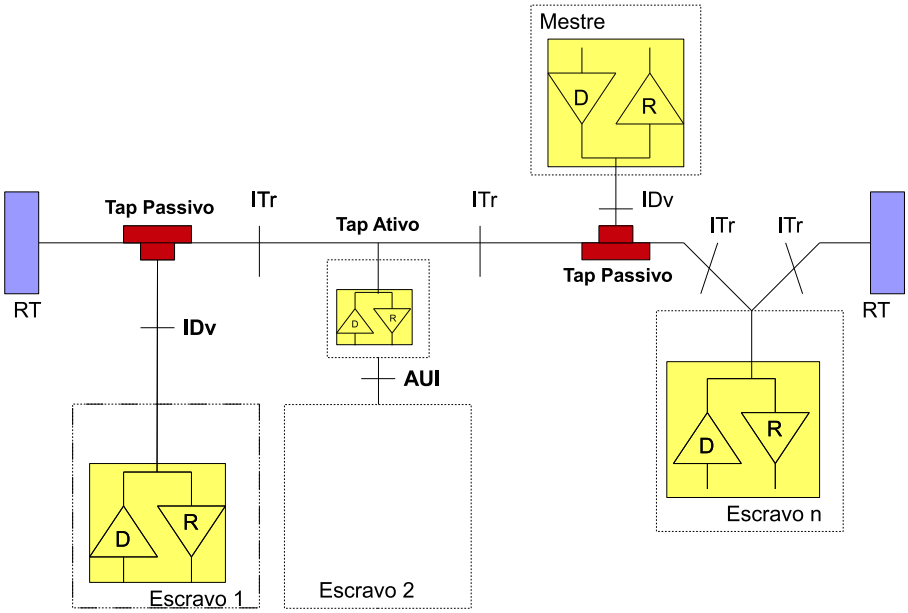

Figura 2.8: Tipos de Conexões possíveis em Modbus/RS485.

- Ligação RS-485 a 2 fios - O mesmo cabo é utilizado pelo Mestre e pelos Escravos para envio e recebimento dos dados sendo necessário controle de fluxo local para determinar a direção do dado - *half-duplex* conforme a Figura 2.9.

- Ligação RS-485 a 4 fios - Dedica uma ligação para envio de dados ao mestre e outra para recebimento de dados pelos Escravos permitindo comunicação simultaneamente para os dois lados - *full-duplex* como pode ser visto na Figura 2.10

Analisando-se as possibilidades, devido a simplicidade buscada na implementação deste projeto, foi escolhida a ligação a dois fios para ser trabalhada. Esta solução implica no uso de sinais para controlar o fluxo de dados tanto no mestre quanto nos escravos. Este controle é local e o barramento a dois fios permite que somente o Mestre ou um Escravo utilizar a rede para enviar dados. Deste modo, é necessário configurar todos os Escravos como receptores da rede, colocando-o em modo de transmissor somente diante de uma solicitação do Mestre. Do mesmo modo, o mestre deve ser sempre transmissor, passando a ser receptor somente por um curto intervalo de tempo, que será utilizado pelo Escravo endereçado para responder a solicitação. Se dois nós quaisquer se tornarem transmissores simultaneamente nenhum dado trafegará pela rede.

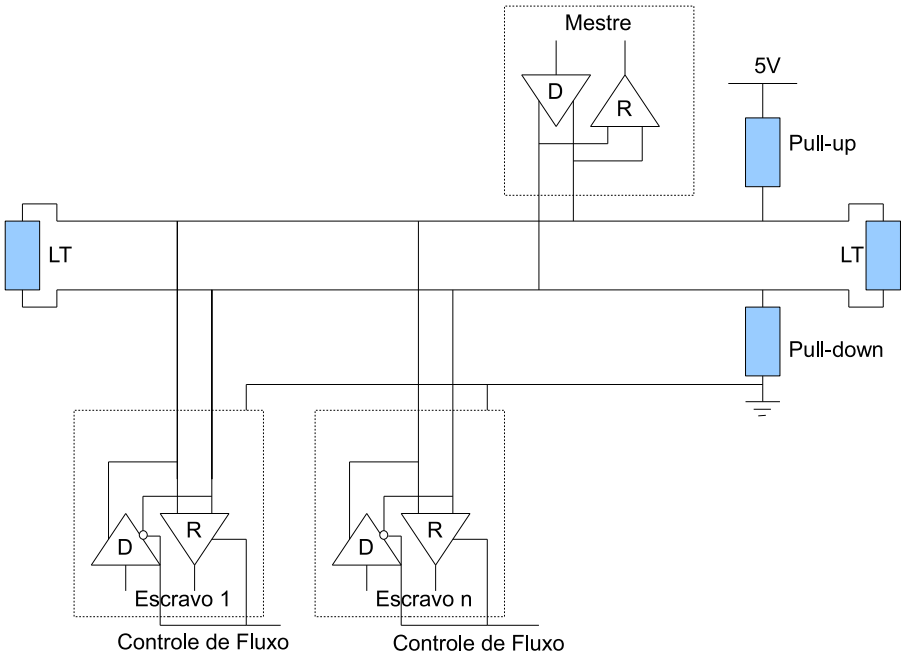

Figura 2.9: Configuração a dois fios com Controle de Fluxo.

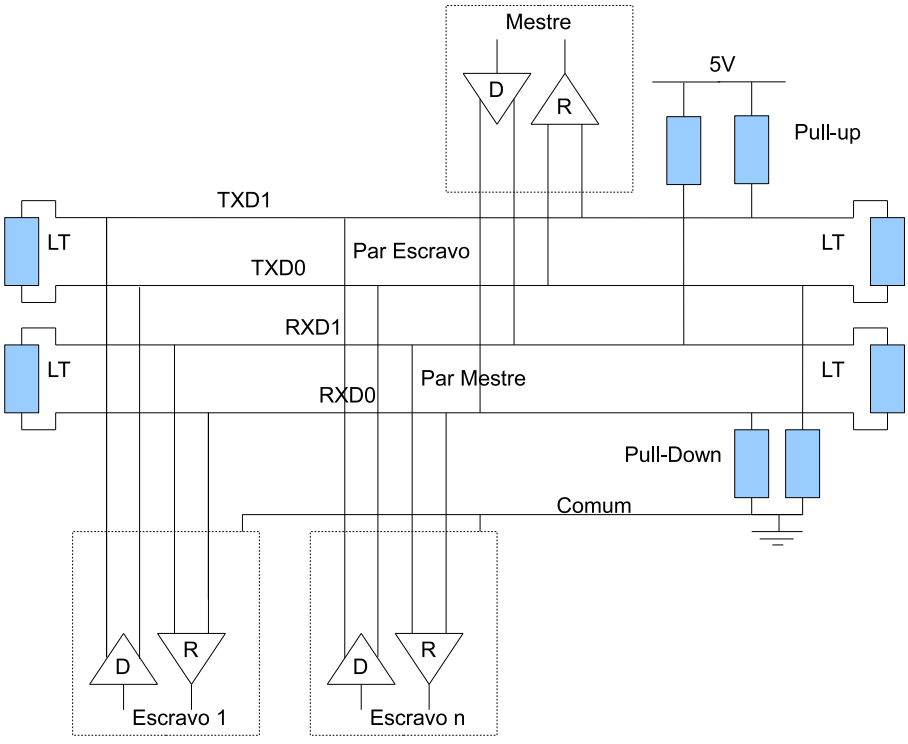

Figura 2.10: Configuração a quatro fios.

2.5.2 Requisitos da Ligação Multiponto em EIA/RS-485

A lista abaixo dispõe várias regras que devem ser observados na construção do sistema de barramentos multiponto 485/modbus. A supressão de algum desses itens pode contribuir para um mal funcionamento do barramento ou inclusive danificar o mesmo.

- 32 escravos são permitidos em qualquer sistema RS-485/Modbus sem repetidores, mas dependendo das características da rede este número pode ser maior;
- É necessário terminadores no barramento RS485/Modbus;
- O tamanho máximo do cabo varia com a capacitância da rede, do tipo de ligação, da velocidade de transmissão, do tipo de cabo, do nível de ruído e de outros fatores. O tamanho máximo é de 1200m para comunicação a 9,6Kbps e este valor diminui a proporção que a taxa de transmissão é aumentada. O gráfico da Figura 2.11 mostra genericamente o comportamento desta rede.
- As derivações não devem ser maiores que 20 metros.
- Preferencialmente todo o sistema deve estar ligado a um ponto de terra único;
- Os terminadores podem variar de 120ohms/0.5W a 150ohms/0.5W no final de cada um dos barramentos;
- Quando se deseja cabos polarizados, um capacitor de 1nF 10V em serie com um resistor de 120Ω/0.25W é altamente recomendado. Cabos polarizados são recomendados na utilização do Modbus/RS485;
- Em caso de polarização dos fios, para reduzir o ruído quando não houver transmissão de dados, faz-se necessário a introdução de resistores de pull-up e pull-down nos cabos diferenciais de transmissão D1 e D0. O valor destes resistores devem estar entre 450 e 650 ohms.

2.5.3 Conectores

Os conectores para conexão dos Mestres e Escravos no barramento podem ser do tipo DB9 (Fig. 2.12) ou RJ45 (Fig. 2.13) tanto para ligação a dois fios quanto para quatro fios.

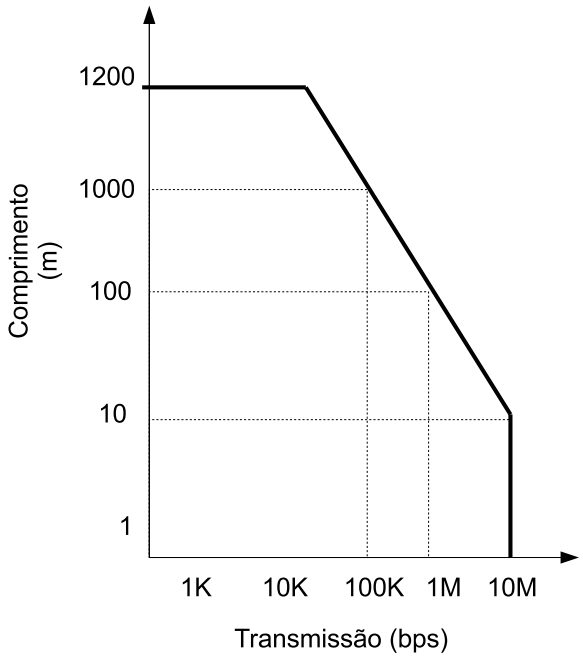

Figura 2.11: Distância vs Velocidade de Transmissão. Adaptado de [15].

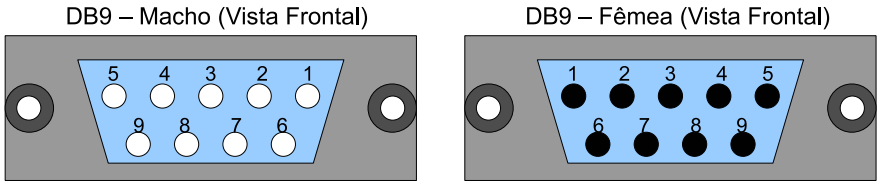

Figura 2.12: Conector DB9.

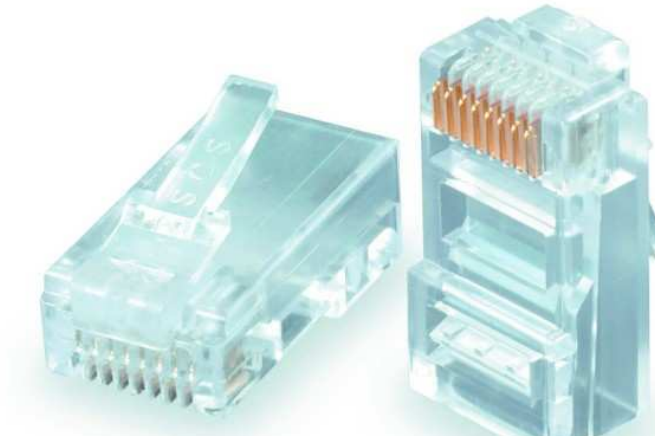

Figura 2.13: Conector RJ45 Macho.

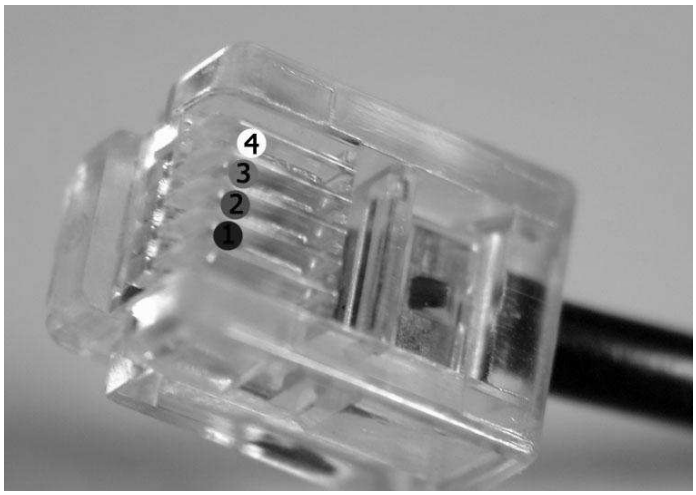

Figura 2.14: Conector RJ11 Macho.

Para o caso de RJ45 ou D-Shell em dispositivos MODBUS é necessário que o conector fêmea seja blindado para maior rejeição a ruído. Neste trabalho, no entanto, o conector utilizado foi o RJ11 (Figura 2.14) devido ao tamanho físico do mesmo em relação ao circuito em que foi colocado.

2.5.4 Cabos

A conexão RS485/Modbus requer dois fios equilibrados para o par D0-D1 e um terceiro fio comum para aterramento do sistema. Também foi utilizado um quarto fio para transmissão da alimentação evitando o inconveniente da alimentação individual nos

nós.

O cabo é espesso o suficiente para suportar o comprimento máximo de 1200m permitido pelo RS-485. O tipo AWG24 [8] diâmetro de 0,51mm secção de 0,21mm^2, resistência de 0,084ohm/m e corrente máxima de 4A é a opção recomendada para este objetivo.

2.5.5 Diagnóstico Visual

Para observação da comunicação e detecção de possíveis irregularidades faz-se necessário o uso de LEDs indicadores como mostrado na Tabela 2.1.

Tabela 2.1: Tabela Indicadora de Leds

LED	**Padrão**	**Estado**	**Cor Recomendada**
Comunicação	Necessário	Aceso durante recepção ou transmissão.	Amarelo
Erro	Recomendado	Aceso: Falha Interna Piscando: Outros Erros	Vermelho
Estado do Dispositivo	Opcional	Aceso: Dispositivo Ligado	Verde

2.5.6 Considerações Sobre a Implementação Física do Modbus

Todos os componentes presentes na rede devem seguir uma série de requisitos necessários para o bom funcionamento do sistema. A Tabela 2.2 mostra um resumo destes requerimentos, juntamente com a solução adotada neste trabalho para implementação do mesmo.

2.6 Conclusão do Capítulo

A rede Modbus se mostrou adequada para a proposta deste projeto. Na comunicação Mestre/Escravo não há colisão de dados pois o mestre pode fazer somente uma requisição por vez e os escravos são passivos, somente respondendo às solicitações do Mestre. A rede multiponto é possibilitada pelo padrão EIA/RS-485 que permite um máximo de

Tabela 2.2: Tabela de Requisitos Modbus/RS485

Endereçamento	**Mínimo Recomendado** Escravo: Endereços configuráveis de 1 a 247	 Mestre: Capacidade de endereçar escravos de 1 a 247	**Adotado** Mesmo que o Básico
Broadcast	Sim		Sim
Paridade	Par		Par
Modo	RTU		RTU
Baud Rate	9.6Kbps		115.2Kbps
Interface Elétrica	RS485 2-Fios ou RS 232		RS485 2-fios
Tipo de Conector	RJ-45, RJ-11 ou DB9		RJ-11
Cabo	Cabo Par Trançado Blindado 4 pares		Par Trançado UTP 24 AWG 2 pares

32 nós sem o uso de repetidores. A velocidade escolhida de 115.2Kbps é apta para abranger as funcionalidades dos dispositivos embarcados. O conector RJ-11 visa ocupar um mínimo de espaço físico no nó escravo, e o cabo escolhido, UTP 24AWG - 2 pares, além de ser mais leve que o proposto na especificação permite maior flexibilidade na alocação e posicionamento do mesmo no veículo.

Para o Capítulo 3, com base nos dados obtidos a respeito da configuração Modbus/485 foi traçada uma estratégia para sua implementação. Uma vez efetuado este passo, foram realizados testes em bancada e em campo.

Capítulo 3

Desenvolvimento da Rede

3.1 Introdução

A partir dos dados constantes no Capítulo 2 foram levantados os requisitos necessários para a implementação do Modbus. Dentre os recursos de hardware necessários pode-se citar:

- Placa-protótipo já existente no laboratório que contém um Microcontrolador PIC modelo 18F2550 conforme consta na folha de dados[18], um Regulador de Tensão para alimentação e comunicação USB para programação *in-circuit* do PIC e também como interface de entrada e saída dos dados;
- Um Transceiver RS-232 e um Transceiver RS-485 com os quais será montado um conversor RS232/RS485 para ligação do Mestre na rede;
- Transceivers RS-485 para montagem nos nós;
- Cabo de par trançado UTP AWG24/2 Pares para aplicação a que se destina este trabalho.

Além dos detalhes de implementação física, a construção de uma rede envolve o estudo de algumas características que medirão a capacidade da mesma e sua viabilidade. Uma rede de tempo real necessita, além disso, de um estudo dos intervalos de confiabilidade, da banda necessária e disponível, do *throughput*, da escalabilidade do sistema e das distâncias máximas permitidas. Com relação ao software para o Mestre, a linguagem de programação foi o C++. A vantagem desta linguagem é sua universalidade para uso tanto em Linux quanto Windows.

3.2 Modbus Implementado em PC/PC via RS-232

O primeiro passo dado no sentido de elaborar a rede foi a sua implementação em PC. Utilizando-se 2 PC's Pentium 4 foi programada uma aplicação em C++ para comunicação MODBUS Mestre/Escravo entre os dois.

O Mestre, como mostrado anteriormente, pode ser implementado em diferentes sistemas operacionais, porém é também objetivo deste trabalho tornar os programas utilizados os mais paramétricos possíveis de modo que ao menos as interfaces do programa principal sejam universais e independentes do sistema operacional. As subrotinas para configuração, controle, envio e recepção dos bytes são:

- int Open(int nPort = 2, int nBaud = 115200) - Responsável por abrir a porta serial indicada por 'nPort' e configurá-la com velocidade de transmissão 'nBaud'. Caso não seja repassado nenhum parâmetro para esta função, a mesma definirá os valores da porta e da taxa de transmissão em 2 e 115.2Kbps respectivamente.
- int Close(void) - Comando para fechar a porta serial.
- int ReadData(void $xread^*$, int n) - Lê n bytes recebidos via serial e os guarda no arranjo $xread^*$.
- int SendData(const char $xsend^*$, int n) - Envia os n primeiros caracteres do vetor apontado por $xsend^*$.
- int ReadDataWaiting(void) - Subrotina sem valor de retorno, que aguarda a recepção do primeiro byte provindo do escravo após uma solicitação.

A configuração escolhida para os testes de bancada da comunicação Modbus foi:

- *Baudrate* = 115.2Kbps;
- Paridade = Par;
- Detecção de erro CRC16, conforme a especificação;
- Porta de Comunicação disponível no PC.

Apesar da interface semelhante, os códigos utilizados para acesso e controle da porta serial são significativamente diferentes em Linux e Windows devido as bibliotecas utilizadas, chamadas de funções e diretivas presentes em cada código.

Frame Modbus RTU

Start Frame	Endereço	Função	Dados	CRC	Stop Frame
>= 3.5 chars	8 bits	8 bits	N x 8 bits	16 bits	>= 3.5 chars

Figura 3.1: Quadro de Mensagem Modbus/RTU.

A programação do mestre foi feita primeiramente no Windows e vários testes utilizando programas como o RComSerial disponível em [17]. Ao final, o mestre deveria ser capaz de enviar dados e recebê-los simultaneamente na mesma porta através da conexão entre os pinos RX e TX da porta serial.

O quadro de mensagem a ser enviado é o Modbus RTU mostrado na Figura 3.1, e, como indicado no capítulo anterior, a ADU do quadro de mensagens neste modo é composto pelos seguintes campos:

- *Start Frame* - Silêncio de 3 a 5 bits que indica que uma nova mensagem irá se iniciar.
- Endereço - Campo de 8 bits que contém o endereço do dispositivo para o qual a mensagem se destina. São atribuídos valores de 1 a 247 para endereços individuais. O valor 0 é reservado para *broadcast*.
- Função - Contém o valor da função a ser executada pelo dispositivo. Podem ser atribuídos valores de 1 a 127 pois as mensagens com o bit mais significativo em ”1” indicam exceção na execução do comando.
- Dados - Tamanho e conteúdo do campo de dados variam com a função e o papel da mensagem, requisição ou resposta, podendo mesmo ser um campo vazio. Seu valor é de Nx8 bits.
- CRC - Campo de 16 bits para verificação de erro no envio da mensagem.
- *Stop Frame* - Silêncio de 3 a 5 bytes para indicar que a transmissão da mensagem foi finalizada.

O CRC foi calculado através de um algoritmo de CRC16/Modbus obtido em [25]. Um dos PCs foi colocado como mestre e o outro como escravo para teste de recepção e

transmissão de dados. O algoritmo utilizado em ambos os casos é semelhante e pode ser observado na Figura 3.2. A sequência de passos pode ser descrita como sendo:

Mestre

- Abre Porta Serial;
- Configura Porta Serial;
- Entra em Loop;
- Envia Comando ao Escravo;
- Recebe Resposta do Escravo;

Escravo

- Abre Porta Serial;
- Configura Porta Serial;
- Entra em Loop;
- Recebe Comando do Mestre;
- Executa a Função;
- Envia Resposta ao Mestre;

3.3 Modbus Implementado em PC/PIC RS232

Uma segunda parte dos testes envolveu a comunicação entre o PC e nó composto por um PIC18F2550 no qual uma mensagem proveniente do mestre/PC seria enviada ao escravo/nó microcontrolado e o mesmo decodificando a mensagem enviada responderia com um novo quadro Modbus a ser projetado no console do PC.

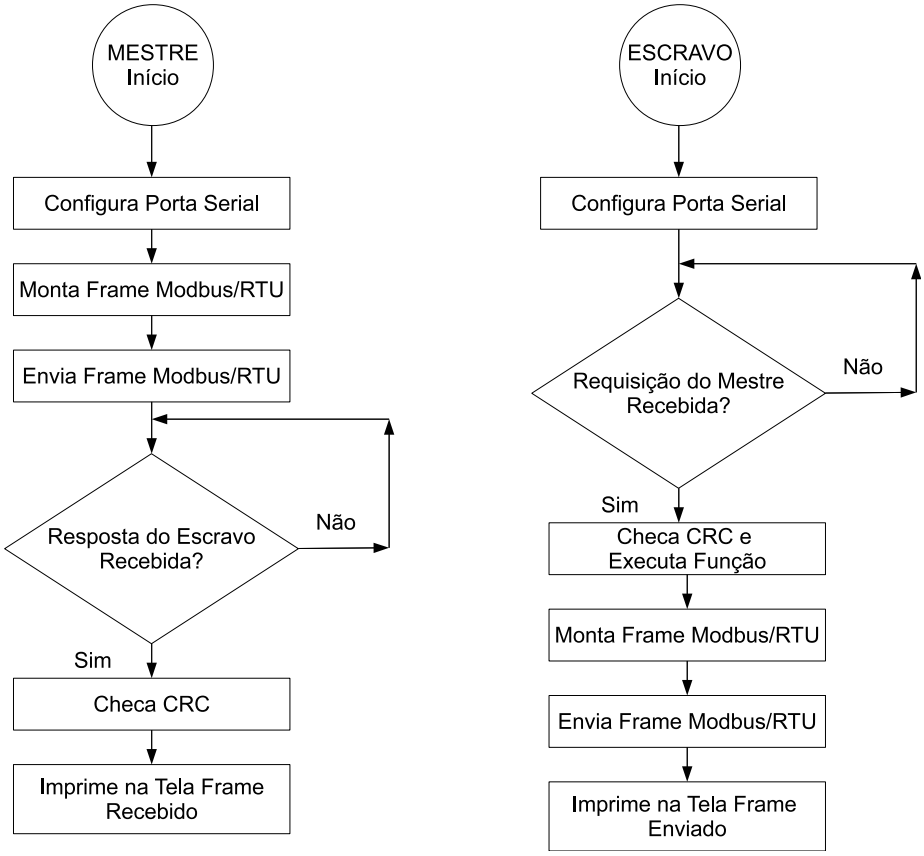

Figura 3.2: Algoritimo de envio e recepção de mensagens no Mestre e no Escravo.

3.3.1 Hardware Utilizado

A montagem do nó utilizou uma placa pré-fabricada disponível no laboratório CORO na qual o PIC 18F2550 é programado através de uma porta USB por meio de um programa *bootloader* e todas as portas do PIC estão disponíveis para ser utilizadas através de uma placa de extensão projetada para este uso. O nó contendo o PIC pode ser visto na Figura 3.3.

3.3.2 Conversão de Sinais

A conversão de sinais entre o PIC e o RS-232 proveniente do computador exigiu um *tranceiver* denominado MAX-232 [32] que é um conversor de tensão dada a incompatibilidade entre as interfaces do PIC TTL de 0 a 5V e os pinos de comunicação do RS-232 de -10V a +10V.

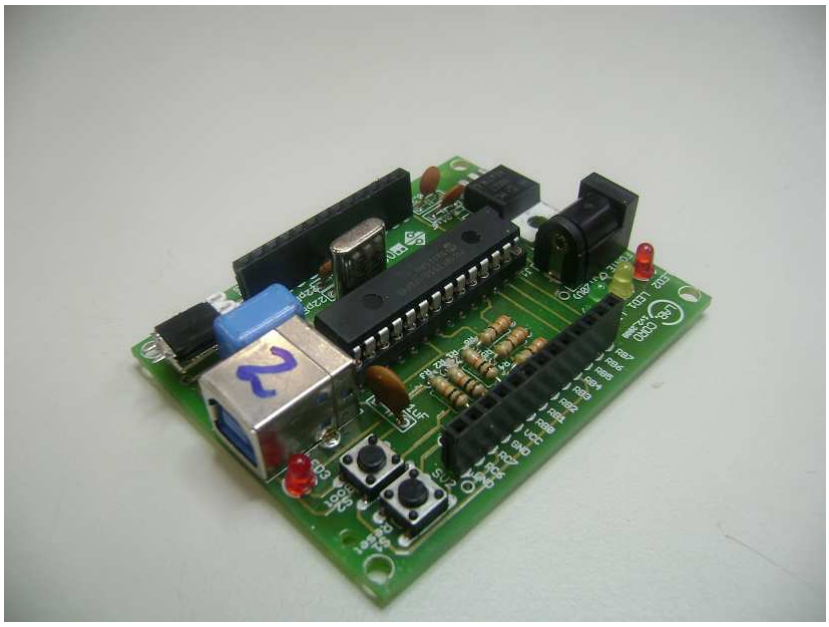

Figura 3.3: Placa de Desenvolvimento com PIC18F2550.

3.3.3 Cabos e Conectores

Os cabos utilizados são os recomendados na especificação [19] sendo o AWG 24 par trançado utilizado de acordo com o especificado anteriormente, em conjunto com o conector RJ11 de quatro vias. Porem não há blindagem nestes cabos e conectores. A Figura 3.4 mostra as cores do cabo e os respectivos sinais utilizados no mesmo.

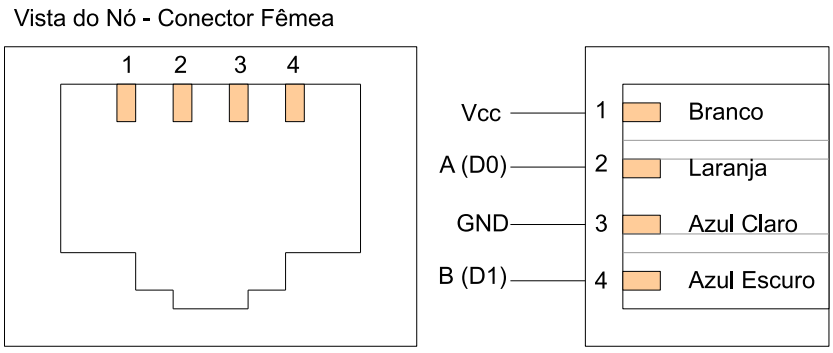

Figura 3.4: Configuração do conector/cabo utilizado neste trabalho.

3.3.4 Software

A programação do PIC foi feita em C utilizando-se o ambiente de desenvolvimento MPLAB da MICROCHIP. O software do PIC para este trabalho foi desenvolvido de modo a ser computacionalmente leve uma vez que a comunicação via Modbus é considerada como função secundária, sendo a função primária definida a posteriori.

3.4 Modbus Implementado em PC/PIC RS-485

A terceira fase de testes consistiu em implementar um PC como mestre e um nó como escravo e obter a comunicação entre estes dois elementos porém utilizando a camada física proposta anteriormente EIA RS-485.

3.4.1 Conversor RS232/RS485

Para viabilizar os testes o primeiro elemento construído foi um conversor RS-232/RS-485 partindo-se do pré-suposto que a comunicação será a 2 fios como anteriormente determinado. Esta conversão exigiu o uso de um *tranceiver* MAX232 para realizar a conversão RS-232/TTL e um outro *tranceiver* MAX485 para converter TTL/RS-485. Uma vez que a comunicação RS-485 escolhida é *half-duplex* faz-se necessário o uso de um sinal extra para controlar o fluxo em cada um dos nós, de modo a deixar os nós escravos sempre como receptores e o nó mestre como transmissor, deste modo estabelecendo uma sincronia no barramento e possibilitando a comunicação. Para o Mestre, o sinal utilizado para controle de fluxo foi o RTS (*Request To Send*) disponível em um dos pinos da porta serial. Este sinal é colocado em baixo (GND) quando o Mestre vai transmitir dados e colocado em alto(Vcc) quando o mesmo irá receber dados de algum dos Escravos. Este sinal é convertido por um MAX232 e ligado nas portas 6 e 7 do CI MAX485, conforme descrito no *datasheet* do mesmo.

Analogamente no nó escravo também foi utilizado o pino RB3 do PIC para comandar o fluxo de dados no *tranceiver* MAX485. Ao contrário do Mestre, todos os Escravos são colocados em modo de recepção por todo o tempo exceto no momento de enviar os dados solicitados pelo Mestre no qual o mesmo passa a ser o transmissor.

3.5 Conclusão do Capítulo

Uma vez concluída a fase de testes iniciais partiu-se para as investigações a respeito da comunicação estabelecida para determinar os melhores caminhos para se ter uma comunicação de alta qualidade e que atenda os requisitos além de otimizar os pontos em que isto for possível. Assim tanto o hardware quanto o software foram analisados de modo a fornecer substrato para as escolhas.

Capítulo 4

Experimentos

Com as informações recolhidas, como mostrado nos capítulos anteriores, um protótipo foi montado utilizando um nó mestre e dois nós escravos. Utilizando-se de softwares já existentes e em uso no carro autônomo, foram realizadas alterações nos mesmos de modo a incluir a rede desenvolvida no sistema. Uma vez modificados, verificou-se o desempenho dos mesmos em relação ao original e as mudanças observadas. A partir destes dados foi possível concluir a viabilidade da rede ou não para o destino a que se propôs.

4.1 Sensor de Velocidade das Rodas

O projeto realizado em C via MPLAB e denominado “Rodas” é um nó sensor desenvolvido por bolsistas do Laboratório do CORO que visa obter o valor da velocidade em tempo real das rodas do veículo [9]. Este valor é obtido com base nos sinais fornecidos pelos sensores de relutância magnética incluídos no sistema de freio ABS do veículo e em um conversor freqüência/tensão que é lido pelo conversor A/D do PIC. A relação entre o valor lido pelo A/D do PIC e a velocidade de giro das rodas foi obtida por meio de calibração onde uma equação que relaciona a tensão à velocidade angular do veículo foi obtida. Este circuito foi projetado sobre a placa protótipo desenvolvida no CORO. O valor de tensão é lido nos pinos RA1 e RA2 do microcontrolador, sendo o RA1 da roda esquerda e o RA2 da roda direita. O contador Timer#0 é utilizado para para enviar os dados ciclicamente uma freqüência de 50Hz para um PC via USB que fará os cálculos necessários e então enviará o comando de set point, via USB para o nó atuador de freios.

A máquina de estados do nó sensor de velocidade é composta pelos estados:

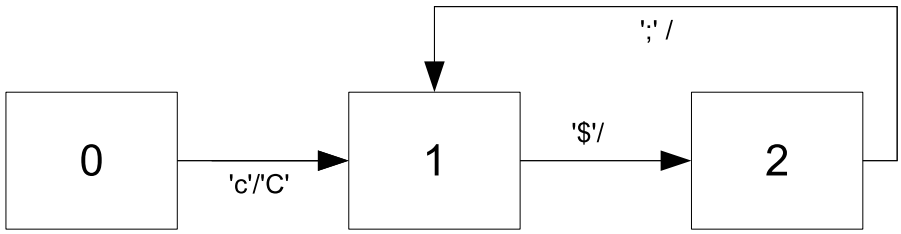

Figura 4.1: Máquina de Estados do nó sensor de velocidade.

- 0 - Aguarda um caracter 'c' via USB para iniciar a conexão;
- 1 - Passa ao estado de espera, aguardando o caracter $ para iniciar o envio.
- 2 - Envia o valor da conversão A/D com uma frequência de 100Hz e fica neste estado até receber um ';' que indica que cessou-se a transmissão, então o sistema volta ao estado 1.

O sistema fica neste loop indefinidamente. Esta máquina de estados pode ser vista na Figura 4.1. O envio é composto de quatro bytes em sequência que significam roda direita e esquerda, MSB e LSB respectivamente. Caso o nó seja desenergizado ele volta ao estado zero. Este programa se encontra atualmente em uso no carro e já tem o seu uso em conjunto com outros programas como o atuador do freio.

4.2 Atuador de Freios

O sistema de atuação de freios[16] foi desenvolvido como trabalho final de curso no CORO e consiste em um controlador de motor de corrente contínua. Uma vez recebido um set-point de posição na entrada, este nó, utilizando-se de um controlador PI, aciona uma ponte H via PWM do PIC e atua de modo a fazer com que o freio chegue na referência o mais rapidamente possível, com determinados limites de oscilação e *overshoot* a serem respeitados.

A máquina de estados que representa o sistema pode ser encarada como mostrado abaixo:

- 0 - O sistema aguarda um caracter 'c' via USB para iniciar a conexão.

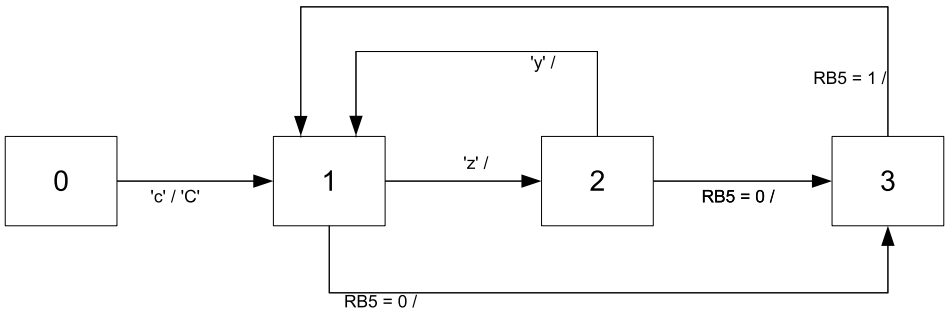

Figura 4.2: Máquina de Estados do nó-atuador de freios.

- 1 - Aguarda um caracter ’z’ para iniciar o controlador. Uma vez iniciado, o controlador iniciará o algoritmo de controle para manter a saída no mesmo nível que a referência.

- 2 - Aguarda um byte compreendido entre 0 e 125 para regular o set-point, o caracter ’y’ para terminar a conexão ou um sinal externo de modo a ir ao estado de emergência(3) do sistema.

- 3 - Neste estado o sistema entra em modo de emergência e o freio é acionado com o máximo de potência e é aguardado novamente o caracter ’z’ para voltar ao estado de controle.

A máquina de estados correspondente a este nó-atuador pode ser vista na Figura 4.2.

4.3 Introdução da Rede Modbus/485

A introdução da rede nos sistemas acima descritos tem como objetivo o mínimo de intervenção no código original, uma vez que a camada de rede é considerada como uma substituta para a comunicação USB atualmente implementada nos mesmos. O algoritmo da rede implementado no nó segue o diagrama da Figura 4.3 que exemplifica todo o procedimento adotado para tratamento dos nós. Sobre este algoritmo é importante levar em consideração os seguites pontos:

- O *query* de *broadcast* enviado pelo Mestre não tem resposta.

- A função de configuração do Modbus não deve interferir em outras configurações realizadas previamente nos nós.

- Outras interrupções tendem a tornar o tempo máximo de transmissão de mensagens maior.

- A função checaEstado() foi criada exclusivamente para tratamento dos algoritmos dos nós sensor e atuador, podendo ser modificada livremente. Dentro de checaEstado é chamada a subrotina frameCompleto() responsável pelo cálculo do CRC.

- A função send() se encontra dentro da interrupção para evitar erro de transmissão de mensagens que poderiam ocorrer caso esta transmissão fosse em espera ocupada. Seus parâmetros de entrada são um arranjo contendo os dados a serem enviados e um inteiro com o número de bytes. Dentro desta função o *query* completo é montado e o cálculo do CRC realizado para envio completo da resposta.

- Todas as vezes em que o Timer#3 é iniciado, a recepção de bytes pela serial é desabilitada, enquanto o intervalo de tempo entre dois bytes for menor que o tempo mínimo entre mensagens, como indicado no Capítulo 2.

- Todas as vezes em que as variáveis de leitura de recepção de um frame Modbus são levadas ao estado inicial, o primeiro byte recebido pela interrupção é sempre considerado como sendo direcionado ao nó que o lê. A partir do segundo byte a leitura é desprezada caso o endereço enviado pelo Mestre não corresponda ao endereço do Escravo.

- Para este algoritmo, todas as mensagens enviadas pelo mestre terão necessáriamente 6 bytes sendo eles:

  - Byte 0 - Endereço do Nó Escravo;
  - Byte 1 - Função (Não utilizado);
  - Byte 2 - Valor do Set-Point (Quando pertinente);
  - Byte 3 - Byte de Controle, responsável por mudar o estado dos Escravos;
  - Byte 4 - Byte mais significativo do CRC;
  - Byte 5 - Byte menos significativo do CRC.

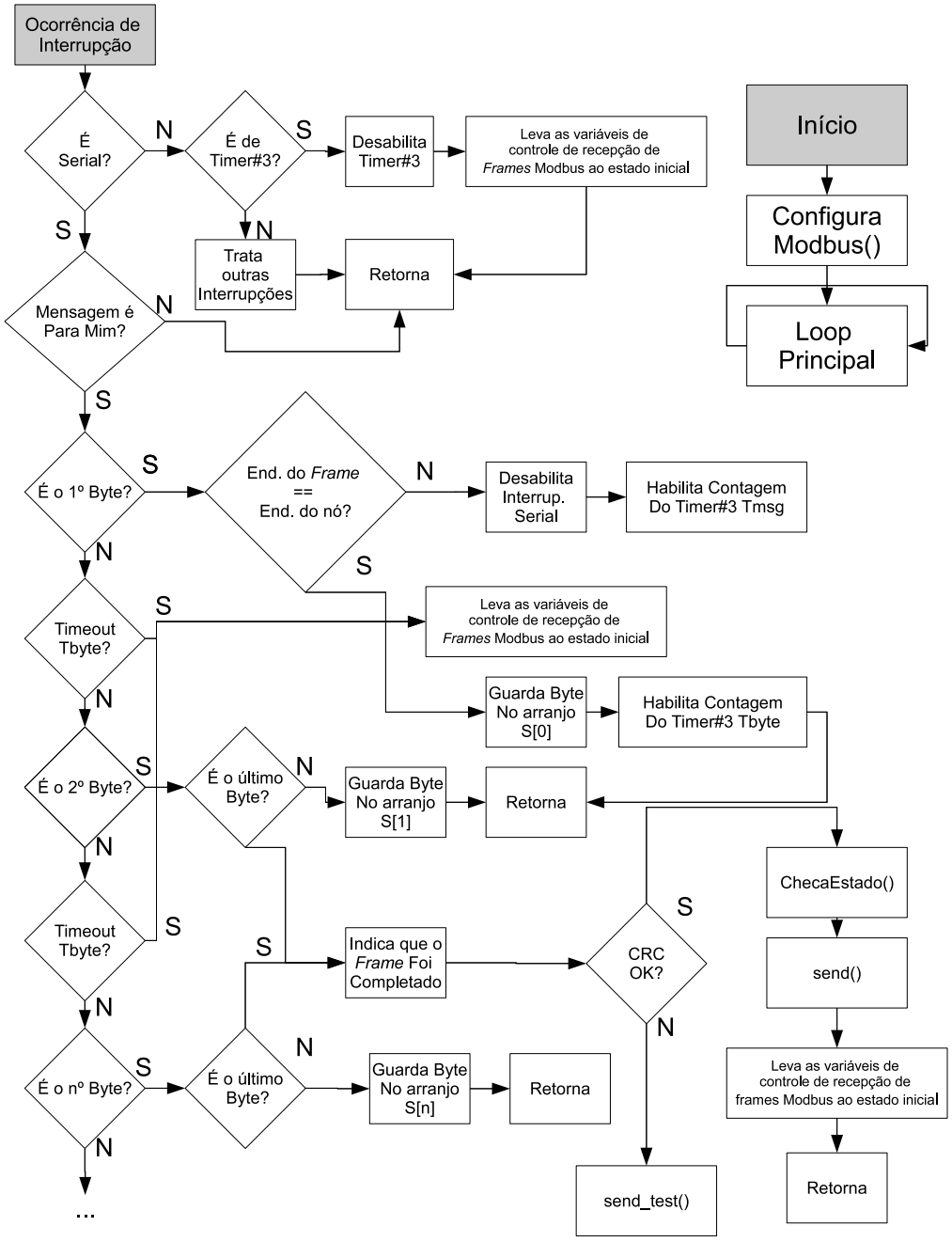

Figura 4.3: Algoritmo para Recepção e Transmissão dos quadros Modbus.

Com base neste algoritmo o procedimento para a mudança nos programas seguiu os seguintes passos:

- Identificação dos pontos onde havia comunicação do nó com o meio externo;
- Retirada das bibliotecas e funções USB de todo o programa;
- Mudança de paradigma no caso do nó sensor onde anteriormente o valor era enviado ciclicamente, para um envio via solicitação do mestre em intervalos que podem ser cíclicos ou aleatórios.
- Mudança em alguns estados visando melhorar a eficiência do programa com a introdução da comunicação Modbus;
- Configuração da porta RB3, do Timer#3 e das interrupções para habilitar a recepção dos quadros Modbus;
- Compilação, gravação e teste do programa para levantar possíveis irregularidades.

Para a construção do protocolo Modbus nos microcontroladores, optou-se por utilizar a interrupção serial para detecção das mensagens, de maneira que ao final da recepção dos seis bytes previstos para compor o quadro nesta fase de testes, fosse checado o CRC dos bytes recebidos. Após esta checagem, caso a recepção tenha sido um sucesso, uma subrotina de mudança de estados é chamada para analisar o quadro recebido e o próximo estado para o qual o sistema tem que ir. Caso não haja erro no quadro e o comando não seja reconhecido pelo nó, uma mensagem com os mesmos bytes recebidos é enviada de volta ao Mestre. Toda esta operação é determinística de modo que é possível calcular quanto tempo se gasta em cada uma destas operações e deste modo determinar o tempo que a comunicação utiliza para realizar estas operações. Isto será visto nas seções subsequentes. O código referente a parte Modbus implementada nos nós Escravos se encontra em Anexo.

Outro ponto importante neste teste foi a determinação da quantidade de memória utilizada pela introdução deste protocolo. Utilizando-se de um recurso disponível no MPLAB, foram obtidas a quantidade de memória utilizada em cada nó antes e após a introdução da rede. A Tabela 4.1 mostra o uso de memória do PIC e as alterações em função da introdução da rede, e na mesma observa-se o relativo ganho de memória

quando da troca do modelo baseado em comunicação USB em relação ao modelo utilizando o protocolo Modbus.

Tabela 4.1: Tabela de Uso de Memória do PIC

Nó	**Mem. RAM(Max 2K)**	**Mem. Flash(Max 32K)**
Rede Modbus	254 bytes	938 bytes
Nó-Sensor de Velocidade USB	565 bytes	3486 bytes
Nó-Sensor de Velocidade Modbus	324 bytes	1819 bytes
Nó-Atuador de Freios USB	566 bytes	6061 bytes
Nó-Atuador de Freios Modbus	370 bytes	4399 bytes

Uma diferença fundamental observada quando da construção dos novos estados é que no modo de comunicação USB, todos os comandos são enviados pelo mestre utilizando-se apenas um byte, enquanto que na comunicação Modbus, todos os comandos são enviados em um quadro que contém, além dos bytes de endereço e função, bytes de dados e correção de erro permitindo-se assim uma gama muito maior de comandos a serem enviados pelo Mestre.

A máquina de estados para o algoritmo de sensor de velocidade sofreu leve alteração. Onde anteriormente o valor de tensão das rodas era enviado a cada 10ms, agora aguarda uma requisição vinda do mestre solicitando esta medida. Quanto a máquina de estados do atuador de freio, a mesma permaneceu inalterada. A diferença, porém, que não pode ser vista através da máquina de estados é a troca da comunicação que passou de USB para Modbus e onde antes eram enviados caracteres de controle, agora são enviados requisições de controle contendo em um dos bytes de dados o mesmo caracter dos algoritmos originais.

Para ser possível a troca de mensagens via RS-485 nos nós escravos, foi necessário a introdução do *transceiver* MAX485 como citado anteriormente. A ligação física deste chip ao PIC é feita ligando-se os pinos 1 e 4 do MAX485 respectivamente nos pinos TX e RX do PIC. O pino RB3 que é configurado como sendo uma saída digital no microcontrolador, deve ser ligada nos pinos 2 e 3 do MAX para controle do fluxo de dados. A alimentação deste chip é feita de acordo com a folha de dados[14]. A Figura 4.4 mostra o esquema de ligação do MAX485 ao PIC18F2250.

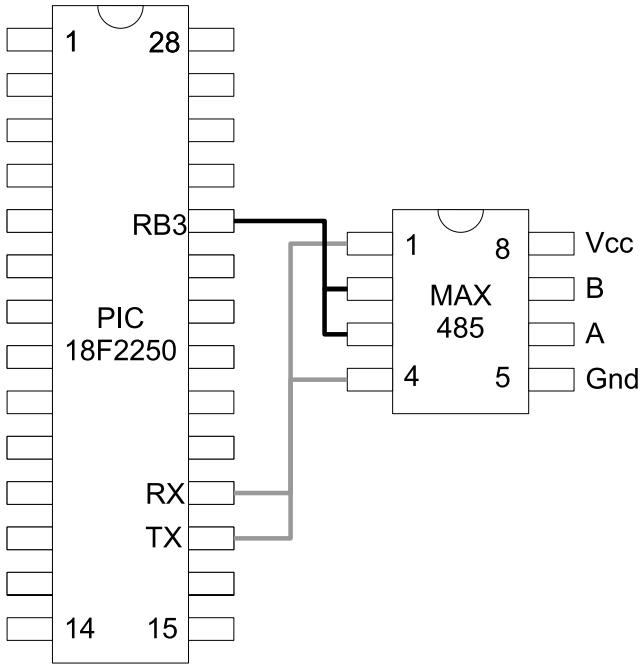

Figura 4.4: Esquema de Ligação entre os pinos do MAX485 e do PIC18F2250

4.4 Concepção do Mestre

Para o tipo de rede a ser construída no modelo Mestre/Escravo, o mestre será um computador portátil PC104 que é uma plataforma portátil compatível com o IBM/PC que possui todas as funcionalidades existentes neste primeiro além de possuir a interface RS-485, que foi utilizada neste trabalho. A Figura 4.5 mostra uma foto do PC-104 utilizado neste projeto. O PC-104 [24] possui todas as características de um computador pessoal excetuando-se pela memória e processamento reduzidos. O modelo deste PC disponível no laboratório é o PFM535i que tem como características principais:

- Processador de 300 Mhz AMD;
- Memória RAM de 256Mb;
- 1x interface EIA/TIA/RS-232;
- 1x interface EIA/TIA/RS-232/RS-422 ou RS-485;
- 2x Entrada USB;
- 1x Entrada para monitor VGA;

Figura 4.5: Computador do tipo PC104 modelo PFM535i

- 1x Entrada para Mouse/Teclado PS2;
- Entrada para Cartão Flash (SSD).

Neste PC foi instalado o sistema operacional Conectiva Linux RTAI, que é uma versão modificada do conectiva Linux.

O sistema operacional é um Linux Conectiva RTAI que permite a funcionalidade de tempo real, caso esta esteja habilitada. Neste trabalho, este recurso não foi utilizado. A construção do algoritmo levou em consideração vários pontos necessários para um funcionamento correto do protocolo Modbus, como se segue abaixo:

- Todas as requisições provém do Mestre que é único para cada rede;
- O Mestre espera uma resposta para cada solicitação enviada. Um *timeout* determina o tempo máximo que o nó tem para responder a determinada solicitação.
- Do ponto de vista do Mestre, o barramento sempre estará em modo de transmissão, exceto pelo tempo de *Timeout* após uma requisição enviada pelo mesmo, onde o barramento estará em modo de recepção e permitirá ao nó escravo endereçado responder.

No PC-104 o controle de fluxo do sinal RS-485 half-duplex é feito pelo sinal RTS que precisa estar ativado ou desativado dependendo da direção dos dados. As mensagens são enviadas a 115,2 Kbps. O *Timeout* é definido por um *loop* que contém em seu interior uma função do tipo *sleep* que indica por quantos milisegundos o sistema ficará em suspenso aguardando a resposta. O número de iterações totais fornece o tempo total de *Timeout* o qual o sistema esta submetido. O recurso de tempo real (RTAI) disponível no Linux, caso utilizado, possibilitará o uso de um Timer preciso sem a necessidade de *loops* que permitirão obter informações de grande relevância sobre os tempos da rede e a possibilidade de seu uso em sistemas de tempo real *soft* e *hard*.

4.5 Teste em Bancada

O teste em bancada visou averiguar a integridade da rede e avaliar a viabilidade da solução para aplicação em um sistema embarcado, mais especificamente no veículo autônomo desenvolvido pelo PDVA/UFMG. O esquema de teste é composto por um mestre, responsável por coletar os dados de velocidade e enviar um *set-point* de corrente para o nó atuador de freio, e dois nós escravos sendo um deles o nó sensor de velocidade das rodas, com endereço de rede 0x43 e o outro o nó atuador de freio com seu sistema de controle do motor de corrente CC incluído, de endereço 0x44 na rede. O hardware utilizado foi:

- Mestre: PC104 modelo PFM535i com saída serial padrão DB9 RS-485 e software escrito em linguagem C++;
- Escravo: Microcontrolador PIC18F2550 programado em C via programa MPLAB de distribuição gratuita. Cada placa confeccionada tem um *jumper* no qual é possível selecionar se haverá ou não resistor de terminação de linha e assim cada nó pode ocupar qualquer posição ao longo da rede, como pode ser observado na Figura 4.6. Todos os nós Escravos na rede Modbus que utilizam esta CPU estão sob a restrição de não se utilizar de alguns recursos do PIC, a saber:
  - Pinos RC6 e RC7 - TX e RX utilizados pela serial para envio e recepção de dados.

  - Pino RB3 - Pino de saída digital utilizado para determinar a direção de fluxo de dados no barramento.
  - Interrupção Serial de Recepção - Utilizado pela rede para receber mensagens vindas do mestre de maneira prioritária, sem interferir diretamente no fluxo principal do programa.
  - Timer#3 - Utilizado para gerenciar o tempo de *Timeout* entre frames e entre bytes.
  - Interrupção de Timer#3 - Utilizado na detecção de falhas de envio/transmissão para resetar a máquina de estados da comunicação Modbus.

- *Tranceiver:* É utilizado um MAX485 da Maxim para conversão dos sinais UART/TTL do PIC em padrão RS-485[14].

- Barramento: O barramento utiliza um cabo UTP-CAT5 de dois pares para transmissão e recepção dos dados. Este barramento é polarizado, de acordo com a especificação, e utiliza-se da mesma fonte de alimentação do PC104. No barramento são encontrados os dois resistores de terminação de 120Ω com a função de evitar reflexão no cabo. O desenho esquemático da construção do cabo polarizado para conexão no PC104, juntamente com o cabo construído pode ser visto na Figura 4.7

- Conectores: Os conectores utilizados nos nós serão do tipo RJ11 de 4 vias como especificado no capítulo 3.

- Alimentação: A fonte de alimentação é uma fonte chaveada que fornece +5/+12 Volts na saída sendo esta a tensão necessária para o funcionamento tanto do nó Mestre quanto dos nós Escravos.

Neste teste, o mestre (PC104) envia o sinal de inicialização para os dois nós escravos. Após receber a confirmação de conexão efetuada com sucesso, os dois nós estarão no estado de espera. Depois um tempo de aproximadamente 10 segundos, que é o tempo necessário para o controlador iniciar o algoritmo de controle, são enviadas mensagens com intervalo de 25ms entre cada uma para o nó-sensor, obtendo-se assim o valor da tensão e em seguida para o nó atuador fornecendo o valor do *set-point*. Esta última sequência é repetida até que se encerre o programa.

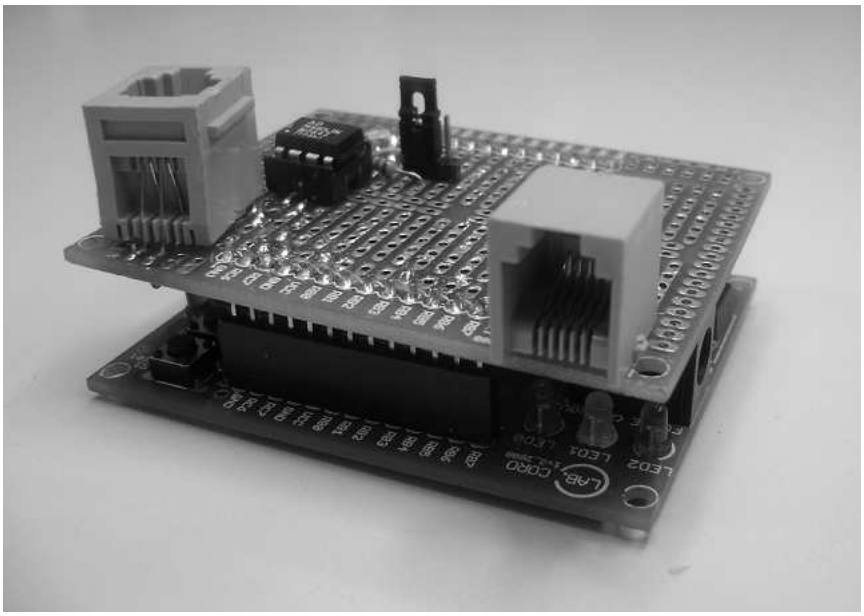

Figura 4.6: Placa de Teste Modbus completa com *Jumper* de resistor de terminação.

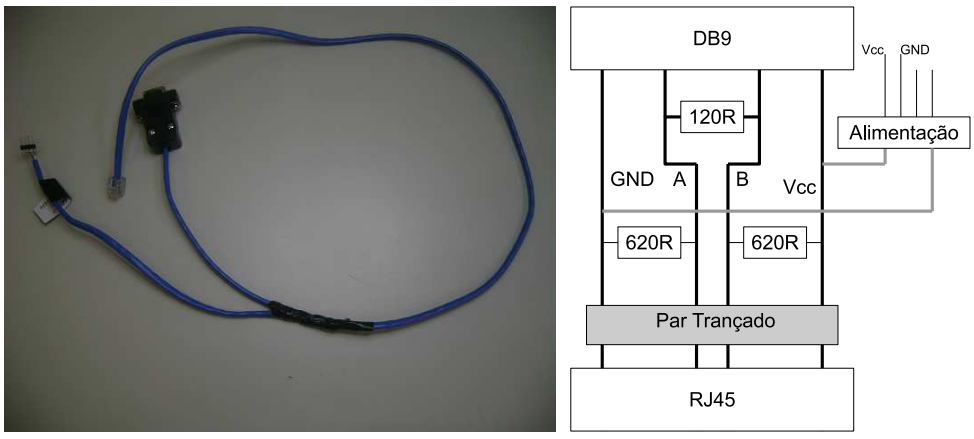

Figura 4.7: Cabo utilizado pelo Mestre para comunicação com os Escravos.

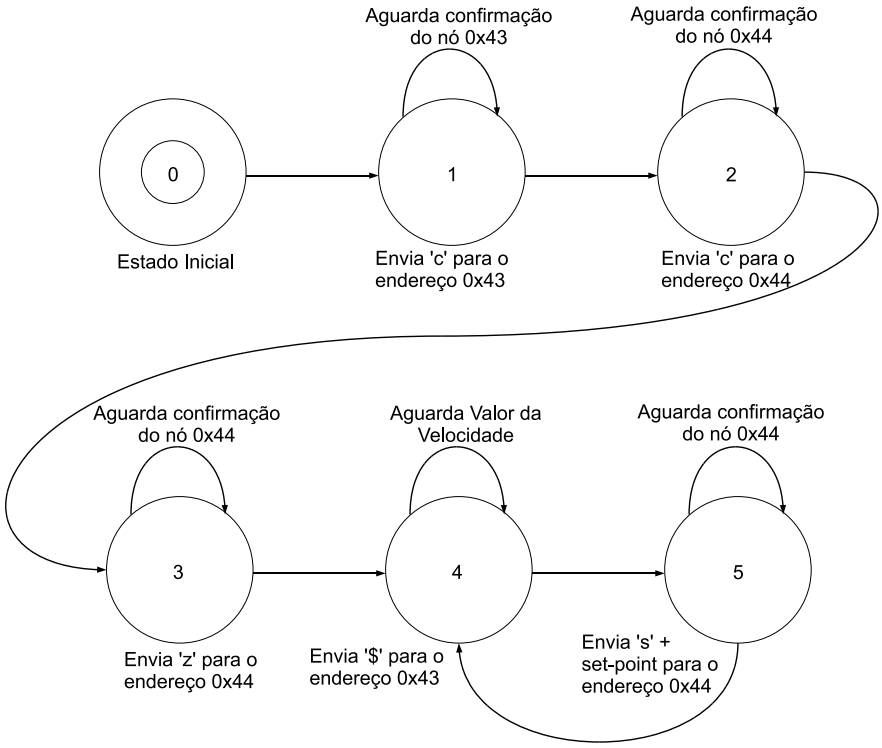

Figura 4.8: Máquina de Estados do teste em bancada.

O algoritmo da Figura 4.8 mostra o ciclo de execução do teste realizado em bancada.

4.6 Determinação dos tempos de comunicação

Um dos focos deste trabalho é a questão temporal. O que se deseja nesta parte é saber se este protocolo pode ser considerado como determinístico e para tanto, é necessário calcular os tempos de interesse e compará-los com os valores reais obtidos. Podemos separá-los por etapa onde cada uma corresponde a uma das fases do envio/recepção. Estes tempos são analisados do ponto de vista do mestre uma que é o elemento no qual se faz necessário saber o tempo máximo que uma determinada requisição demorará para ser respondida. Neste trabalho, esperou-se que o protocolo estivesse apto a responder a freqüências próximas de 100Hz. Isto garante o funcionamento correto dos nós em uso no carro. O nó sensor de velocidade das rodas é atualmente o que trabalha a uma frequência mais alta, sendo esta em torno de 50Hz.

A construção desta rede em PIC se deu através do uso de interrupção, ou seja, uma vez que não haja outras interrupções, o tempo máximo de resposta para uma requisição

será a soma dos tempos de processamento de pacotes de requisição adicionados cálculo do CRC somados tempo de envio dos bytes via serial mais o tempo da maior instrução existente em todo o código. Isto porque a interrupção, no caso do PIC, espera o fim da instrução para que o fluxo seja desviado. O tempo em que o byte trafega na rede pode ser negligenciado pois a distância que o dado percorre é desprezível se comparada a velocidade de transmissão do sinal elétrico dentro do fio.

Para exemplificar, segue o procedimento utilizado para envio do valor de sensor de rodas, uma vez que o mesmo não tem interrupções além das previstas no protocolo Modbus:

- Determinação da maior instrução presente no código;
- Contagem do número de instruções de loop e atribuições;
- Contagem do tempo utilizado na função de cálculo de CRC;
- Contagem do tempo nas chamadas de função executadas dentro da interrupção;
- Contagem do tempo utilizado no envio e recepção de n bytes via serial.

É importante ressaltar que todos os itens, exceto o primeiro, estão sumariamente ligados ao número de bytes recebidos e transmitidos. O último item é função também da velocidade de transmissão definida nos nós. Caso haja outras interrupções, faz-se necessário adicionar a estes itens o maior tempo gasto em uma interrupção como fator limitante superior. Outra informação importante concerne ao uso de rotinas de atraso nativas do MPLAB. É altamente recomendável que não haja nenhum atraso no programa uma vez que o mesmo é entendido como uma instrução atômica podendo este interferir de maneira significativa nos cálculos caso estejam presentes.

Este algoritmo para recepção e envio dos bytes para o nó sensor é composto de:

- 43 Atribuições - 1 ciclo de instrução cada;
- 20 Condicionais - 3 ciclos de instrução cada;
- 23 Chamadas de função - 3 ciclos de instrução cada;
- 6 bytes lidos via serial - 69μs cada a 115,2Kbps;
- 8 operações de deslocamento - 2 ciclos de instrução cada;

- 12 cálculos de CRC - 123 instruções cada cálculo de um byte;
- 6 bytes enviados via serial - 69μs cada a 115,2Kbps adicionado a um delay de 84μs para cada byte.
- 2 delays - Um deles de 84μs e outro de 840μs.

O ciclo de instrução está diretamente ligado ao oscilador externo de 20MHz. Cada ciclo de instrução deste código levará 0,2μs para ser executado. Deste modo, para este algoritmo, foi calculado um tempo de 2589μs entre a recepção do primeiro byte e a transmissão do último byte de dado. Deste valor 414μs correspondem a recepção serial, 918μs correspondem ao envio serial, 924μs correspondem aos delays e 333μs corresponde ao tempo total gasto nas instruções. Na prática o valor obtido por uma medição feita via osciloscópio foi de 2440μs, tempo este que é aproximadamente 94,2% do valor calculado. Era esperado um determinado erro, uma vez que os cálculos levam em consideração o pior caso em todas as instruções executadas e não considera possíveis otimizações no código. Este estudo de tempo mostra que a rede está preparada para receber tráfegos de dados em frequências que podem chegar a 100Hz, sem que a mesma se encontre sobrecarregada.

4.7 Teste no veículo

O teste no veículo tem a finalidade de, além de testar a rede em um ambiente menos estruturado, averiguar se a mesma é capaz de suprir todas as necessidades antes resolvidas por USB. Neste teste os nós originais presentes no carro foram substituídas pelos nós nos quais foi implementada a rede. O algoritmo utilizado foi o mesmo do teste em bancada, porém de modo a evitar oscilações bruscas em intervalos curtos de tempo na ponte-H, a frequência de amostragem dos valores da roda foram diminuídos para 0,5Hz, sendo conseqüentemente esta a frequência com que o set-point foi fornecido ao nó atuador. Um computador portátil foi utilizado para comunicar-se via Ethernet com o computador PC104 e os nós foram encaixados em suas respectivas posições. Utilizou-se a mesma fonte de alimentação chaveada utilizada no teste em bancada para alimentar o sistema. O veículo alvo do teste é o Astra descrito no Capítulo 1. O câmbio automático presente no mesmo foi útil no teste, pois uma vez que o carro se encontre na posição

Drive, tão logo o freio seja liberado, o carro começa a acelerar.

O algoritmo implementado no Mestre simula um controlador digital de malha fechada do tipo liga-desliga. A entrada deste controlador é o limiar de velocidade das rodas para o qual o acionamento do freio ocorrerá ou não. A variável manipulada é o valor do *set-point* de torque enviado ao nó atuador. O sinal de realimentação é fornecido pelo nó sensor de velocidade das rodas que a cada 2 segundos envia o valor da tensão lida no módulo A/D do PIC. Esta tensão, uma vez que esteja disponível ao Mestre, é facilmente convertida em velocidade utilizando-se para isto de constantes anteriormente calibradas neste sistema.

O Mestre inicialmente envia uma requisição de valor de velocidade das rodas ao nó responsável por esta informação. Este nó por sua vez, tão logo receba a mensagem do Mestre responde com um quadro Modbus onde está contida a informação de tensão em cada uma das rodas. Ao chegar ao Mestre, este quadro é desmontado e através de uma conversão, é obtido o valor em metros por segundo da velocidade de ambas as rodas. O controlador implementado compara esta velocidade com o limiar estabelecido de $6m/s$. Caso a velocidade seja inferior a este limiar, um novo quadro Modbus, desta vez endereçado ao nó atuador, é montado com a informação para estabelecer um novo *set-point* de zero no mesmo. Um *set-point* zero indica que o freio será completamente liberado pelo motor de modo que tão logo isto aconteça a roda iniciará novamente a se acelerar. Caso a velocidade seja superior a este limiar este quadro endereçado ao nó atuador conterá a mensagem para estabelecer um *set-point* de 80. Neste valor de *set-point* o nó atuador aciona o motor, de modo que este exerce um torque grande o suficiente para acionar completamente o freio, impedindo o movimento das rodas.

O teste consistiu em acionar o sistema completo de freios e leitura de rodas por 9 minutos perfazendo um total de aproximadamente 270 ciclos completos de leitura da velocidade das rodas e envio do set-point ao nó atuador. A velocidade da roda variou de zero a 21 metros por segundo e o sistema atuador de freio acionou corretamente o motor durante todo o intervalo de teste.

4.8 Conclusão do capítulo

Este capítulo mostrou que a rede pode ser implementada e que seu uso é factível. A análise do tempo mostrou que a mesma é determinística do ponto de vista do mestre, o que permite o seu uso como rede de tempo real. Este tempo pode se tornar demasiadamente longo caso haja muitas interrupções com prioridades altas e o tempo destas interrupções sejam longos. Esta rede permite o envio de comando múltiplos para um nó, uma vez que o pacote enviado pode ter de zero a 143 bytes e assim as possibilidades de comandos são amplas, porém o uso de mensagens muito longas é desaconselhado caso se queira uma frequência mais alta nas transmissões, uma vez que o tempo de processamento de um quadro de mensagem está diretamente ligado ao número de bytes enviados e recebidos.

O uso do *broadcast* pode ser útil para envio de comandos especiais aos nós. Seu uso em conjunto com a funcionalidade de *watchdog* presente nos nós, poderia por exemplo, servir como uma forma de manter a integridade do sistema, enviando a cada intervalo determinado de tempo uma mensagem a todos os nós avisando-os de que a rede está presente. A ausência desta mensagem indicaria ao nó que o sistema foi comprometido, cabendo ao mesmo executar ações de emergência para evitar danos tanto de si quanto do sistema como um todo.

Capítulo 5

Conclusões

A introdução de uma rede se justifica num ambiente onde vários dispositivos se comunicam com uma central e realizam funções baseadas em comandos fornecidos pela mesma. A necessidade de introduzir mais equipamentos no Veículo Autônomo, que é o principal sistema embarcado visado neste trabalho, demandou a criação de uma rede que atendesse aos objetivos citados no Capítulo 2 deste documento.

5.1 O Trabalho Realizado

A rede foi desenvolvida com base no protocolo Modbus sobre a camada física RS-485 e se mostrou amplamente viável para uso no destino especificado. Do ponto de vista de uso de recursos de hardware, no caso o PIC18F2550 foi mostrado que a rede demanda pouca memória, tanto de programa, quanto de memória RAM e este é um ponto importante, uma vez que a rede será introduzida em softwares já construídos para seu uso no carro, mas que utilizam comunicação USB.

Os recursos disponíveis no PIC também foram utilizados de maneira mínima, de modo que a rede necessita apenas dos pinos RC6, RC7 e B3 para serem o TX, RX e o controle de fluxo respectivamente. No que concerne aos recursos de software são utilizadas interrupção serial e o Timer#3, além das memórias para construção do programa como já citado, cumprindo o requisito de facilidade de implementação em microcontroladores.

Na questão temporal foi mostrado que o tempo de resposta e a frequência máxima de troca de informações podem ser facilmente calculados se o programa sobre o qual a rede será implementada for determinístico. O *throughput* da rede é dado em função

do número de bytes a serem transmitidos na mesma de modo que a velocidade máxima pode variar. Para os testes realizados onde o número de bytes transmitidos e recebidos estavam entre 6 e 8 bytes, uma frequência de requisição e resposta de mensagem de até 100 Hz pode ser facilmente obtida.

A sua construção exige apenas a inclusão de um transceiver MAX485 e conectores RJ11 nas placas de modo que o custo de implementação desta rede é baixo.

O uso do protocolo Mestre Escravo e a pequena distância que a rede terá que percorrer dentro do carro faz com que esta rede seja escalável de forma que, desde que cumprido o requisito de número máximo de nós, a diferença entre utilizar dois ou trinta e dois nós é mínima.

O uso de polarização nos cabos e resistores de terminação garantem a máxima eficiência na transmissão de dados e o uso de um computador com rede EIA/RS-485 nativa descarta a necessidade de um conversor, elemento este que certamente introduziria perda na qualidade do sinal.

5.2 Trabalhos Futuros

Para trabalhos futuros, espera-se uma mudança no layout da placa protótipo de modo a incluir os tranceptores e conectores necessários para que a rede já esteja disponível sem a necessidade de placa de expansão, e deste modo substituir todo o sistema de comunicação USB do carro por esta rede.

Uma vez que as mensagens podem ter tamanhos variados, é possível criar uma padronização nas mesmas utilizando-se dos 127 possibilidades de funções disponíveis no Modbus e também, com o cálculo do CRC para evitar erros na transmissão, pode-se criar mecanismos que garantam a entrega da mensagem mediante troca de informação entre o nó Mestre e os nós Escravos.

O uso de cabos e conectores blindados proporcionam uma qualidade cada vez maior do sinal, permitindo um aumento na velocidade de transmissão e uso de equipamentos em ambientes ruidosos, sujeito a perturbações dos mais variados tipos.

A utilização de um sistema operacional de tempo real permitira que o Mestre tenha seus tempos bem determinados, tornando assim a rede viável para outras aplicações com restrições de tempo mais rígidas como veículos aéreos autônomos.

Apêndice A

Código do Protocolo Modbus Implementado em PIC

```
/** V A R I A B L E S ************************************************/
unsigned char S[6];  //Buffer de solicitação;
unsigned char R[6]; //Buffer de resposta;
unsigned char frameCompleto = 0; //Indica se o quadro Modbus-RTU foi
//completamente preenchido;
unsigned char msgParaMim = 1; //Indica que o frame se refere a este
//endereço;
unsigned char byteMsg = 0; //Indica o índice do byte no vetor de
//mensagem;
unsigned char iniciaContador = 0;  //Contador para checagem de Timeout
int counter;
int state;
int conectado=0;

/*********************************************************************
 Auxiliar functions prototypes
*********************************************************************/
{...}
void checaEstado(void);

//====================================================================
//Tratamento de interrupção - São mostrada somente as duas relativas
// ao Modbus
//====================================================================
void high_isr (void)
{
//Caso receba uma mensagem, esta rotina checará se esta mensagem é
//para este nó. Caso seja, ela irá trata-lo. Caso não ela desabilitara a
//interrupção de recepção serial até que passe o tempo necessário para
//nova recepção.
//prioridade zero - Interrupção Serial
if(PIR1bits.RCIF) //Checa se houve interrupção serial
{
//Se a mensagem é realmente para este nó checa se é o primeiro byte;
if(msgParaMim == 1)
```

```
{

//Checa se é o primeiro byte;
if(byteMsg == 0)
{
//recebe o primeiro byte;
S[0] = RCREG;

//Se coincide com o meu endereço então eu recebo o resto;
if(S[0] == END || S[0] == 0x00)
{
msgParaMim = 1;
byteMsg = 1;
//Timeout de 750us para tempo entre bytes de msgs;
//Clock de 20Mhz calculo do tempo = 20Mhz/4 = 2e-7;
// 0,75ms/20us = EA6 = 3750; FFFF-EA6 = F159
TMR3L = 0x59;
TMR3H = 0xF1;
T3CONbits.TMR3ON = 1;
}
//Se não é para mim, ignoro todos os outros bytes por um
//determinado tempo;
else
{
msgParaMim = 0;
}
}
else //Se não for o primeiro byte, recebe o restante;
if(byteMsg == 1)
{
S[1] = RCREG;
byteMsg = 2;
TMR3L = 0x59;
TMR3H = 0xF1;
}
else
if(byteMsg == 2)
{
S[2] = RCREG;
byteMsg = 3;
TMR3L = 0x59;
TMR3H = 0xF1;
}
else
if(byteMsg == 3)
{
S[3] = RCREG;
byteMsg = 4;
TMR3L = 0x59;
TMR3H = 0xF1;
}
```

```
else
if(byteMsg == 4)
{
S[4] = RCREG;
byteMsg = 5;
TMR3L = 0x59;
TMR3H = 0xF1;
}
else
if(byteMsg == 5)
{
S[5] = RCREG;
byteMsg = 0;

//Recebi a msg completa, para o timer e coloca o valor dentro
//dele de tempo entre msg.
PIR2bits.TMR3IF = 0;
TMR3L = 0xD1;
TMR3H = 0xDD;
//Se eu recebi tudo, posso mandar a resposta;
frameCompleto = 1;
checaEstado();
}
}
else
{
//A mensagem não é para mim então toda vez que eu receber um byte,
//vou simplesmente ignorar e setar novamente o tempo entre
//mensagens
S[0] = RCREG;
TMR3L = 0xD1;
TMR3H = 0xDD;
msgParaMim = 0;
T3CONbits.TMR3ON = 1; //Inicia o contador
}
}
//Se não testa se já passou o tempo necessário para o envio da msg
// para o outro nó;
if(PIR2bits.TMR3IF == 1)
{
//Clock Interno de 48Mhz cálculo do tempo = 48Mhz/4 = 12e-7;
//1,75ms/20us = 8750 = 222E
T3CONbits.TMR3ON = 0;
PIR2bits.TMR3IF = 0;
TMR3L = 0xD1;
TMR3H = 0xDD;

//Retorna ao estado inicial
byteMsg = 0;
frameCompleto = 0;
msgParaMim = 1;
```

```
}

//Dentro do loop principal
void main(void)
{
//Necessário para a configuração das portas dos 3 leds, mais a porta B3
//(controle de fluxo), a serial a 115.2Kbps e o TIMER#3.
configModbus();

loop principal
while(true){...}
}

//Declaração da função checaEstado
//Rotina que checa se houve mudança de estado;
void checaEstado(void)
{
if(frameRecebido())
{
if(S[0] == 0x00)
{
LED0 = 1;
LED1 = 1;
LED2 = 1;
}
else if(S[3]==’c’)
{
R[0]=’o’;
R[1]= ’k’;
send(2);
conectado = 1;
State = IDLE;
}
//estado
else if(S[3]==’z’)
{
//Necessário para garantir toda recepção antes de responder;
State = RUN;
R[0]=’#’;
R[1]= ’#’; //Responde para o mestre que o estado foi mudado e envia
//os valores requeridos;
send(2);
}
//estado
else if(S[3]==’y’)
{
State = TERMINATE;
R[0]=’@’;
R[1]= ’0’; //Responde para o mestre que o estado foi mudado e envia
//os valores requeridos;
send(2);
```

```
}
else if(S[3]==’s’) //Indica que foi enviado um setPoint em S[2].
{
State = SETPOINT;
R[0]=’s’;
setPointPWM = S[2]; //Responde para o mestre que o estado foi
//mudado e envia os valores requeridos;
R[1] = setPointPWM;
send(2);
}
else
{
send_test();
State = IDLE;
}
}
}

//Estas outras funções utilizadas neste código são declaradas em um
//arquivo separado
//=====================================================================
//Funções criadas para possibilitar o uso transparente da Rede nos
// nós escravos
//=====================================================================
void Calc_CRC(unsigned short b, unsigned int* CRC);
void envia_serial(int Tx);
void send_test(void);
void send(int tamanho);
void configModbus(void)
{
//Portas
TRISB = 0x00; //PortB = 00000000; bits 0 - 7, saída

//Comunicação serial RS-232 Usart
//Define a velocidade da comunicação
TXSTAbits.BRGH = 1; //TXSTA.BRGH=1 - Alta velocidade
BAUDCONbits.BRG16 = 0; //Desabilita comunicação de 16bits
SPBRGH = 0x00; //Parte Alta do Byte de BAUDRATE;
SPBRG  = 0x1A;

//Transmissão
TXSTAbits.SYNC = 0; //TXSTA.SYNC=0 - Seleção do Modo
// Full-Duplex - modo assíncrono
TXSTAbits.TXEN = 1; //Habilita transmissão;
PIE1bits.TXIE = 0;
//Sobre interrupções página 100;

//Recepção
RCSTAbits.SPEN = 1; //Habilita comunicação serial;
BAUDCONbits.RXDTP = 0; //Desabilita inversão de bit;
PIE1bits.RCIE = 1; //Habilita Interrupção serial;
```

```
RCSTAbits.RX9 = 0; //Desabilita comunicação de 9bits;
RCSTAbits.CREN = 1; //Habilita recebimento de dados;

// Habilita Interrupções
INTCONbits.GIE = 1;   //Habilita interrupção Global
INTCONbits.PEIE = 1;   //Habilita interrupção de periféricos;
RCONbits.IPEN = 1; //Habilita prioridade nas interrupções;
PIE2bits.TMR3IE = 1; //Habilita interrupção de Timer3

//Configura Timer
T3CON = 0xCC; //Timer de 16 bits, oscilador interno, Pré-scaler = 00;

//Zera todas as portas
LATBbits.LATB0 = 0; //Led Verde
LATBbits.LATB1 = 0; //Led Amarelo
LATBbits.LATB2 = 0; //Led Vermelho
desabilitaEnvio(); //Controlador de Fluxo de dados na RS-485
//habilita/desabilita envio;
T3CONbits.TMR3ON = 0; //Clock inicia parado
TMR3H = 0x00;
TMR3L = 0x00;
TMR2 = 0x00;
counter = 0;
}

void Calc_CRC(unsigned short b, unsigned int* CRC)
{
   int carry;
   int i;
   CRC[0] ^= b & 0xFF;
   for (i=0; i<8; i++)
   {
      carry = CRC[0] & 0x0001;
      CRC[0]>>=1;
      if (carry) CRC[0] ^= GP;
   }
}

void send_test(void)
{
int aux;

habilitaEnvio(); //Habilita envio de dados - Modo SPEAKER;
Delay10KTCYx(2); //Delay para transição de listen para talker
if(CRC == 0)
{
//LATBbits.LATB1 = 1; //Recepção OK, acendo led amarelo de envio;
CRC = SEED; //Zera o CRC para calcular novamente;
envia_serial(S[0]);
Calc_CRC(S[0], &CRC);
envia_serial(S[1]);
```

```
Calc_CRC(S[1], &CRC);
envia_serial(S[2]);
Calc_CRC(S[2], &CRC);
envia_serial(S[3]);
Calc_CRC(S[3], &CRC);
CRCL = CRC&0x00FF;
aux = CRC>>8;
CRCH = aux&0x00FF;
envia_serial(CRCL);
envia_serial(CRCH);
Delay10TCYx(100); //Delay para transição de talk para listen
//LATBbits.LATB1 = 0; //Após terminado o envio, apago o led laranja;
}
desabilitaEnvio(); //Habilita recebimento de dados - Modo LISTENING;
//Inicia frame
S[0]=0; S[1]=0; S[2]=0; S[3]=0; S[4]=0; S[5]=0;
}
void envia_serial(int Tx){

TXREG = Tx;
while(!PIR1bits.TXIF){};
//LATBbits.LATB7 = 1;
Delay10TCYx(100);
}

//Checa se chegou mensagem, e caso tenha chegado, checa se o CRC está
//correto.
//Retorna 1 se sucesso e 0 se falha.
int frameRecebido(void){

if(frameCompleto==1)
{
frameCompleto=0;
CRC = SEED;
Calc_CRC(S[0], &CRC);
Calc_CRC(S[1], &CRC);
Calc_CRC(S[2], &CRC);
Calc_CRC(S[3], &CRC);
Calc_CRC(S[4], &CRC);
Calc_CRC(S[5], &CRC);
if(CRC == 0)
{
return 1;
}
else
{
habilitaEnvio(); //Habilita envio de dados - Modo SPEAKER;
Delay1KTCYx(20); //Delay para transição de listen para talker
//Transmite de volta os bytes que recebeu + um adicional no endereço
//para indicar erro;
LATBbits.LATB2 = 1; //Acende Led Vermelho para indicar ERRO no CRC;
```

```
envia_serial(S[0]+0x7F);
envia_serial(S[1]);
envia_serial(S[2]);
envia_serial(S[3]);
envia_serial(S[4]);
envia_serial(S[5]);
Delay10TCYx(100); //Delay para transição de talker para listen
desabilitaEnvio(); //Habilita recebimento de dados - Modo LISTENING;
}
S[0]=0; S[1]=0; S[2]=0; S[3]=0; S[4]=0; S[5]=0; return 0;
}
else
return 0;
}

//Envia o pacote de resposta completando-o com o CRC
void send(int tamanho)
{
int aux;
habilitaEnvio(); //Habilita envio de dados - Modo LISTENING;
Delay10KTCYx(2); //Delay para transição de listen para talker
if(CRC == 0)
{
//LATBbits.LATB1 = 1; //Recepção OK, acendo led amarelo de envio;
CRC = SEED; //Zera o CRC para calcular novamente;
envia_serial(S[0]);
Calc_CRC(S[0], &CRC);
envia_serial(S[1]);
Calc_CRC(S[1], &CRC);
for(aux = 0; aux < tamanho; aux++)
{ envia_serial(R[aux]);
Calc_CRC(R[aux], &CRC);
}
CRCL = CRC&0x00FF;
aux = CRC>>8;
CRCH = aux&0x00FF;
envia_serial(CRCL);
envia_serial(CRCH);
Delay10TCYx(100*tamanho); //Delay para transição de talker para listen
//LATBbits.LATB1 = 0; //Após terminado o envio, apago o led laranja;
}
desabilitaEnvio(); //Habilita recebimento de dados - Modo LISTENING;
S[0]=0; S[1]=0; S[2]=0; S[3]=0; S[4]=0; S[5]=0;}
```

Bibliografia

[1] André B. Bondi. Characteristics of scalability and their impact on performance. In *Proceedings of the 2nd international workshop on Software and performance*, pages 195–203, 2000.

[2] CANopen Application Days. *CAN specification V2.0*, acessado em 03/2009. Disponivel no site da Bosch, `www.can-cia.de`.

[3] Ai Chunli, Zhang Fengdeng, and Liu Rongpeng. Research on wireless backup for can in process control system. In *RFID Eurasia, 2007 1st Annual*, pages 1–6, Sept. 2007.

[4] Richard C. Dorf. *Modern control systems*. Prentice Hall, 2005.

[5] Larry L. Peterson e Bruce S. Davie. *Redes de computadores : uma abordagem de sistemas*. LTC - Livros Técnicos e Científicos, segunda edition, 2004.

[6] M. Farsi, K. Ratcliff, and M. Barbosa. An overview of controller area network. *Computing & Control Engineering Journal*, 10(3):113–120, Aug 1999.

[7] Hongwei Gao and Weiming Tong. Analysis and evaluation of fieldbus communication and protocol static characteristic. In *ICAIT '08: Proceedings of the 2008 International Conference on Advanced Infocomm Technology*, pages 1–5, New York, NY, USA, 2008. ACM.

[8] American Wire Gauge. Awg cope wire table. `http://www.interfacebus.com/ Copper_Wire_AWG_SIze.html`, acessado em 13/05/2009.

[9] Elias J. de R. Freitas et al Guilherme A. S. Pereira. Desenvolvimento de automação embarcada para um robô móvel baseado em um carro de passeio. *IX Simpósio Brasileiro de Automação Inteligente*, 1:1–6, 2009.

[10] Ronaldo Hüsemann and Carlos Eduardo Pereira. A multi-protocol real-time monitoring and validation system for distributed fieldbus-based automation applications. *Control Engineering Practice*, 15:955–968, 2007.

[11] InterlinkBT. As-interface tutorial. `http://www.ulpgc.es/hege/almacen/download/31/31119/asitutorial.pdf`, acessado em 13/03/2009.

[12] Lee S Lee KC and Lee MH. Worst case communication delay of real-time industrial switched ethernet with multiple levels. *IEEE Transactions on Industrial Electronics*, 53:1669–1676, 2006.

[13] Qing Liu and Yingmei Li. Modbus/tcp based network control system for water process in the firepower plant. *Intelligent Control and Automation. The Sixth World Congress on*, 1:432–435, 2006.

[14] Maxim. *RS-485/RS-422 Tranceivers Datasheet*, acesso em 05/2009. Disponível no site do Modbus, `http://www.maxim-ic.com/`.

[15] Kron Medidores. Conceitos básicos sobre rs-485. `http://www.kronweb.com.br/download2.php?id=353`, acessado em 13/11/2009.

[16] Tiago Mendonça. Controle automático do mecanismo de frenagem de um veículo não tripulado. *Projeto Final de Curso, Escola de Engenharia/UFMG*, 1(1):1–58, Out. 2009.

[17] Antonio Rogerio Messias. www.rogercom.com/download/Downloads.htm.

[18] Microchip. *PIC18F2455/2550/4455/4550 Datasheet*, acesso em 03/2009. Disponível no site do Modbus, `http://www.microchip.com/`.

[19] Modbus-IDA. *MODBUS Application Protocol Specification v1.1b*, acesso em 03/2009. Disponivel no site do Modbus, `http://www.modbus.org/`.

[20] Modbus-IDA. *Modbus Over Serial Line Especification*, acesso em 03/2009. Disponível no site do Modbus, `http://www.modbus.org/`.

[21] James R. Moyne and Dawn M. Tilbury. The emergence of industrial control networks for manufacturing control, diagnostics, and safety data. In *Proceedings of the IEEE*, volume 95, pages 29–47. Invited Paper, 2007.

[22] R. Obermaisser. Reuse of can-based legacy applications in time-triggered architectures. *Industrial Informatics, IEEE Transactions on*, 2(4):255–268, Nov. 2006.

[23] Open DeviceNet Vendor Association. *DeviceNet Specifications*, acesso em 03/2009. Disponivel no site da ODVA, `www.odva.org`.

[24] Peak Link Thecnik. *PCAN-PC/104-Card*, 2003.

[25] Max Power. http://www.control.com/1026204601/index_html.

[26] Profibus. *Profibus Technology And Aplication*, acesso em 03/2009. Disponivel no site da Profibus, `www.profibus.com`.

[27] Samson AG Mess. *Thecnical Information Foundation Fieldbus*, 2004. Disponivel no site da Samson, `www.samson.de`.

[28] Hu Sideng, Zhao Zhengming, Zhang Yingchao, and Wang Shuping. A novel modbus rtu-based communication system for adjustable speed drives. In *Vehicle Power and Propulsion Conference, 2008. VPPC '08. IEEE*, pages 1–5, 2008.

[29] Marcelo de Souza. *Proposta de um sistema de gestão empregando instrumentação inteligente e redes de campo na automação do processo de tratamento de água*. PhD thesis, Universidade de São Paulo, 2006.

[30] Miroslav Svéda and Radimír Vrba. Actuator-sensor-interface interconnectivity. *Control Engineering Pratice*, 7:95–100, 1999.

[31] Andrew S. Tanenbaum. *Structured Computer Organization*. Pearson - Prentice Hall, fifth edition, 2005.

[32] Texas. *Datasheet Max-232*, acesso em 06/2009. Disponível no site `http://www.datasheetcatalog.org/`.

[33] Shen XF et al. Wang Z. Real-time performance evaluation of urgent aperiodic messages in ff communication and its improvement. *Computer Standards & Interfaces*, 27:105–115, 2005.

[34] B. Younus, B. Ahmad, O. Bashir, and F. Azam. A network protocol for distributed embedded systems. In *Students Conference, ISCON '02. Proceedings. IEEE*, volume 1, pages 149–153 vol.1, Aug. 2002.