**TIAGO JOSÉ ÁVILA**

# MÓDULO MODBUS I/O

Florianópolis
2008

**TIAGO JOSÉ ÁVILA**

# MÓDULO MODBUS I/O

Trabalho apresentado à competência de Especialização em Desenvolvimento de Produtos Eletrônicos do Centro Federal de Educação Tecnológica de Santa Catarina – CEFET/SC.

Orientador: Prof. Golberi de Salvador Ferreira, Dr-Eng.

Florianópolis
2008

TIAGO JOSÉ ÁVILA

# MÓDULO MODBUS I/O

Este trabalho foi julgado adequado para obtenção do título de Especialista em Desenvolvimento de Produtos Eletrônicos e aprovado em sua forma final no Curso de Especialização em Desenvolvimento de Produtos Eletrônicos do Centro Federal de Educação Tecnológica, CEFET/SC, Florianópolis.

*Florianópolis, 29 de Agosto de 2008.*

**BANCA EXAMINADORA**

---

Prof. Golberi Salvador Ferreira
*Orientador*

---

Prof. Charles Borges de Lima
*Membro*

---

Prof. Marco Valério M. Villaça
*Membro*

ÁVILA, Tiago José. **Módulo Modbus I/O.** 2008. 53f. Trabalho Final (Especialização em Desenvolvimento de Produtos Eletrônicos) – Centro Federal de Educação Tecnológica de Santa Catarina, CEFET/SC, Florianópolis.

# RESUMO

Módulos de I/O (*inputs/outputs* ou entradas/saídas) são comumente utilizados na indústria para ler sinais, acionar cargas, para controle ou monitoramente de grandezas significativas em um determinado sistema e podem ser conectados através de uma rede de comunicação serial (usualmente RS232, RS485, RS422 ou TCP/IP) a controladores ou sistemas supervisórios SCADA (*Supervisory Control And Data Acquisition*). Esta monografia descreve o desenvolvimento de um módulo de I/O's, com finalidade comercial, baseado no protocolo de comunicação Modbus e meio físico para transmissão de dados no padrão RS232. Para este projeto, foram aplicadas 2 tecnologias distintas de microcontroladores e 2 implementações semelhantes entre si objetivando as mesmas características e benefícios para o integrador/usuário final. As estruturas de software embarcado e hardware são descritas e, evidenciados os prós e contras do uso de cada processador para um definido módulo de Módbus I/O (12 entradas e 8 saídas digitais) baseados no modo RTU (*Remote Terminal Unit*) deste protocolo. As implementações do projeto e testes são feitos em placas de circuitos impressos dedicadas para cada processador; microcontroladores AT89C51RC (CISC) e ATmega64 (RISC), ambos do fabricante Atmel. Uma vez que um módulo de I/O usualmente atua como um escravo em uma rede de comunicação de dados, os testes de validação foram efetuados com o uso de um CLP (Controlador Lógico Programável) e softwares (executados em um microcomputador padrão IBM-PC) operando como mestres de uma rede Modbus RTU.

Palavras chave: Módulo I/O, protocolo, Modbus, microcontroladores

ÁVILA, Tiago José. **Módulo Modbus I/O.** 2008. 53f. Trabalho Final (Especialização em Desenvolvimento de Produtos Eletrônicos) – Centro Federal de Educação Tecnológica, CEFET/SC, Florianópolis.

# ABSTRACT

I/O Modules (inputs/outputs) are commonly used in industry to read signals, to trigger loads for control or monitoring of significant quantities and it can be connected by a serial network communication (usually RS232, RS485, RS422 or TCP/IP) to controllers or SCADA supervisors systems (Supervisory Control And Data Acquisition). This monograph describes the development of an I/O module, for commercial purposes, based on the Modbus communication protocol with the physical layer for data transmission at standard RS232. For this project, two different microcontroller technologies were applied and were built two similar implementations aiming the same features and benefits to the integrator/end user. The structures of the embedded software and hardware are described and highlighted the pros and cons using each processor module to a specific Modbus I/O (12 inputs and 8 digital outputs) based on the RTU mode (Remote Terminal Unit) of this protocol. The implementations of the design and tests were made by using dedicated circuit boards to each processor; microcontrollers AT89C51RC (CISC) and ATmega64 (RISC), both made by the manufacturer Atmel. Since an I/O module usually acts as a slave at a data network communication, validation tests were performed by using a PLC (Programmable Logic Controller) and softwares (running on a standard IBM-PC microcomputer) operating as masters of a Modbus RTU network.

Key words: I/O Module, protocol, Modbus, microcontrollers

# SUMÁRIO

# ÍNDICE DE FIGURAS

# 1 INTRODUÇÃO

Este trabalho descreve o desenvolvimento do projeto de um módulo Modbus I/O, que é um equipamento capaz de interagir com sensores (bóias de nível, transdutores de nível, etc.) e acionar equipamentos (bombas, válvulas, etc.) remotamente com o auxílio de um microcomputador (ou qualquer controlador para esta finalidade) através de um software apropriado que aplique o protocolo de comunicação serial denominado Modbus.

Este tipo de equipamento é amplamente usado para automação industrial podendo ser comandado por um software de supervisão comercial ou até mesmo por um CLP (Controlador Lógico Programável) conectados por uma conexão serial para que, por exemplo, ligue/desligue bombas e válvulas, outrossim, monitorando sinais de bóias de nível e sensores fim-de-curso.

Um módulo Modbus I/O é um dispositivo que apenas recebe informações e as aplica com leituras de sensores (fins-de-curso, por exemplo) e/ou escrita de saídas (a relés, por exemplo) não sendo capaz de controlar plantas industriais autonomamente. Para que seja possível o acionamento de motores e outras cargas, é necessário que o módulo Módbus I/O seja instalado em um quadro elétrico de comando com as devidas proteções de sobrecorrente, falta de fase e seqüência de fase (quando necessário) assim como o uso de contatores ou outros dispositivos de acionamento de potência, uma vez que este módulo é capaz apenas de acionar o comando de cargas e nunca diretamente.

Há módulos Modbus I/O para acionamento de cargas analógicas como a abertura proporcional de válvulas ou controle de bombas dosadoras e, mesmo para tais fins, somente são aplicados no comando e nunca na etapa de potência destas cargas.

A comunicação entre o módulo Modbus I/O e o dispositivo controlador (software remoto ou um CLP) é feita na topologia mestre-escravo. O mestre de uma rede de comunicação Modbus é capaz de ordenar e requisitar tarefas para os escravos, como a requisição da leitura de uma determinada entrada digital em um módulo Modbus I/O remoto ou, por exemplo, ligar o motor de uma esteira. Ao escravo compete atender requisições do único mestre existente na rede, ou seja, cada módulo Modbus I/O somente irá processar e executar as informações oriundas do mestre na rede e respondendo somente a ele.

Existem três modos distintos de protocolo Modbus: RTU, ASCII e TCP/IP. Destes, RTU e o ASCII, usualmente utilizam os meios físicos RS232 ou RS485, enquanto o Modbus TCP/IP, usualmente, utiliza como meio físico a Ethernet. Cada modo do protocolo Modbus será detalhado nesta monografia.

O projeto descrito neste trabalho compreende a implementação de módulos de Modbus I/O com finalidade comercial e feitos com recursos próprios, utilizando-se duas tecnologias de microcontroladores distintas aplicados em placas dedicadas para cada microcontrolador, ambos com I/O’s de uso geral para futura análise e seleção de tecnologia à ser implementada no produto final (comercial).

O meio físico determinado foi o padrão RS232 facilitando a conectividade direta com computadores (sem a necessidade de conversor RS232 para RS485, por exemplo) e com CLP’s que utilizam este padrão de meio físico. Com o uso do meio físico RS232 foi determinado o modo RTU pois este tem densidade maior de dados que o modo ASCII (detalhado neste trabalho).

O projeto em questão, deve ser capaz de gerencias 12 entradas digitais, opto-acopladas, sensíveis a níveis de tensão 24Vcc e 0V para níveis lógicos 1 e 0, respectivamente (como leitura de bóias de nível de um tanque). Deve ser capaz de acionar 8 cargas isoladas oticamente de 24Vcc e corrente máxima de 100mA (por exemplo, acionar bobinas 24Vcc de contatores de bombas). Apesar de se tratar de um módulo Modbus I/O genérico, as especificações deste projeto visam atender as necessidades de automação em saneamento (estações de tratamento de água e esgoto) para a empresa a qual presto serviços na área de automação, Rotária do Brasil Ltda.

# 2 FUNDAMENTAÇÃO TEÓRICA[1]

## 2.1 Microcontrolador

Com o avanço da tecnologia e a utilização da eletrônica digital por grande parte das empresas, o emprego de microcontroladores vêm sendo muito requisitado para um melhor desenvolvimento da produção, diminuindo os custos e trazendo benefícios para as empresas que utilizam este sistema. Segundo Marinho e Marinho (2002), é importante salientar, que considerando a relação custo/benefício, os microcontroladores podem não só ser usados em empresas de médio/grande porte, como podem também ser utilizados em vários projetos de eletrônica, na substituição de vários componentes digitais, obtendo-se assim no final do projeto melhor acabamento – pois um microcontrolador ocuparia um menor espaço físico – e uma vez que todos os comandos seriam executados via software.

Na década de 70 começa-se a utilizar microprocessadores em computadores para uma maior eficiência no processamento de dados. O microprocessador Intel foi um dos precursores e a partir daí, houve uma preocupação em melhorar cada vez mais o sistema de processamento de dados através desses componentes. Baseado na arquitetura de um microprocessador e seus periféricos foi criado um componente que (fisicamente em uma unidade) comportasse todo um sistema que equivalesse a um microprocessador e seus periféricos; assim surgiu o microcontrolador.

Os microcontroladores utilizados neste projeto possuem duas arquiteturas distintas de núcleo: ATmega64 de arquitetura RISC (*Reducted Instruction Set Core*, núcleo com set de

[1] Baseado na NBR 10520: 2002 da ABNT.

instruções reduzidas) e AT89C51RC de arquitetura CISC (*Complex Instruction Set Core*, núcleo com set de instruções complexas).

Segundo Leonardo Marcilio Schunk e Aldo Luppi (2004), a maior diferença entre ambas arquiteturas é que núcleos RISC possuem barramentos separados para a memória de programa e demais periféricos, enquanto os núcleos CISC possuem apenas um barramento para o transporte de todos os dados; usualmente processadores CISC são capazes de executar tarefas matemáticas mais rápidas enquanto os processadores RISC são capazes de executar programas com velocidade mais elevada.

## 2.2 Padrão RS232

A interface RS232 é um padrão serial e assíncrono usualmente encontrado em microcomputadores padrão IBM-PC, não necessitando de *clock* de sincronismo de bits. Cada byte é sincronizado por um bit de partida (*start bit*) e o sistema reconhece automaticamente o *clock* pelo sinal recebido, e no fim do byte, por 1 (ou 1,5 ou 2) bit de parada (*stop bit*).

A comunicação serial RS232, quando está em *Idle* (ociosa), tem nível lógico 1. A transmissão começa quando o bit de partida aparece com nível lógico 0. Então cada bit de dado é recebido/enviado, sendo que o bit menos significativo é enviado primeiro, e por fim vem o bit de parada, que tem nível lógico 1, finalizando a transmissão daquele byte em questão. O termo em inglês para este byte envolto pelos bits de partida e parada é *framed byte*.

Em termos de tensão, a norma estabelece que o nível lógico 1 deve estar entre -3Vcc e -15Vcc, e o nível lógico 0 entre +3Vcc e +15Vcc, e a região entre +/-3Vcc é indefinida, permitindo alta imunidade a ruídos entre os níveis (ou faixas) que o sinal é considerado como nível lógico aceitável.

A definição de *framed byte,* anteriormente descrita é trafegada pelos pinos de recepção (RX) e transmissão (TX); os pinos nomeados de RTS, CTS, DCD, DSR, DTR, e TI (não utilizados para a comunicação serial deste projeto) são sinais paralelos e representam bits de controle e também possuem níveis de tensão padrão RS232.

## 2.3 Protocolo Modbus

Modbus é um protocolo de transmissão de dados desenvolvido por Gould Modicon (agora Schneider Electric) para sistemas de controle de processos e teve sua primeira aplicação em 1979. No entanto, é considerada como um protocolo "público" e se tornou o padrão para vendedores de integração.

Contrariamente a outros protocolos, não possui interface física definida. Modbus é um protocolo público, simples e flexível, que permite aos dispositivos troca de dados discretos e analógicos.

Segundo MACKAY (2004), os usuários finais estão cientes que especificando o Modbus como interface necessária entre subsistemas é o caminho para chegar a vendedores de integração  e obter as opções mais vendidas ao menor custo. Pequenos fabricantes de equipamentos também estão cientes que devem oferecer Modbus com RS232 e/ou RS485 para venderem seus equipamentos a integradores de sistemas para uso em projetos maiores. Integradores de sistemas sabem que o Modbus é uma interface segura para comunicação, assim como eles podem ter certeza de encontrar equipamentos suficientes no mercado para satisfazerem as necessidades dos projetos e suportar as inevitáveis mudanças no decorrer deste.

A comunicação entre um módulo Modbus I/O e o dispositivo controlador (software remoto ou um CLP) é feita na topologia mestre-escravo (máximo 247 escravos por rede). O mestre de uma rede de comunicação Modbus é capaz de ordenar e requisitar tarefas para os escravos, como a requisição da leitura de uma determinada entrada digital em um módulo Módbus I/O remoto ou, por exemplo, ligar o motor de uma esteira. Ao escravo compete atender requisições do único mestre existente na rede; ou seja, cada módulo Modbus I/O somente irá processar e executar as informações oriundas do mestre na rede e responder somente a ele.

A documentação desse protocolo foi disposta desde os primórdios de suas aplicações como “código aberto” garantindo sua proliferação entre fabricantes de produtos que utilizam comunicação de dados em ambientes industriais.

A camada física deste protocolo foi especificada como livre ficando a critério do fabricante defini-la. Inicialmente o padrão RS232 foi amplamente aplicado tornando-se um uso comum entre fabricantes de dispositivos industriais e, posteriormente, foi introduzido no meio industrial o padrão RS485, que tem como principais características um maior alcance (aproximadamente 1km), se comparado ao até então padrão dominante, o RS232 (aproximadamente 12m), e maiores velocidades de comunicação. Apesar do padrão RS485 ter uso predominante na indústria (motivos já citados), o padrão RS232 ainda possui seu espaço por ter maior conectividade (sem a necessidade de conversores RS232 para RS485) com sistemas de supervisão instalados em computadores padrão IBM-PC que, usualmente, possuem tal padrão de comunicação. Muitos CLP’s comercializados hoje em dia utilizam software de programação para plataforma IBM-PC e, usualmente, tais CLP’s vem dotados de porta de comunicação com o padrão RS232, tornando conveniente o uso desta mesma porta de comunicação para a aplicação de protocolos de comunicação industriais, como o protocolo Modbus.

O padrão Modbus se destaca pela flexibilidade e fácil implementação. O protocolo Modbus tem funções de acesso a registros específicos e a registros em que o desenvolvedor de um equipamento industrial pode usar para ajustar configurações específicas de seu produto, todavia, mantendo a compatibilidade com ou outros equipamentos que comunicam com este protocolo. Em pouco tempo, o protocolo Modbus, foi implementado por diversos fabricantes de equipamentos industriais tornando-se o padrão para redes de comunicação industrial.

Muitos sensores inteligentes (dotados de um processador de dados) com implementação de uma interface Modbus escravo, são capazes de fornecer valores exatos de leituras do seu conversor analógico/digital eliminando qualquer possibilidade de interferências eletromagnéticas oriundas de um ambiente hostil. Tal imunidade a interferências eletromagnéticas ocorre pelo fato que todo protocolo de comunicação industrial utiliza uma verificação dos dados transmitidos chamado de *checksum*, que é compreendido em um número resultante de um cálculo que envolve todos os dados da mensagem a ser transmitida sendo este anexado ao fim da mesma. Mesmo que, por interferência eletromagnética, um dado ou mais tenham sido corrompidos durante a transmissão (com a alteração de um ou mais bits), o *checksum* será a verificação de integridade da mensagem transmitida. Uma vez que o *checksum* não é o correspondente da mensagem recebida, esta é descartada.

Com o uso de usuais sensores com saída analógica 4 a 20mA, mesmo se tratando de fontes de corrente que, por definição devem suprir a resistência dos cabos mesmo a longas distâncias (determinadas pelo fabricante); estas leituras estão sujeitas a interferências induzidas devido a um possível ambiente agressivo ocasionando a leitura que difere ao valor real fornecido pelo sensor em sua origem.

### 2.3.1 Funções/Registros de dados

Uma vez que o protocolo de comunicação Modbus foi inicialmente desenvolvido para a comunicação entre CLP's, este tem funções de acesso a registros de forma objetiva para total interface com controladores/transdutores em ambiente industrial; cada função acessa a um tipo específico de registro.

Por definição, os registros são de 16bits (2 bytes) e podem ser exclusivos de leitura, como entradas digitais (cada registro de 16bit corresponde a 16 entradas digitais) ou analógicas (1 registro para resolução até 16bits); e podem ser de leitura/escrita como em registros de saídas digitais ou analógicas e, registros de configurações específicas proprietárias de algum fabricante de equipamento compatível com o protocolo Modbus. Fabricantes podem fazer leituras/escritas proprietárias com mais de 16bits, como por exemplo, o uso de valores 32bit com o uso de dois registros (total 4bytes).

Uma vez que funções exclusivas do protocolo Modbus como a função de número 2 (0x02, hexadecimal) para leitura de entradas digitais não podem ser usadas para outro fim diferente de leitura de entradas digitais, equipamentos dotados deste protocolo de comunicação possuem compatibilidade absoluta. Havendo necessidade de alguma configuração específica para um determinado equipamento, o fabricante deve informar quais os registros e quais suas funções; pois suas configurações proprietárias não constam nas definições do protocolo Modbus.

As funções são empregadas no protocolo com o uso de um código de função de 8bits, ou seja, número máximo de funções para este protocolo é de 255 (0xFF, hexadecimal).

Abaixo, códigos de funções (apresentados em hexadecimal) seguidos dos tipos de dados mais usuais para aplicações de controle e supervisão.

- (0x01) *Read Coils* – Ler saídas digitais;
- (0x02) *Read Discrete Inputs* – Ler entradas digitais;
- (0x03) *Read Holding Registers* – Ler registros inteiros de 16bits (temporizadores, contadores, saídas analógicas, etc);
- (0x04) *Read Input Registers* – Ler registros inteiros de 16bits (entradas analógicas, em freqüência, etc);
- (0x05) *Write Single Coil* – Escrever em uma saída digital;
- (0x06) *Write Single Register* – Escrever em um registro inteiro de 16bits (temporizadores, contadores, saídas analógicas, etc);
- (0x0F) *Write Multiple Coils* – Escrever em saídas digitais;
- (0x10) *Write Multiple Registers* – Escrever em registros inteiros de 16bits (temporizadores, contadores, saídas analógicas, etc);

Há também registros para armazenamento de arquivos e outros dados que fabricantes de transdutores/controladores podem utilizar para gravar configurações, eventos e histórico de processos; outrossim, registros com sub-funções e função de interface de transporte encapsulado.

### 2.3.2 Mensagens de erro

Estas são originadas de um escravo respondendo uma requisição do mestre da rede. Usualmente, mensagens de erro ocorrem por requisição indevida do mestre da rede Modbus.

Abaixo, códigos de erros, exceções, (em hexadecimal) mais usuais empregados em módulos de Modbus I/O seguidos de suas definições.

- (0x01) *Illegal Function* – Função ilegal. Caso o escravo não tenha a função solicitada implementada em seu sistema. Ex.: mestre tenta escrever múltiplas registros simultâneos (função 0x10) mas o escravo tem a implementação apenas da função de um único registro por mensagem (função 0x06);
- (0x02) *Illegal Data Address* – Endereço de dado ilegal. Quando o endereço do dado requisitado para leitura/escrita por parte do mestre, não tenha sido implementado pelo escravo. Ex.: mestre quer escrever na saída digital 18 quando o escravo possui 16 saídas digitais;
- (0x03) *Illegal Data Value* – Valor de dado ilegal. Ocorre em uma função de escrita solicitada pelo mestre não sendo um valor válido para o escravo. Ex.: mestre solicita escrever o valor 0x0FFF em uma saída analógica do escravo de 10bits;
- (0x04) *Slave Device Failure* – Falha em dispositivo do escravo. Algum periférico do escravo não é capaz de responder. Ex.: mestre solicita leitura de uma entrada analógica mas o processador do escravo detecta conversão não completa (por *timeout* interno) não sendo possível responder a solicitação embora o escravo em questão outrora tivesse tal capacidade;

### 2.3.3 Modbus RTU

O modo RTU (*Remote Terminal Unit*), usualmente implementado nos padrões físicos RS232 e RS485, utiliza mensagens compostas por seqüência de dados transmitidos continuamente.

Cada dado serial é formado por 8 bits (1 byte) que contém 2 caracteres de 4 bits que representam 2 números em hexadecimal. A principal vantagem do modo RTU é a maior densidade de dados para uma mesma taxa de transmissão, comparado ao o modo ASCII.

Cada mensagem é transmitida por uma seqüência de caracteres enviados serialmente com seus bits de controle totalizando 11 bits: 1 bit de partida, 8 bits de dados, 1 bit de paridade e 1 bit de parada. Paridade pode ser “par”, “ímpar” ou “sem paridade” (para o caso de “sem paridade”, deve-se usar 2 bits de parada).

#### 2.3.3.1 Mensagem RTU

Uma mensagem Modbus RTU é formada por um tempo de início (*start*) de 3,5 caracteres, posteriormente, o endereço do escravo que irá receber/responder a mensagem, a função correspondente à requisição/ordenança esperada pelo mestre, os dados objetos de requisição/ordenança e o *checksum* CRC-16 (*Cyclical Redundancy Checking*) seguido por silêncio de 3,5 caracteres (podendo ser este último tempo o mesmo que o marcado como tempo de início em caso de sucessivas requisições pelo mestre da rede). A figura 1 mostra a formação de uma mensagem segundo o protocolo Modbus RTU.

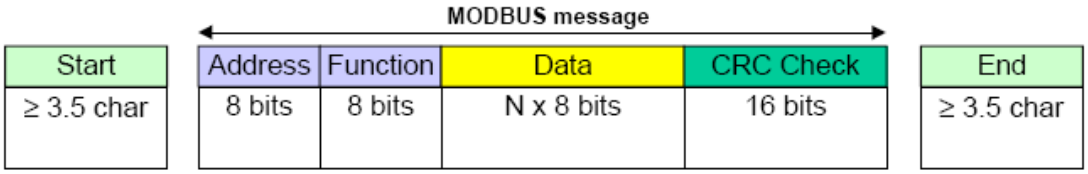


Figura 1 – Mensagem RTU

A mensagem enviada (seqüência de bytes) não deve exceder o tempo máximo de 1,5 caracteres, caso o tempo exceda, tanto o mestre quanto o escravo devem desconsiderar a

mensagem. A figura 2 mostra a temporização entre bytes, correta (indicado como *Frame 1*) e incorreta (indicada como *Frame 2*), segundo o protocolo Modbus RTU.

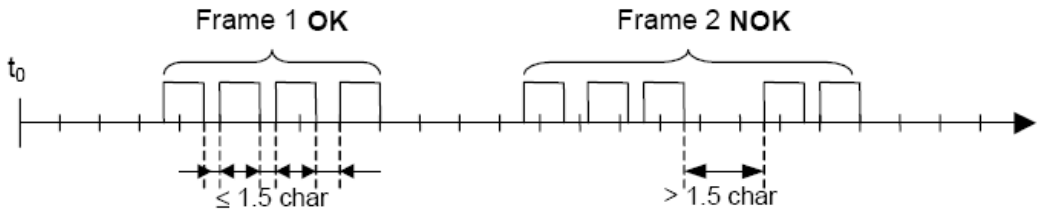


Figura 2 – Tempo entre bytes RTU

2.3.3.2 Checksum

O método aplicado para *Checksum* no modo RTU, independe do uso ou não da conferência de paridade de cada byte, é o CRC (*Cyclical Redundancy Checking*) que é formado por um campo de 16 bits composto por 2 bytes. Esta conferência envolve todos os caracteres enviados/recebidos contabilizados ao término do tempo de 3,5 bytes, indicador de fim de mensagem.

2.3.4 Modbus ASCII

O modo ASCII (*American Standard Code for Information Interchange*), usualmente implementado nos padrões físicos RS232 e RS485, utiliza mensagens compostas por seqüência de dados transmitidos em seqüência contínua.

Cada dado serial é formado por 7 bits, 1 caractere ASCII representando 1 número em hexadecimal. O modo ASCII necessita da transmissão de 2 dados seriais para se obter a mesma informação (0x00 a 0xFF), oqual o modo RTU é capaz de realizar com apenas 1 dado serial. Este modo é aplicado em casos em que haja a impossibilidade da implementação do

modo RTU, usualmente relacionado a dificuldades nas temporizações necessárias para correto funcionamento do protocolo.

Cada mensagem é transmitida por uma seqüência de caracteres enviados serialmente com seus bits de controle totalizando 10 bits: 1 bit de partida, 7 bits de dados, 1 bit de paridade e 1 bit de parada. Paridade pode ser “par”, “ímpar” ou “sem paridade” (para o caso de “sem paridade”, deve-se usar 2 bits de parada).

#### 2.3.4.1 Mensagem ASCII

Uma mensagem Modbus ASCII é formada por um caractere de controle ASCII “:” (0x3A, hexadecimal), posteriormente, o endereço do escravo que irá receber/responder a mensagem, a função correspondente a requisição/ordem esperada pelo mestre, os dados objetos de requisição/ordenança e o *checksum* LRC seguido por caracteres de controle ASCII “CR”(0x0D) e “LF” (0x0A). A figura 3 mostra a formação de uma mensagem segundo o protocolo Modbus ASCII.

| Start | Address | Function | Data | LRC | End |
|---|---|---|---|---|---|
| 1 char<br>: | 2 chars | 2 chars | 0 up to 2x252 char(s) | 2 chars | 2 chars<br>CR,LF |

Figura 3 – Mensagem ASCII

#### 2.3.4.2 Checksum

O método aplicado para *Checksum* no modo ASCII, independe do uso ou não da conferência de paridade de cada byte, é o LRC (*Longitudinal Redundancy Checking*) que é formado por um campo de 8 bits composto por 2 bytes em ASCII. Esta conferência envolve todos os caracteres enviados/recebidos contabilizados ao término da mensagem, exceto caracteres de controle.

### 2.3.5 Modbus TCP/IP

Uma comunicação através do Modbus TCP/IP requer o estabelecimento de uma conexão TCP entre Cliente e Servidor com o uso da porta registrada 502 (porta definida pelo protocolo Modbus). Cada dado TCP (dentro do encapsulamento Modbus) é formado por 8 bits (1byte) que contém 2 caracteres de 4 bits que representam 2 números em hexadecimal, semelhante ao modo RTU. A figura 4 mostra a disposição cliente/servidor e o fluxo de informação; requisição (*Request*) e confirmação da requisição (*Confirmation*) pelo cliente; Indicação de requisição (*Indication*) e resposta a indicação de requisição recebida (*response*) pelo servidor. O cliente ocupa a posição de mestre da rede enquanto o servidor a posição de escravo, segundo protocolo Modbus TCP/IP.

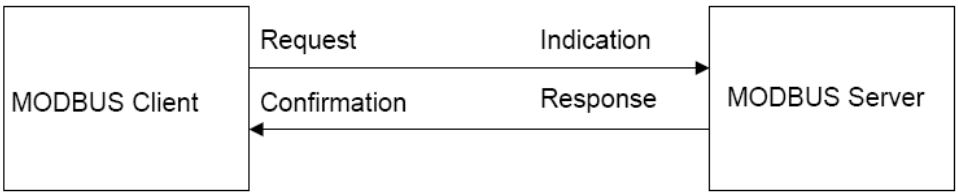


Figura 4 – Disposição Cliente/Servidor

#### 2.3.5.1 Mensagem TCP/IP

Cada mensagem Modbus é encapsulada dentro do protocolo TCP/IP. Cada mensagem TCP pode apenas enviar uma requisição sem a necessidade de adicionar qualquer valor calculado de *checksum* além do executado pelo protocolo TCP (CRC-32, sobre ethernet). A figura 5 mostra o encapsulamento do Modbus TCP/IP sobre ethernet.

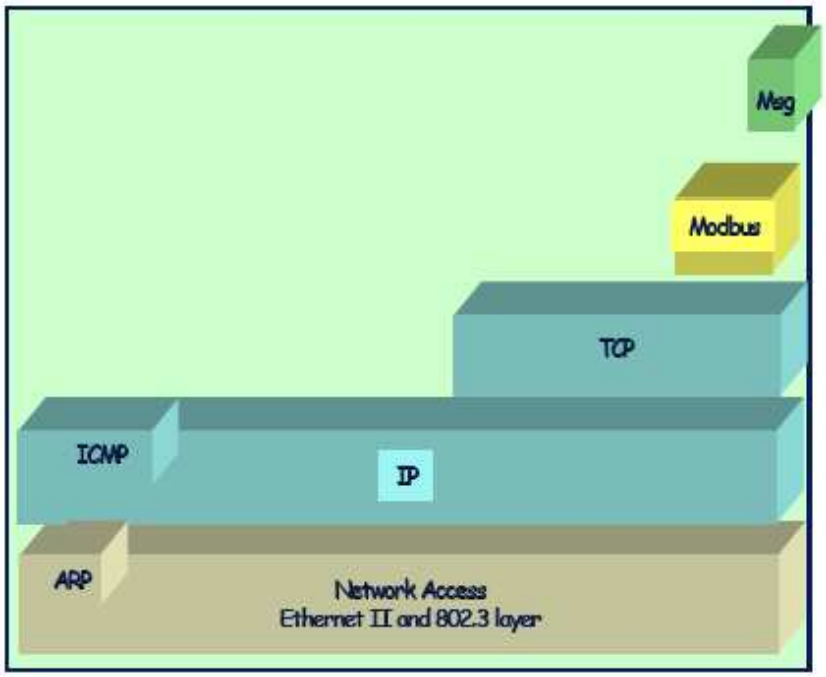


Figura 5 – Encapsulamento Modbus TCP/IP

## 2.4 Modbus I/O

Modbus I/O é chamado todo módulo eletrônico capaz de se comunicar através do protocolo Modbus para interfacear entradas/saídas (*I/O's*) digitas e/ou analógicas como escravos de uma rede Modbus.

Usualmente, entradas digitais (isoladas galvanicamente ou não) são sensíveis aos níveis: 24Vcc, 12Vcc, 0Vcc, 110Vca ou 220Vca; entradas analógicas (isoladas galvanicamente ou não) podem ser sensíveis às variações (*ranges*): -10 a +10Vcc, 0 a 10Vcc, 0 a 20mA, 4 a 20mA ou sensores de temperatura específicos.

As saídas digitais podem ser isoladas galvanicamente como a relé, opto-acoplada (com coletor e emissor disponíveis) ou não isoladas como usuais PNP e NPN. As saídas analógicas (isoladas galvanicamente ou não) usualmente são para os níveis 0 a 10Vcc, 0 a 20mA ou 4 a 20mA. Há também módulos que assumem saídas "híbrida" por serem fisicamente digitais

(usualmente com isolação galvânica) todavia acionam cargas proporcionais, com PWM (*Pulse Width Modulation,* modulação por largura de pulso), podendo ter largura de pulso e/ou freqüência programáveis.

Este tipo de equipamento é utilizado amplamente para controle e supervisão de sistemas automatizados podendo ser comandado por controladores com interface de mestre de uma rede Modbus e softwares SCADA (*Supervisory Control And Data Acquisition*).

Em muitos casos, módulos Modbus I/O podem substituir o uso de módulos de expansão de CLP's (que podem custar mais caros que módulos Modbus I/O genéricos). CLP's podem utilizar redes Modbus para leitura de grandezas analógicas de diversos transdutores em uma única rede reduzindo a fiação necessária e diminuindo a necessidade de entradas analógicas neste controlador, facilitando a manutenção e a detecção de falhas no sistema.

Os módulos de I/O possibilitam que empresas voltadas apenas para desenvolvimento de software possam implementar supervisão de plantas industriais com as leituras de sinais (digitais/analógicos), tendo como pré-requisito o conhecimento do protocolo Modbus a ser utilizado pelo escravo (módulo de I/O).

# 3 PROPOSIÇÃO/HIPÓTESES

Este projeto visa atender, além de sistemas de supervisão e controladores em geral (por se tratar de um módulo Modbus I/O genérico), a CLP's que possam trabalhar como mestre de uma rede Modbus RTU (modo RTU possui maior densidade de dados comparado ao modo ASCII) e, meio físico RS232 (RS485 como uma variação do produto final) para que o módulo Modbus I/O possa ser utilizado como expansão de entradas e saídas digitais. O módulo Modbus I/O deve ser capaz de interfacear 12 entradas e 8 saídas digitais. Para a aplicação comercial deste projeto, as entradas devem ser opto-acopladas (isolação galvânica) sensíveis a níveis de tensão 24Vcc e 0V para níveis lógicos 1 e 0, respectivamente (como leitura de bóias de nível de um tanque); e as saídas devem ser capazes de acionar 8 cargas opto-isoladas (isolação galvânica) 24Vcc e corrente máxima de 100mA (por exemplo, acionar bobinas 24Vcc de contatores de bombas).

O sistema deve ter um *timeout* (tempo esgotado) de recepção, configurável como: inexistente, 1, 10 ou 60 segundos. Uma vez que o Modbus I/O permaneceu por um determinado tempo sem recepção de dado válido (toda mensagem recebida pelo escravo com seu endereço e *checksum* correto), o módulo desliga todas suas saídas digitais. Este tipo de prevenção para saídas, também conhecida como *Watch Dog* (cão de guarda), é muito importante para aplicações em que o mestre da rede, um CLP controlando uma planta industrial, por exemplo, necessite ligar/desligar uma esteira de acordo com a leitura de fins-de-curso. Caso ocorra uma falha na comunicação (ex.: rompimento do cabo de comunicação), a esteira será desligada evitando maiores danos ao sistema automatizado. Para a maioria dos casos de redes industriais, o mestre mantém a comunicação contínua de dados, requisitando

periodicamente informações de todos os escravos na rede (*polling*). Todavia, não são todos os casos, contudo, o módulo de Modbus I/O deve ter a possibilidade de não executar o *timeout* de recepção.

Definiu-se a implementação de dois módulos de Modbus I/O com as mesmas características e com microcontroladores distintos, ambos do fabricante Atmel; AT89C51RC (CISC de baixo custo) e ATmega64 (RISC com melhor desempenho). Para tal, ambos os módulos seão implementados em placas apropriadas com I/Os genéricas até que seja definido o melhor sistema a ser empregado em uma versão finalizada e comercial do projeto.

A maior diferença entre o projeto em questão e os módulos Modbus I/O comerciais encontra-se na disposição do hardware na comunicação com a rede Modbus. É proposto o uso de duas portas RS232 para interface com Mestre da rede; desta forma, é possível conectar diversos módulos de Modbus I/O em única porta RS232 do mestre (uma vez que a porta adicional funciona como um repetidor RS232), outrossim, mantendo a conectividade direta para um computador ou um CLP (ambos como mestre de uma rede Modbus).

A figura 6 mostra a topologia de rede utilizando o hardware proposto (pode ser aplicado dentro de um quadro de comando substituindo I/Os de um CLP).

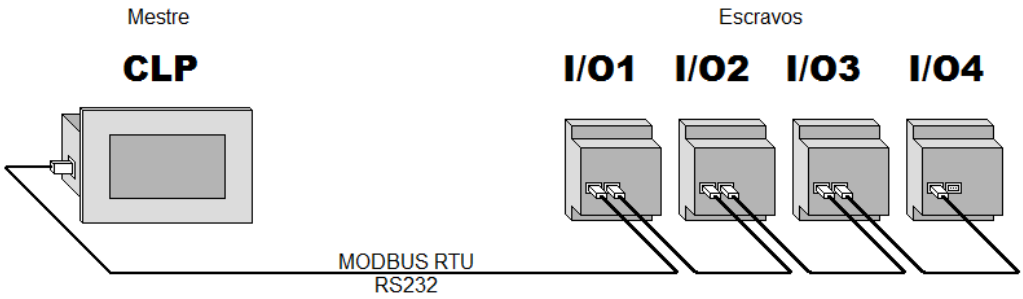


Figura 6 – Topologia de uma rede Modbbus RTU, via RS232

As figuras 7 e 8 mostram placas de I/O's dos 2 microcontroladores propostos para o projeto. A primeira PCI (placa de circuito impresso) na figura 7 foi desenvolvida na empresa Rotária do Brasil Ltda. para fins de depuração (verificação e validação do projeto durante o desenvolvimento) para projetos em geral e está sendo apresentada como objeto de estudo desta monografia; enquanto a segunda PCI foi desenvolvida dedicada a este projeto.

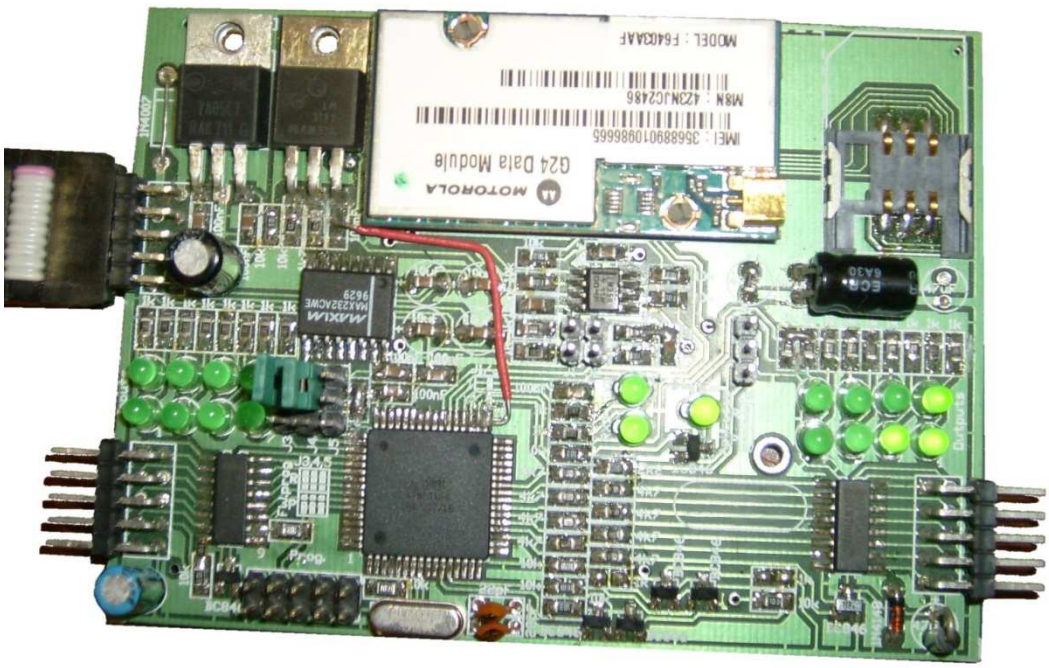


Figura 7 – Placa utilizada para depuração do projeto baseada no ATmega64

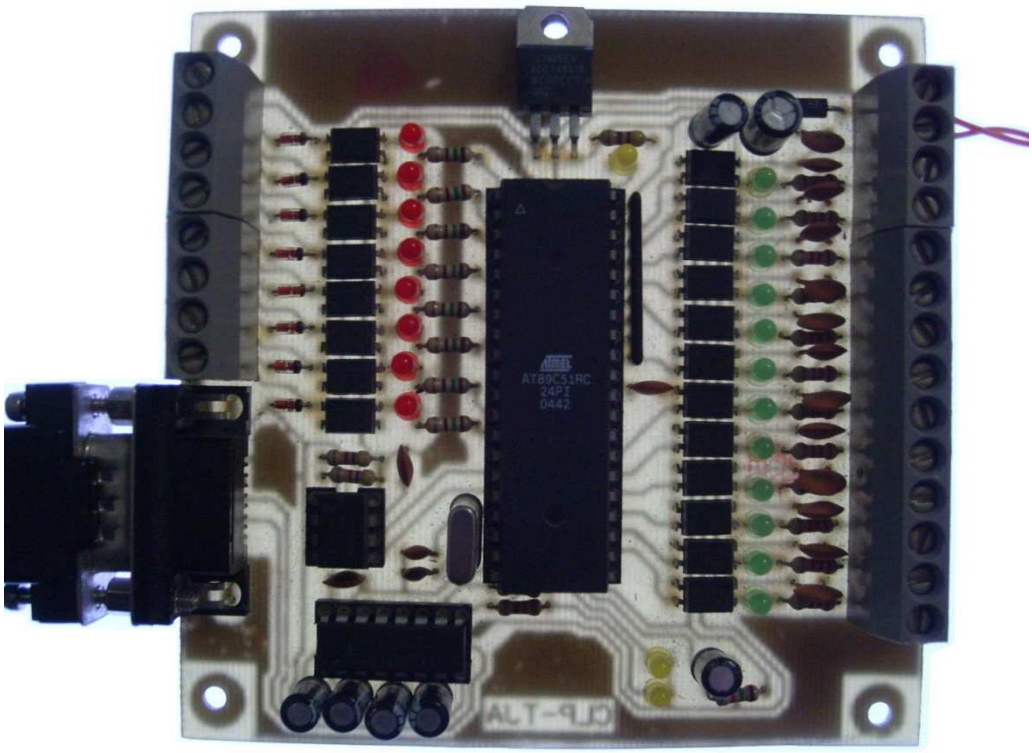


Figura 8 – Placa utilizada para depuração do projeto baseada no AT89C51RC

Objetivando velocidade e portabilidade para o desenvolvimento deste projeto, optou-se por programação em linguagem C. Mesmo que a linguagem C apresente menor desempenho de processamento que a programação em linguagem Assembly, tal "perda" de desempenho não seria significativa para a aplicação uma vez que o projeto dispõe de um processador dedicado apenas para atualizar dados para comunicação serial e suas I/O's.

Para o produto final, a taxa de transmissão deve ser programável em 4800bps, 9600bps, 19200bps ou 57600bps, 8bits de dado, sem paridade, 2 bits de para e sem controle de fluxo. O endereço da rede Modbus, tempo de *timeout* para saídas assim como o taxa de transmissão, são configuráveis através de *dip switch's* (micro-chaves) dispostas na PCI finalizada. A figura 9 mostra a estrutura do hardware proposto.

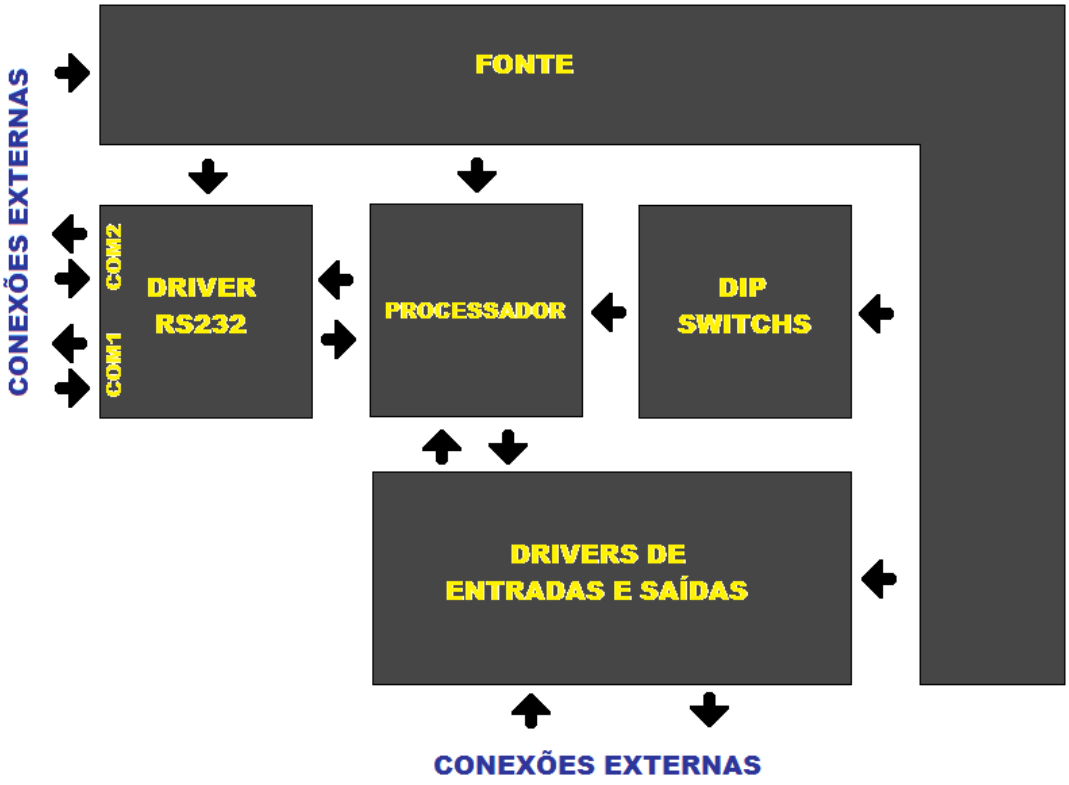


Figura 9 – Estrutura de hardware proposto

As funções Modbus implementadas são capazes de acessar completamente as entradas e saídas dispostas no módulo de Modbus I/O. Foram implementadas no projeto funções essenciais para o funcionamento do módulo: leitura de entradas (1 ou todas disponíveis), leitura de saídas (1 ou todas disponíveis) e escrever em 1 saída; outras funções não necessárias para esta aplicação foram implementadas para futura exploração deste protocolo para novos projetos. Foram especificadas os seguintes códigos (apresentados em hexadecimal) e suas respectivas funções Modbus :

- (0x01) *Read Coils* – Ler saídas digitais;
- (0x02*) Read Discrete Inputs* – Ler entradas digitais;
- (0x03) *Read Holding Registers* – mesmo não necessário para esta aplicação, executa gravação de dados em uma memória EEPROM;

- (0x04) *Read Input Registers* – mesmo não necessário para esta aplicação, executa leitura de dados em uma memória SRAM do sistema;
- (0x05) *Write Single Coil* – Escrever em uma saída digital;
- (0x06) *Write Single Register* – mesmo não necessário para esta aplicação, executa escrita de dados em uma memória EEPROM;

# 4 MATERIAL E MÉTODOS

Para que fosse possível testar o hardware/firmware foram utilizados softwares para este fim específico, Modbus Tester e Modbus Poll (adquiridos gratuitamente na internet com limitações de uso), assim como aplicações em rede Modbus com um CLP da marca Unitronics, modelo Vision290 (utilizado por disponibilidade na empresa a qual presto serviços na área de automação de estações de tratamento de água e esgoto, Rotária do Brasil Ltda.) e um software supervisório da marca Elipse, modelo E3 (também por disponibilidade na Rotária do Brasil Ltda.) rodando em um computador padrão IBM-PC.

Com o uso do Modbus Tester, foi possível depurar completamente o processo de comunicação e este software é livre. Com este aplicativo, é possível que um computador pessoal trabalhe como mestre de uma rede Modbus conectado a um escravo em uma das suas portas COM. Ambiente muito objetivo e de fácil uso. Este possui algumas limitações (como a conexão de apenas 1 escravo) e pequenos defeitos mas é gratuito e eficaz para testar todas funções básicas de leitura/escrita de entradas e saídas digitais, fundamentais para um módulo de Modbus I/O. A figura 10 mostra a tela principal de monitoramento das funções modbus pelo software Modbus Tester.

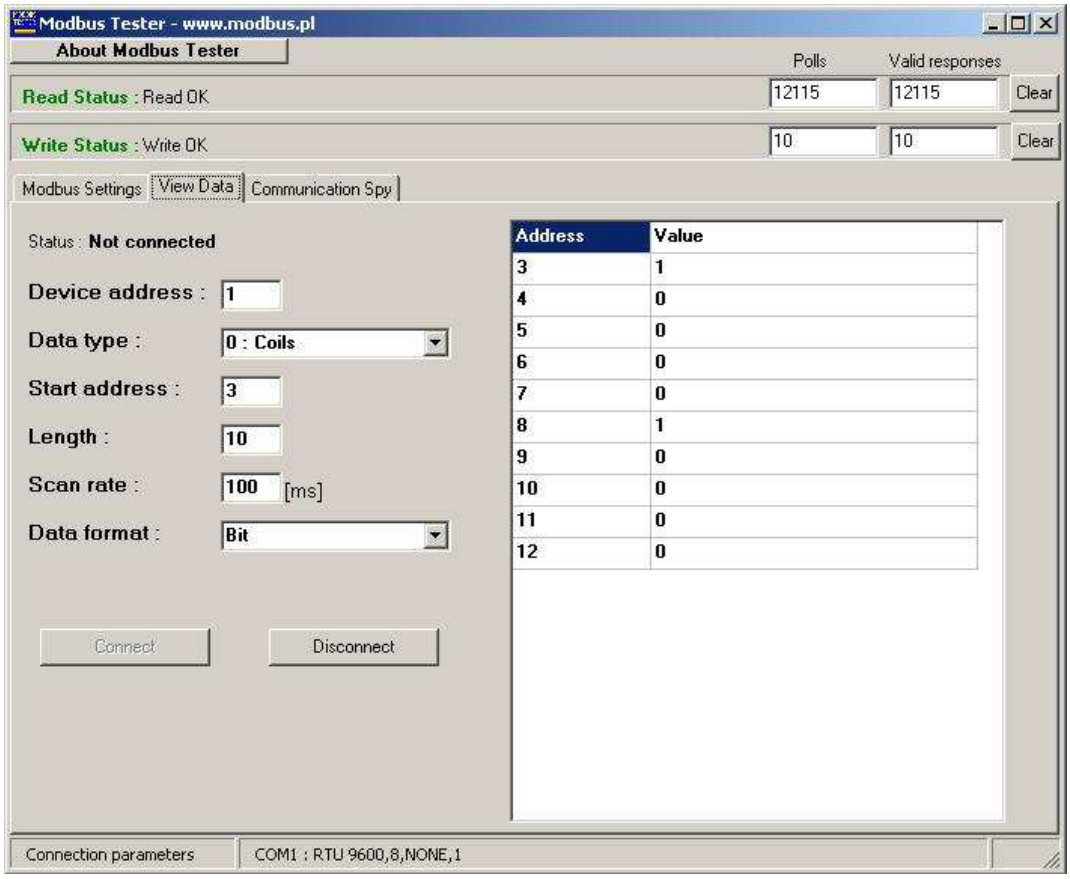


Figura10 – Modbus Tester

Objetivando utilizar a estrutura do módulo Modbus I/O para futuras implementações mais complexas que também utilizem interface Modbus RTU escravo, foi utilizado o Modbus Poll. Este aplicativo é capaz de simular perfeitamente uma rede Modbus com diversas funções como leitura/escrita em registros (voláteis / não voláteis). Este também pode ser mestre de uma rede Modbus permitindo diversos escravos simultâneos. Ótima ferramenta, mas é um software comercial. Foi utilizado como avaliação por 30 dias. A figura 11 mostra a tela principal de monitoramento das funções modbus, de diferentes escravos da rede, pelo software Modbus Poll.

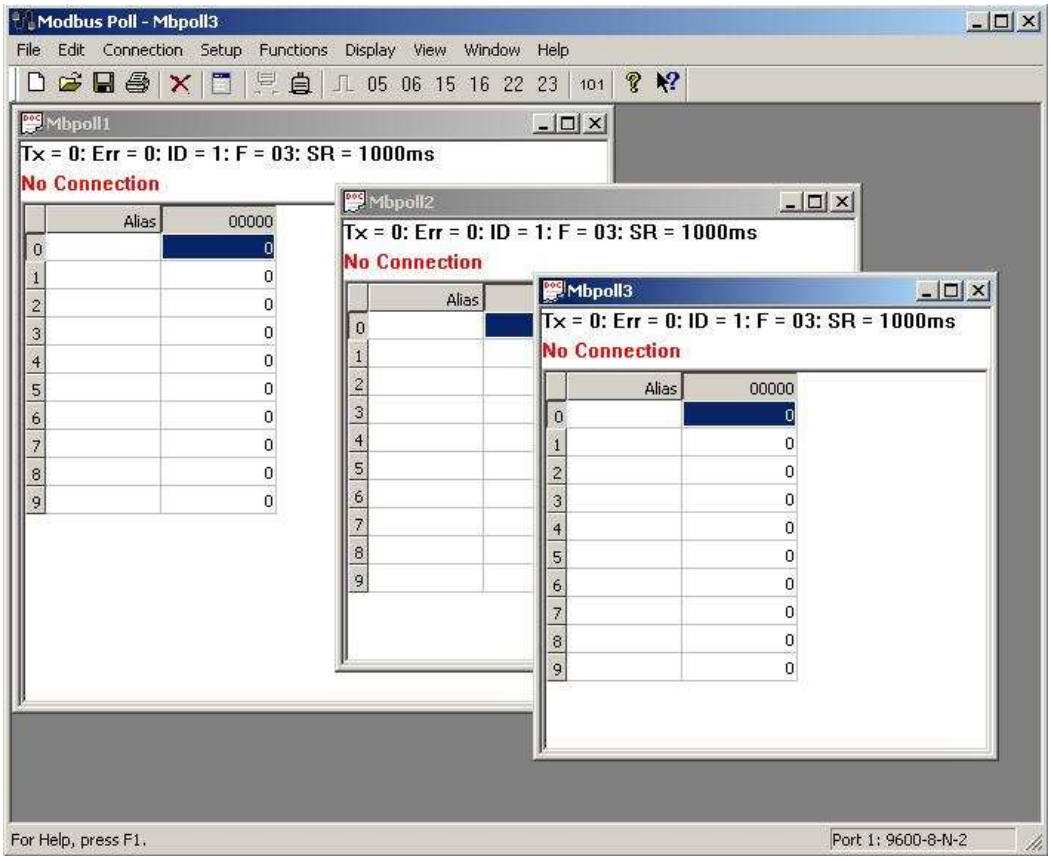


Figura 11 – Modbus Poll

# 5 RESULTADOS E DISCUSSÃO

## 5.1 Processador

O primeiro processador a ser implentado é o AT89C51RC (padrão MCS-51) com arquitetura CISC. Este possui 32kb de memória flash para código, 512bytes de memória RAM para dados voláteis, temporizadores 16bits, *Watch Dog Timer* (temporizador cão de guarda). Sua máxima performance é de 2MIPs com cristal externo de 24Mhz e, para o projeto, foi utilizado cristal de 22,1184Mhz para maior precisão do gerador de taxa de transmissão serial. Optou-se por implementar inicalmente neste microcontrolador devido a disponibilidade e conhecimento prévio desta arquitetura.

A segunda implementação foi com o microcontrolador ATmega64 (núcleo AVR). Sua arquitetura RISC executa 1 instrução a cada 1 *clock* (pulso do oscilador) atingindo até 8Mhz sem necessidade de uso de qualquer cristal/oscilador externo e, até 16MHz (utilizado no projeto) com cristal externo (embora alguns AVRs variantes normalmente podem utilizar até 20MHz). Possui 64Kb de memória flash para código, 4kb de memória RAM para dados voláteis, 2kb de memória EEPROM para dados não voláteis, diversos temporizadores 16bits, *Watch Dog Timer* (temporizador cão de guarda), além de conversores anlógico-digital (10 bits) e diversas saídas *Pulse Width Modulation* (modulação por largura de pulso) configuráveis até 16 bits. Com esta segunda opção de processador poderíamos facilmente implementar um Módulo de I/Os analógicas.

Com a devida avaliação do produto final ou com possível linha de produtos; este segundo microcontrolador torna-se mais interessante facilitando a logística da produção uma

vez que todas as variantes Modbus I/O podem estar baseadas em uma mesma tecnologia simplificando a aquisição dos microcontroladores e, podendo ser desenvolvido apenas um *layout* do hardware que comporte todas as possíveis variações de módulos analógicos e digitais.

Importante ressaltar que o núcleo MCS-51 não foi um fator limitante para as diferentes versões possíveis do Modbus I/O. A comparação anteriormente feita é estritamente entre um modelo específico de microcontrolador que utiliza núcleo MCS-51 com poucos periféricos internos em relação ao microcontrolador de núcleo AVR que possui mais periféricos internos. Um exemplo de microcontrolador de núcleo MCS-51 com equivalência ao do núcleo AVR utilizado, é o ADuC841 da *Analog Devices*. Este microcontrolador possui conversores analógicos/digitais de 12bits, conversores digitais/analógicos de 16bits e pode executar até 20MIPS com 20Mhz de cristal externo; portanto, este modelo de núcleo MCS-51 é superior em todos os quesitos relevantes para o desenvolvimento de um módulo Modbus I/O. O uso do modelo AT89C51RC e ATmega64 foram por disponibilidade e conhecimento prévio respectivamente.

## 5.2 Firmware

Com a proposta de implementar o módulo Modbus I/O em 2 controladores diferentes, optou-se o desenvolvido do firmware em liguagem C, devido a sua portabilidade.

Inicialmente, fez-se implementação simples e objetiva de um Modbus I/O para o microcontrolador AT89C51RC, pois já era uma tecnologia conhecida e havia disponibilidade deste microcontrolador, acelerando a depuração.

### 5.2.1 Implementação AT89C51RC

Para AT89C51RC, toda a funcionalidade do programa acontece na interrupção serial onde ocorre a confirmação da mensagem completa ao se contabilizar a recepção de exatos 8 bytes enviados pelo mestre da rede. O mestre sempre envia mensagens de 8 bytes que contém: 1 byte para endereço do escravo, 1 byte para função Modbus, 2 bytes para endereço inicial do dado, 2 bytes para número de dados requisitados e 2 bytes para o *checksum*.

Mesmo que cada mensagem enviada pelo mestre seja definida por 8 bytes, erros de comunicação podem ocorrer. Isto leva a necessidade de um *timeout* de recepção entre bytes que deve ser menor do que o tempo entre o envio de mensagens pelo mestre. Firmando-se na documentação do protocolo Modbus RTU, pode-se definir que este *timeout* de recepção deve ser maior do que 1,5bytes e menor do que 3,5bytes, ou seja, para uma taxa de transmissão de 9600bps, este deve ser maior do que 1,6ms e menor do que 3,6ms. Na prática, este tempo pode ter um valor razoavelmente alto (como 10ms), contanto que o mestre Modbus não envie pacotes mais rápidos do que o *timeout* entre bytes pode detectar, pois este se tornaria inútil e foi aplicado *timeout* de 3ms sem qualquer problema de comunicação nos testes executados.

Quando uma mensagem foi completamente recebida, o programa testa se o endereço de destinatário é o seu endereço de escravo, senão, ignora a mensagem. Após confirmação do endereço de escravo, o *checksum* é verificado, se resultado do cálculo efetuado pelo escravo não corresponder ao enviado pelo mestre, a mensagem é ignorada. Estando o endereço e o *checksum* de acordo, o programa analisa se a função e os endereços de dados são permitidos, caso não forem, o escravo responde com uma mensagem de erro correspondente, senão, executa a ação solicitada pelo mestre, seja leitura ou escrita, sempre enviando uma mensagem de resposta. Somente após estes passos a interrupção serial é liberada. O desenvolvimento foi feito com o compilador Franklin Proview32. A figura 12 mostra o ambiente de desenvolvimento no software Franklin Proview32.

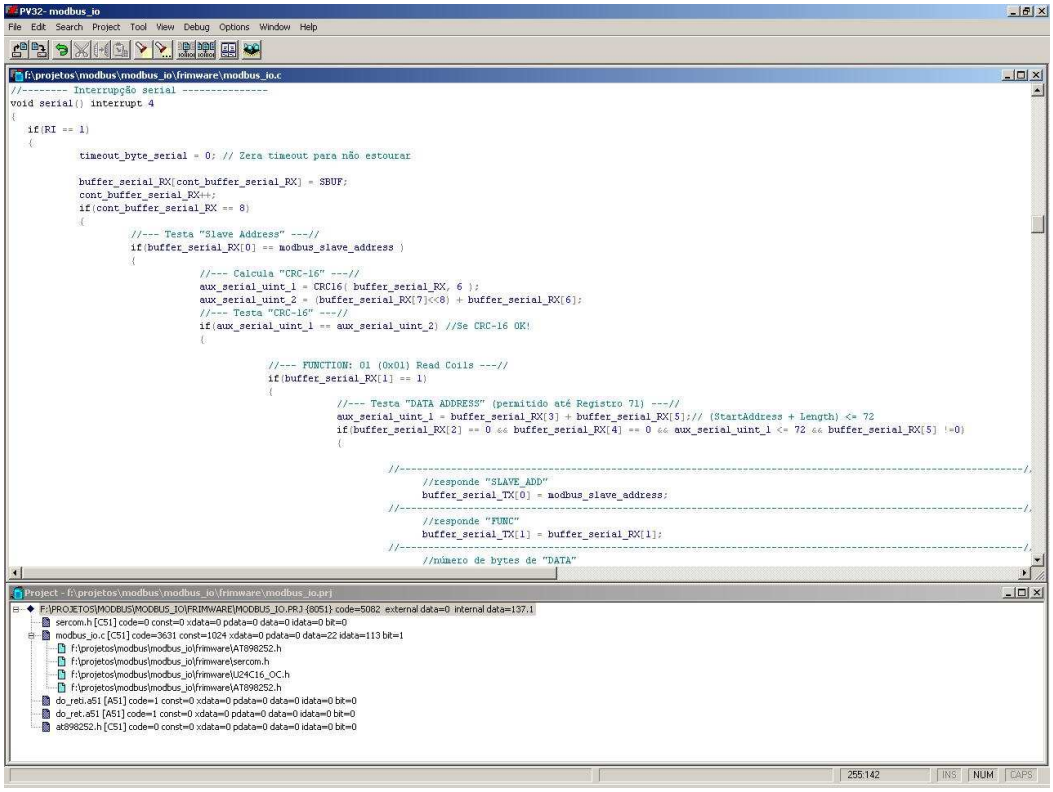


Figura 12 – Ambiente de desenvolvimento do firmware para AT89C51RC

A implementação do código escrito em C para MCS-51 não possui grandes diferenças ao implementado no núcleo AVR. Por se tratar não apenas de ANSI-C, fora necessárias alterações para as configurações de periféricos, assim como eventuais manipulações de registradores em interrupções.

### 5.2.2 Implementação com ATmega64

Devido à maior capacidade de processamento do ATmega64, foram feitas algumas mudanças na estrutura do firmware. A mais significativa delas é o uso de um *Preempetive Real-Time Operational System,* RTOS (Sistema Operacional em Tempo Real Preemptivo). Com o uso de um RTOS, o desenvolvimento e a depuração de projetos embarcados fica

facilitada, acelerando o desenvolvimento. Apesar de não ter sido uma tarefa simples desenvolver um sistema operacional (não foi desenvolvido especificamente para esta aplicação), este não havia sido rigorosamente testado e se mostrou robusto para esta aplicação.

O programa foi definido em 2 *tasks* (tarefas), uma de *polling* do *flag* (verificação periódica do registro de sinalização) de recepção de mensagem serial e outra para piscar o LED de estado atual do sistema. O tempo de troca de tarefas (*Time-Slic*), foi definido em 1ms e com um *Overhead* (ocupação do processador para gerenciamento do RTOS) de apenas 1,5% que é completamente satisfatório para a aplicação (o *Time-Slice* poderia ter sido definido em 10ms sem comprometer o projeto e baixando o *Overhead* para 0,15%).

A interrupção serial nunca se mantém com grande processamento, apenas armazena o byte recebido no *buffer RX* (um *array*) e a finalização da mensagem ocorre apenas por *timeout* entre bytes (definido em 3ms com taxa de transmissão de 9600bps, assim como na implementação com AT89C51RC).

Com apenas uma função escrita em C, pode-se ler a mensagem, responder apropriadamente e atualizar *buffers* de entrada e saída para que possam fisicamente serem atualizados, que é o propósito de um módulo Modbus I/O.

Devido a característica de agendamento das tarefas em um RTOS preemptivo, o envio da mensagem de resposta do escravo para o mestre é feita byte a byte, com o uso da interrupção de envio completo de byte diferente do implementado pelo AT89C51RC, que faz uso de um laço que aguarda a sinalização de envio de byte concluído para então enviar o próximo byte. Para um sistema operacional que pode suspender uma tarefa a qualquer momento, não é possível “confiar” que este sempre responderá com a mesma velocidade entre o envio de cada byte, tão pouco se pode confiar que o tempo entre envios de bytes seja em um intervalo menor ao máximo estabelecido pela documentação do protocolo em questão,

podendo ocorrer *timeout* entre bytes na recepção do mestre da rede Modbus. O uso de um sistema operacional não implica em melhoria no desempenho do sistema e sim em melhoria (rapidez) no desenvolvimento do projeto, devido ao uso de *tasks*, assim como facilitará o desenvolvimento de outros projetos baseados no protocolo Modbus RTU.

O uso de um sistema operacional implica em um uso maior de memória RAM pois, sistemas operacionais modernos utilizam uma s*tack* (pilha) para cada t*ask* alocando RAM dinamicamente. Tal estrutura de firmware se tornaria de difícil implementação em sistemas com pouca memória RAM e baixa velocidade (o que poderia não garantir a execução de tarefas em tempo real). Embora possível, optou-se pelo uso de um sistema operacional apenas para o microcontrolador ATmega64, pois este possui RAM suficiente para muitas tarefas simultâneas e velocidade suficiente para executa-las em tempo real. O desenvolvimento foi feito com o compilador AVR Studio e WinAVR. A figura 13 mostra o ambiente de desenvolvimento no software AVR Studio.

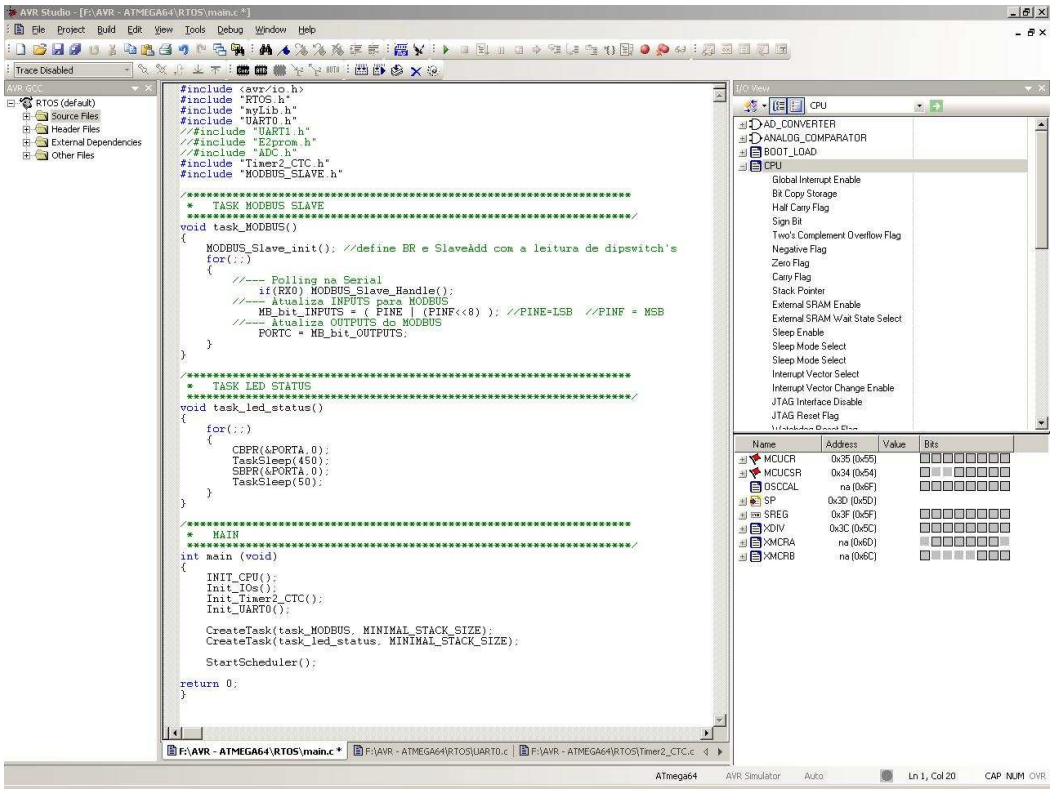


Figura 13 – Ambiente de desenvolvimento do Firmware para ATmega64

## 5.3 Aplicação

Para o produto final, a taxa de transmissão deve ser programável em 4800bps, 9600bps, 19200bps ou 57600bps, 8 bits de dados, sem paridade, 2 bits de para e sem controle de fluxo. As placas de circuito impresso utilizadas para os testes não possuem meios de interfacear tal configuração, desta forma, a taxa de transmissão foi definida em 9600bps.

Para os testes de comunicação, foi utilizado exaustivamente o software Modbus Tester. Com este foi possível testar todas as entradas e saídas digitais do módulo Modbus I/O. A figura 14 mostra o software Modbus Tester com um "contador" de mensagens efetuadas e um contador para mensagens respondidas com sucesso (pelo escravo, Modbus I/O).

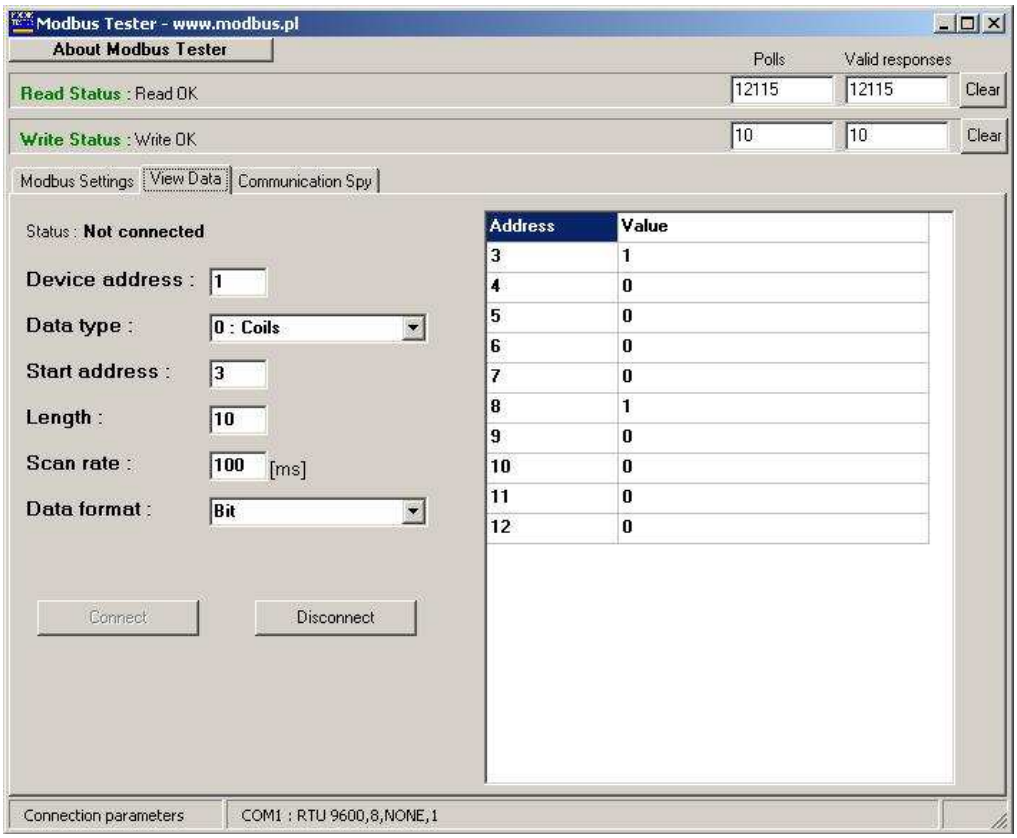


Figura 14 – Monitoramento das mensagens válidas no software Modbus Tester

Como teste de acionamento das saídas digitais, todas foram ligadas e desligadas, uma por vez utilizando a função Modbus 0x05. No software Modbus Tester existe a possibilidade de ligar e desligar múltiplas saídas digitais simultaneamente com a função Modbus 0x0F. Função esta que não foi aplicada neste projeto por não ser fundamental para o funcionamento absoluto do projeto, uma vez que a função Modbus 0x05 (função de escrever em uma saída) foi implementada.

Embora funções redundantes, 0x05 e 0x0F (ambas acionam as mesmas saídas digitais), o uso da função 0x0F pode tornar a comunicação mais rápida devido a escrita em diversas saídas simultaneamente em uma única mensagem Modbus. Com o uso da função 0x01, diferente da função 0x0F, pode-se aumentar o trafego de informações na rede e aumentar a necessidade de configurações e/ou memória necessária no mestre da rede, o qual terá que definir 8 instruções de envio de mensagens para escrita das bobinas que poderia ter sido implementado em apenas 1 instrução com a função 0x0F.

Para leitura das entradas (função 0x02) e leitura das saídas (função 0x01), foram feitos testes de variações do endereço inicial do registro a ser lido, assim como do número de valores desejados para as leituras.

O software Modbus Tester possui opção de monitorar a comunicação na aba chamada *Communication Spy* (espião da comunicação), onde é disposto em tempo real as mensagens enviadas pelo mestre e respondidas pelo escravo da rede (Modbus I/O a ser testado). A figura 15 mostra o software Modbus Tester na tela de monitoramento das mensagens trafegadas entre mestre e escravo, em uma rede Modbus.

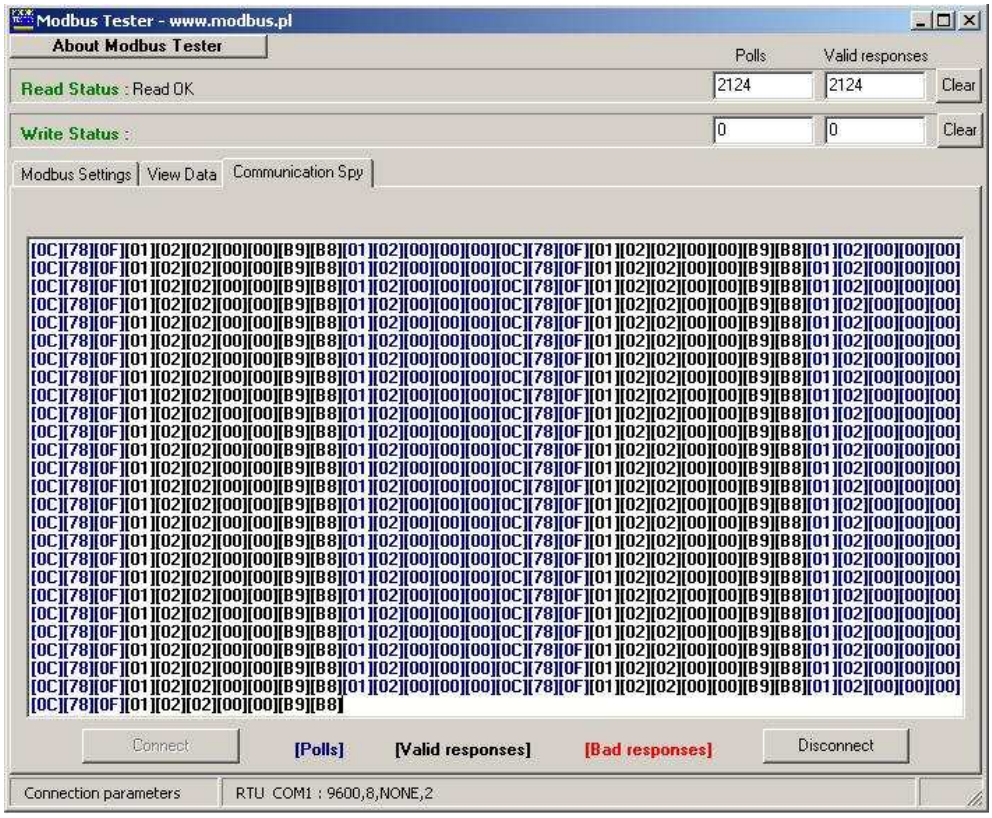


Figura 15 – Monitoramento das mensagens trafegadas no software Modbus Tester

Como teste, foi utilizado o módulo de Modbus I/O em rede Modbus RTU com um CLP da marca Unitronics como mestre da rede (este CLP foi utilizado devido a disponibilidade na empresa Rotária do Brasil Ltda., que emprestou gentilmente para este projeto). Para a programação do CLP, foram dispostas memórias de 1 bit relacionando-as com entradas e saídas remotas. A varredura do mestre da rede (CLP) sempre atualiza as entradas e posteriormente as saídas remotas periodicamente (*polling*), com isso, mesmo que as saídas digitais não mudem seu estado lógico, o mestre sempre escreve o valor já armazenado nas memórias bit para as saídas, acarretando maior tráfego na rede comparado com acionamento por demanda. Por exemplo, caso o CLP (mestre da rede) precise ligar uma determinada saída digital remota (no escravo da rede, Modbus I/O) no momento presente e, desligá-la após 1 hora, pode-se usar apenas 2 mensagens conforme ocorrida a demanda, uma para ligar e uma

para desligar. Entretanto, isto pode se tornar um risco para a confiabilidade do sistema uma vez que, por exemplo, o módulo Modbus I/O pode ter passado por falta de energia implicando no desligamento da saída que outrora o CLP, mestre da rede, tinha por verdade que estava ligada. A figura 16 mostra a aplicação do módulo Modbus I/O efetuando leitura de entradas digitais para um CLP como mestre da rede Modbus RTU.

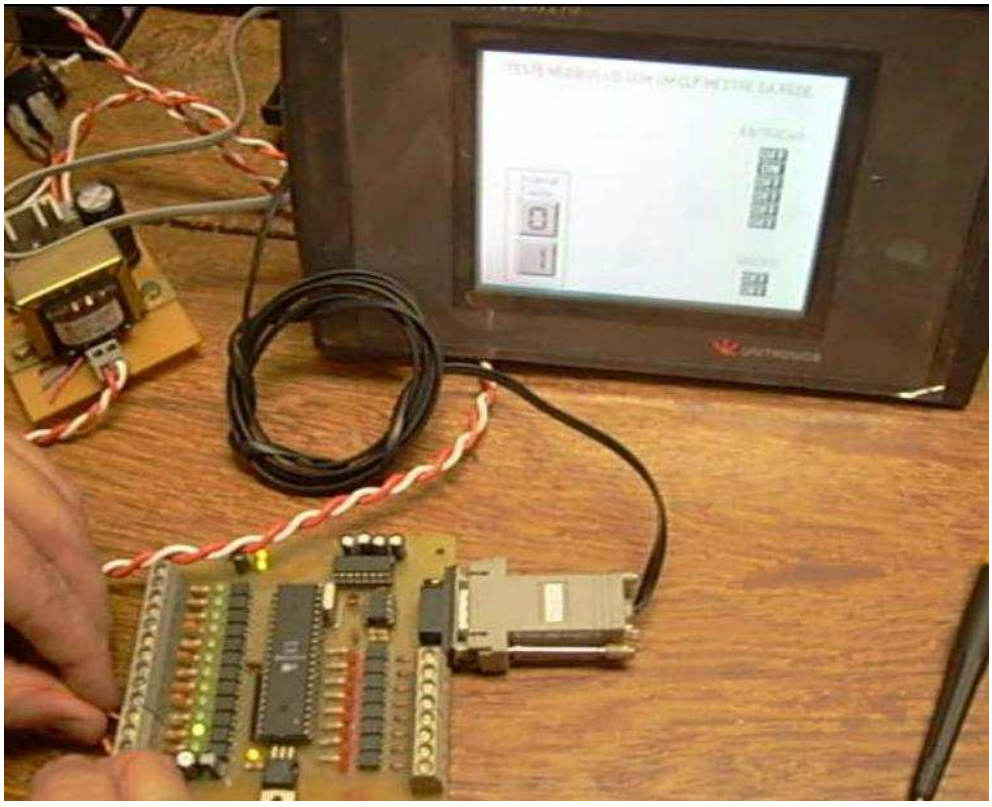

Figura 16 – Módulo Modbus I/O e um CLP da Unitronics (leitura de entradas)

Por se tratar da aplicação da função Modbus 0x01 para escrita em uma saída digital por mensagem (uma vez que a função 0x0F não foi implementada no projeto), houve a necessidade de configurar 8 blocos Modbus na programação do CLP ao invés de 1 bloco com a função 0x0F (escrever em múltiplas saídas), para que fosse possível escrever em 8 saídas digitais. Esta situação se agrava quando se tratar de, por exemplo, 32 saídas digitais

necessitando de 32 blocos da função Modbus 0x01 ou apenas 1 bloco da função 0x0F. A figuras 17  mostra a aplicação do módulo Modbus I/O efetuando a escrita de saídas digitais conforme acionamento na IHM (Interface Homem-Máquina) com tela sensível ao toque incorporado no CLP, que está disposto como mestre da rede Modbus RTU.

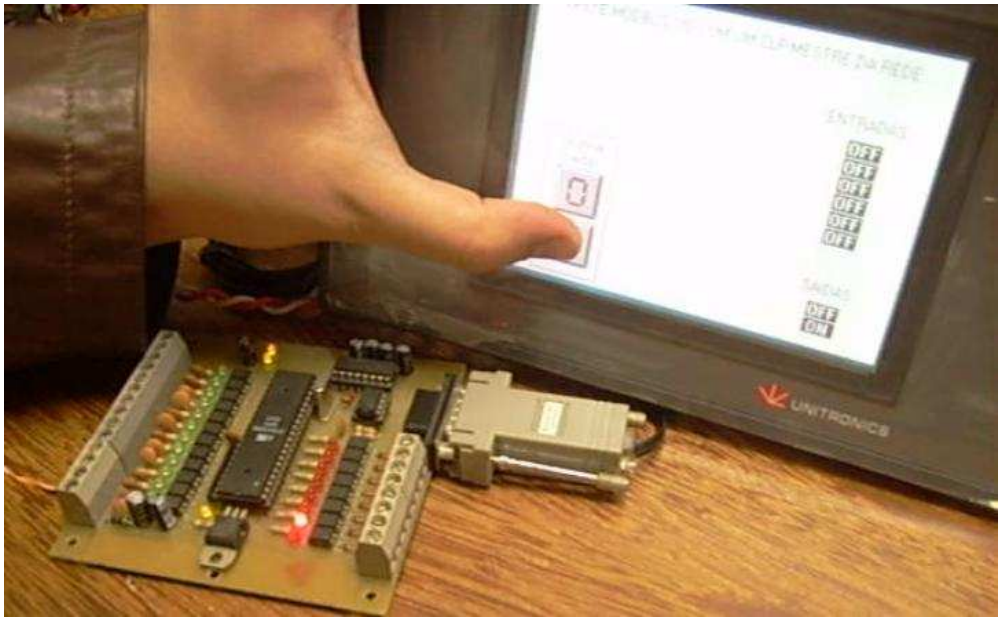


Figura 17 – Modulo Modbus I/O e um CLP da Unitronics (escrita da saída 1)

Uma aplicação usual para módulos de I/O é a conexão direta com sistemas supervisórios SCADA. Para efetuar testes neste tipo de sistema, foi utilizado o Software Elipse E3 na versão de demonstração. O Software foi programado apropriadamente para monitorar e acionar todas as entradas e saídas dispostas no módulo Modbus I/O proposto. A figura 18 mostra a tela de teste de driver de comunicação (configurado como Modbus RTU) do software Elipse E3 em execução como mestre da rede.

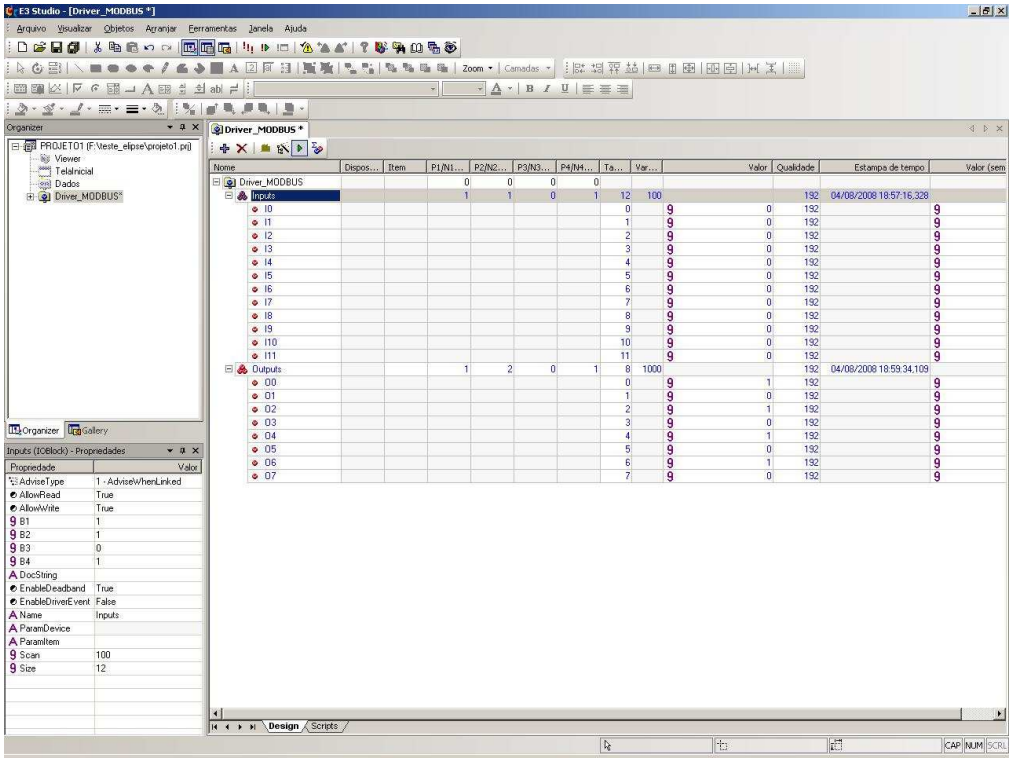


Figura 18 – Software Elipse E3 comunicando com Modbus I/O

As figuras 19 e 20, mostram a tela de execução configurada apropriadamente para efetuar leituras de entradas e saídas digitaiss assim como o acionamento discreto das saídas por um “click” nos ícones associados a estas.

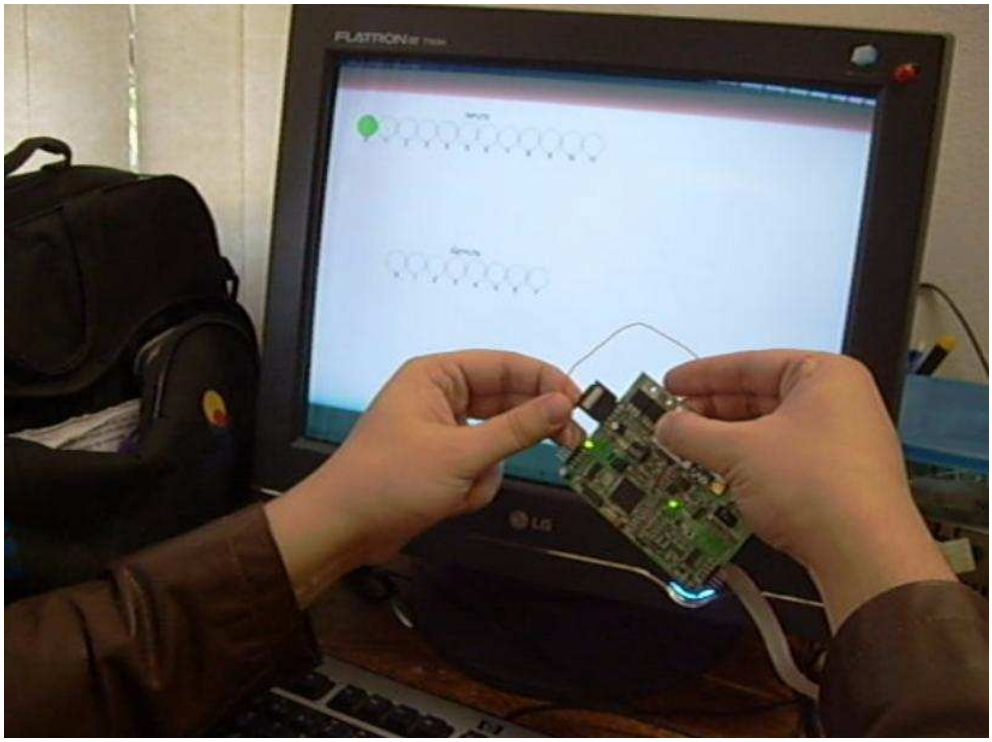

Figura 19 – Modulo Modbus I/O e software Elipse E3 (teste entradas)

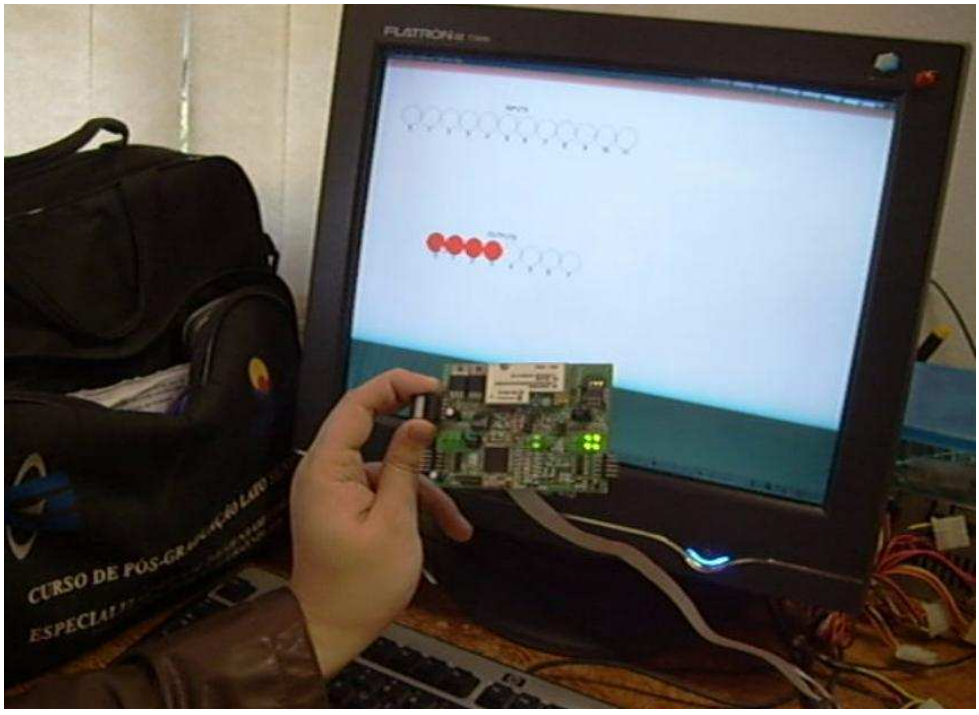

Figura 20 – Modulo Modbus I/O e software Elipse E3 (teste entradas)

Caso a comunicação entre mestre escravo seja interrompida por algum problema físico (rompimento do cabo de comunicação) e/ou falta de energia para manter o mestre da rede ativo, usualmente, módulos de I/O's (escravos da rede) disponibilizam um *timeout* por falta de recepção de mensagens válidas (toda mensagem recebida pelo escravo com seu endereço e *checksum* correto) oriundas do mestre.

Esta configuração é muito importante para aplicações em que o mestre da rede, um CLP controlando uma planta industrial, por exemplo, necessite ligar/desligar uma esteira de acordo com a leitura de fins-de-curso. Caso ocorra uma falha na comunicação (ex.: rompimento do cabo de comunicação), a esteira será desligada evitando maiores danos ao sistema automatizado. Para a maioria dos casos de redes industriais, o mestre mantém a comunicação contínua de dados, requisitando periodicamente informações de todos os escravos na rede (*polling*). Todavia, não são todos os casos, contudo, o módulo de Modbus I/O deve ter a possibilidade de não executar o *timeout* de recepção.

Devido a ausência de *dip switch's* (micro-chaves) em ambas as placas de circuito impresso usadas para os testes deste projeto, foi fixado um valor de 10 segundos para esta configuração. O produto final será configurável como: inexistente, 1, 10 ou 60 segundos.

Para testar esta funcionalidade, foram executados 2 testes diferentes que simulam a danificação do cabo de comunicação e a falta de energia do mestre da rede. Inicialmente foram acionadas todas as 8 saídas digitas presentes no projeto com o uso do software Elipse E3, devidamente configurado como mestre da rede e posteriormente retirou-se o cabo de comunicação; após 10 segundos todas saídas foram desligadas pelo próprio módulo Modbus I/O. O cabo de comunicação foi re-conectado e, novamente foram acionadas todas as 8 saídas pelo mestre da rede. O software foi finalizado abruptamente como em uma falta de energia do mestre; passados 10 segundos do ocorrido, todas as saídas digitais foram desligadas pelo

módulo Modbus I/O. Este procedimento foi realizado para as duas plataformas abordadas de microcontroladores abordadas neste projeto.

# 6 CONCLUSÕES E RECOMENDAÇÕES

O projeto alcançou os objetivos propostos para as funcionalidades de um módulo Modbus I/O depurado nas placas de circuito impresso adotadas para os microcontroladores AT90C51RC e ATmega64, entretanto, o produto final ainda não foi desenvolvido. Será feita uma avaliação mais específica sobre as diferentes variações de produto final para definir com maior cuidado qual microcontrolador aplicar como padrão de uma linha de módulos Modbus I/O (ou até mesmo optar por 2 tecnologias distintas, cada qual para uma determinada linha de produtos). Ambos microcontroladores foram satisfatórios e desempenharam o proposto com êxito; interfacear 12 entradas e 8 saídas digitais sobre o protocolo de comunicação Modbus RTU.

A definição do uso do AT89C51RC foi por conhecimento prévio da tecnologia enquanto a aplicação com o ATmega64 foi selecionada após análise de custo/benefício. Apesar do microcontrolador AT89C51RC ter custo mais baixo do que o ATmega64, ele não é capaz de interfacear I/O's analógicas sem adição de hardware externo. Com o acréscimo de circuitos integrados (CI's) no projeto, este necessitaria de maior complexidade do layout do circuito impresso (maior densidade) e poderia aumentar a necessidade de área na placa de circuito impresso do produto final, conseqüentemente, aumentando o custo.

Uma possível alteração do Módulo para a versão comercial é a configuração da taxa de transmissão, endereço de rede, *timeout* (tempo esgotado) de recepção de mensagem válida

(toda mensagem recebida pelo escravo com seu endereço e *checksum* correto), através de uma conexão serial entre o Modbus I/O e um computador pessoal com porta COM disponível externamente. Para que tal ocorra, basta definir uma mensagem Modbus (com uma função apropriada, a ser definida), que passe o módulo para o "modo de programação". A programação pode então proceder por um software aplicativo dedicado ou, devido aos poucos ajustes, através de algum terminal serial gratuito de tantos disponíveis na internet. Este tipo de mudança requer uma adição de hardware no caso da implementação com o microcontrolador AT89C51RC. Esta mudança necessita do uso de uma memória não volátil que, por questão de tamanho de hardware e custo, pode-se utilizar uma memória serial como 24C16, com 2kbytes de E2prom com protocolo de comunicação I2C implementado por software. Com o uso do Atmega64, a necessidade de memória não volátil não implica em adição de periféricos ao hardware, pois o mesmo já possui 2kbytes de E2prom interna; e, mesmo que isto fosse necessário, a comunicação com o protocolo I2C pode ser implementada por hardware neste microcontrolador.

O uso de um sistema operacional para a implementação com o ATmega64 não implica em melhoria no desempenho do sistema e sim em melhoria (rapidez) no desenvolvimento do projeto, devido ao uso de *tasks (tarefas)*, assim como facilitará o desenvolvimento de outros projetos baseados no protocolo Modbus RTU.

O uso de um sistema operacional implica em um uso maior de memória RAM pois, usualmente, sistemas operacionais modernos utilizam uma s*tack* (pilha) para cada t*ask* alocando RAM dinamicamente. Tal estrutura de firmware se tornaria de difícil implementação em sistemas com pouca memória RAM e baixa velocidade (o que poderia não garantir a execução de tarefas em tempo real). Embora possível, optou-se do uso de um sistema operacional apenas para o microcontrolador ATmega64, pois este possui RAM

suficiente para muitas tarefas simultâneas e velocidade suficiente para que execute-as em tempo real.

Usualmente, fabricantes de módulos de Modbus I/O possuem linhas de produtos com I/O's digitais bidirecionais (normalmente sem isolação galvânica) como o modelo D1000 da marca GGH CORP.; módulos exclusivos de entradas ou saídas, como os modelos TRP-C26 e TRP-C24 da marca Trycom Technology; e módulos com entradas e saídas, digitais e/ou analógicas, como o modelo KTA-215 da Ocean Controls.

Os módulos de I/O comerciais supracitados utilizam Modbus RTU sobre os meios físicos RS232, RS422 e RS485. Alguns destes produtos comerciais possuem a possibilidade de configurar qual meio físico utilizar e outros são específicos para um determinado padrão. O módulo Modbus I/O proposto neste projeto (para a versão comercial) utilizar 2 portas RS232 para interface com Mestre da rede; desta forma, é possível conectar diversos módulos de Modbus I/O em única porta RS232 do mestre (uma vez que a porta adicional funciona como um repetidor RS232), outrossim, mantendo a conectividade direta para um computador ou um CLP (ambos como mestre de uma rede Modbus).

A especificação de um módulo Modbus I/O com 12 entradas e 8 saídas digitais foi a escolha para um módulo que pudesse atender as necessidades de automação em saneamento (estações de tratamento de água e esgoto) para a empresa a qual presto serviços na área de automação, Rotária do Brasil Ltda.

# 7 REFERÊNCIAS BIBLIOGRÁFICAS[2]


ATMEL, datasheet AT89C51RC. On-line: http://www.atmel.com/dyn/resources/prod_documents/doc1920.pdf, acessado em 25 de fevereiro de 2008.

ATMEL, datasheet ATmega64. Revision N, Update 05/08 On-line: http://www.atmel.com/dyn/resources/prod_documents/doc2490.pdf, acessado em 12 de maio de 2008.

AVR Freaks, The nr.1 AVR community. On-line: http://www.avrfreaks.net/, acessado em 10 abril de 2008.

AVR STUDIO 4. Versão 4.14, Build 589. 2008.

CLARKE, Gordon R., REYNDERS, Deon, WRIGHT, Edwin. Practical Modern SCADA Protocols: DNP3, 60870.5 and Related Systems. Editora Newnes. 2004.

MACKAY, Steve; WRIGHT, Edwin; REYNDERS, Deon; PARK, John. Industrial Data Networks: Design, Installation and Troubleshooting. Editora: Elsevier. 2004.

MARINHO, José Edson dos S.; MARINHO, Ednaldo dos S. Revista Saber Eletrônica Especial: Mini-curso de microcontrolador. São Paulo: Saber, n.2, Jan 2002

MODBUS Application Protocol Specification. V1.1b. Modbus-IDA, 2006.

MODBUS over Serial Line. Specification and Implementation Guide. V1.02. Modbus-IDA, 2006.

MODICON Modbus Protocol Reference Guide. PI-MBUS-300 Rev. J. Modicon, Inc. 1996.

MODBUS-IDA, The Architecture for distributed automation. On-line: http://www.modbus.org/, acessado em 10 abril de 2008.


[2] Baseado na NBR 6023: 2002 da ABNT.

MODBUS Poll. Modbus Tools. Versão 4.3.3, Build 298. 2007.

MODBUS Tester. Versão 0.3 beta. 2004.

NICOLOSI, Denys E. G. Laboratório de microcontroladores família 8051. São Paulo: Érica, 2002.

PROVIEW 32, Developers Version. Franklin Software, Inc. Versão 3.3.4, Build 8.63. 1996.

SCHUNK, Leonardo Marcilio e LUPPI, Aldo. Microcontroladores AVR. Teoria e Aplicações Práticas. Editora Érica, 2004.

WINAVR GCC. Versão 4.3.0, Build 8.63. 2008.

YAGHMOUR, Karim. Building Embedded Linux Systems: Concepts, Techniques, Tricks, and Traps. Editora O'Reilly. 2003.