PROTOCOLO MODBUS IMPLEMENTADO EM MICROCONTROLADOR PIC PARA AUTOMATIZAÇÃO DE UM CONTROLE DE TEMPERATURA

Daniel Koslopp[1]; Talita Tobias Carneiro[1]; Flávio Trojan[1]; Murilo Leme[1]
talitatobias@outlook.com; koslopp@gmail.com; muriloleme@utfpr.edu.br; trojan@utfpr.edu.br;
[1]Universidade Tecnológica Federal do Paraná – UTFPR – Ponta Grossa – Brasil

Resumo

O objetivo deste documento é apresentar o desenvolvimento da implementação do protocolo MODBUS no microcontrolador PIC18F2550 para controle de temperatura em um tanque de um processo, onde o enfoque é dado à comunicação e não a técnica de controle. Utilizou-se o software ScadaBR como mestre e o referido microcontrolador como escravo. A metodologia empregada constitui-se em descrever os passos tomados para o desenvolvimento da automatização de um controle de temperatura, a partir da implementação do protocolo MODBUS Escravo no microcontrolador PIC. Os resultados são apresentados sob perspectiva do ScadaBR, mostrando a leitura de dados no sistema supervisório.

Palavras-chave: *Automatização; Microcontrolador; Protocolo MODBUS.*

1. Introdução

A automatização é um processo de evolução tecnológica, uma vez que auxilia o ser humano na execução de tarefas repetitivas, além do aumento da qualidade e o menor custo de produção.

Desta forma a maioria dos processos industriais precisam necessariamente ser controlados e supervisionados. Segundo Moraes (2001), a automação exige implantação de sistemas interligados, e é nesta etapa que a comunicação entre os equipamentos desempenha papel fundamental para a automatização de um processo.

Há diferentes tipos de protocolos para comunicação industrial, a escolha da utilização de um protocolo depende de variáveis como tipo de planta industrial, cabeamento, custo, entre outros.

O protocolo de comunicação industrial MODBUS é uma estrutura de mensagens desenvolvida em 1979 pela Modicon, Inc. Hoje pertence a Schneider Electric que transferiu os direitos do protocolo para a MODBUS *Organization* em 2004, desde então a utilização do protocolo é livre de taxas de licenciamento.

A integração dos vários processos de uma planta geralmente é feita por um sistema supervisório, que pode atuar no controle da mesma ou simplesmente realizar a leitura das diversas variáveis do processo. Neste último caso, o controle é chamado descentralizado.

A proposta deste artigo é a implementação do protocolo de rede MODBUS Escravo (MODBUS, 2002) em um microcontrolador PIC18F2550 (MICROCHIP, 2009) para a leitura das variáveis de um processo de controle de temperatura de um tanque através de um sistema supervisório (ScadaBR) com licença gratuita atuando como mestre MODBUS.

2. Metodologia

Três passos foram tomados para a construção da metodologia do presente documento. O primeiro deles consistiu no desenvolvimento do *software* do protocolo MODBUS Escravo para o microcontrolador (MODICON, 1996). A segunda etapa constituiu-se na implementação do sistema supervisório ScadaBR como Mestre MODBUS (SCADABR, 2010). Na etapa final a comunicação física entre o Mestre e o Escravo MODBUS é realizada, bem como a implementação de modo simplista do controle de temperatura.

Para a primeira parte do desenvolvimento do presente artigo, é necessária a definição de aspectos importantes para o *software*. O embasamento teórico em questão foi realizado a partir do documento oficial disponibilizado pela Schneider Electric. Neste documento serão apresentados somente os tópicos principais requeridos.

Assim, é imprescindível destacar a importância do embasamento teórico para este projeto. Como o escravo (implementado pelos autores) se comunicará com o mestre (programa desenvolvido por outra iniciativa) eles devem "falar" necessariamente a mesma "língua", ou seja, exatamente o mesmo protocolo.

O protocolo MODBUS define uma comunicação Mestre-Escravo entre os dispositivos conectados em diferentes topologias e a diferentes tipos de barramentos.

Esta técnica Mestre-Escravo permite apenas que o dispositivo mestre inicie o processo de mensagens com até 247 escravos em um nó. Os outros dispositivos da rede, os escravos, respondem ao mestre ou realizam a ação solicitada. Quando um erro é gerado, há o tratamento

do mesmo, possibilitando inclusive que o mestre tenha acesso ao tipo de erro ocorrido.

Há possibilidade de implementação deste protocolo de duas formas. A primeira de modo serial (principalmente com os meios físicos EIA-485 e EIA-232), a segunda com o Ethernet™. No modo serial a interface mais utilizada é a RS485 com dois fios, podendo também ser implementada com RS485 de quatro fios. Em distâncias curtas, a interface serial RS232 é sugerida.

No protocolo MODBUS o escravo não pode transmitir dados sem a solicitação do mestre**,** nem se comunicar com outros escravos. Apenas uma transação MODBUS é inicializada pelo mestre de cada vez.

O pedido do mestre pode ser feito de duas formas, no modo *unicast* (específico para um escravo que recebe a solicitação e envia sua resposta para o mestre) ou no modo *broadcast* (mandado para todos os escravos, onde não há retorno para o mestre).

O endereço do quadro MODBUS tanto no pedido do mestre quanto na resposta do escravo deverá necessariamente ser preenchido com o endereço do escravo, que deve ser único em toda a rede.

O quadro completo deste protocolo serial pode ser visto na figura 1. O campo *address* deve ser preenchido de acordo com o escravo, em seguida o campo *function code* deve ser escrito com o número da função MODBUS a ser requerida pelo mestre para o escravo, as funções serão abordadas posteriormente neste trabalho. O campo *data* deverá conter no pedido do mestre parâmetros sobre a função requerida e na resposta o que foi feito pelo escravo. O último campo é o de checagem de erro, ele pode ser CRC ou LRC dependendo do modo de comunicação serial utilizado, que será visto a seguir.

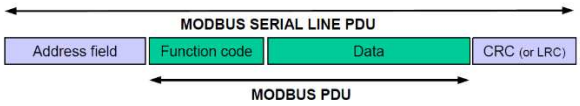

Figura 1. Formato do quadro MODBUS. (MODBUS, 2002)

Há duas definições para transmissão no protocolo serial MODBUS, sendo elas o RTU (*Remote Terminal Unit*) e o ASCII.

O modo RTU utiliza o sistema de codificação binária e contém 11 bits, sendo 1 *start bit*, 8 *bits* de dados (LSB enviados primeiro), 1 *bit* de checagem de paridade (não é obrigatório) e 1 *stop bit*.

No modo RTU o campo de checagem deve ser dado por CRC (*Cyclic Redundancy Check*). A figura 2 apresenta a descrição do quadro neste modo.

Slave Address	Function Code	Data	CRC
1 byte	1 byte	0 up to 252 byte(s)	2 bytes CRC Low CRC Hi

Figura 2. Quadro do modo RTU, não podendo exceder 256 bytes. (MODBUS, 2002)

A transmissão da mensagem no modo RTU deve ser feita de modo que os dispositivos (tanto o mestre quanto o escravo) identifiquem o começo ou o final da mensagem.

O inicio de um quadro é dado pelo primeiro *start bit*, já o fim de um quadro é definido por um intervalo de tempo em que a linha fica ociosa. Esses tempos devem ser identificados pelos dispositivos, onde o intervalo de cada *byte* não deve demorar mais do que 1,5 char para os dispositivos que estão se comunicando ou 3,5 char para os que estão aguardando o fim da transmissão (char é o tempo necessário para transmissão de um *byte* completo e varia com a velocidade de comunicação).

Caso haja um intervalo maior que 1,5 char, o receptor assumirá a transmissão encerrada, considerando os dois últimos *bytes* sendo o CRC. Caso a transmissão não tiver sido completada, um erro será detectado e enviado para o mestre.

A comunicação Mestre-Escravo pode ser descrita por meio de diagramas de estado, que representam o comportamento dos dispositivos.

No caso do mestre, quando sem envios e sem recebimentos, o mesmo permanece no estado *Idle*, ao enviar um pedido em *broadcast*, aguarda um tempo e então volta ao estado inicial, visto que nenhum escravo deve responder neste modo.

Se o mestre inicia a comunicação com um escravo específico, ele espera pela resposta. Porém, se o endereço do quadro recebido for de um escravo inesperado, um erro deverá ser gerado. Se durante esta espera o escravo não responder, outro erro ocorrerá. Caso tudo ocorra como o esperado, o escravo responde. Nesta etapa um outro erro de quadro pode ser gerado (quadro corrompido), caso contrário o processo é finalizado e o mestre volta ao estado inicial.

O diagrama de estado do escravo é descrito da mesma forma que o do mestre. O estado *Idle* também se aplica ao escravo. Se o mestre faz um pedido, o escravo faz a recepção e o checa. Dois erros podem acontecer nesta etapa, pode haver erro no CRC em que o escravo volta ao estado inicial ou então pode acontecer um erro nos dados do pedido do mestre, este último caso é acompanhado de uma formulação de um quadro de erro ocorrido para o mestre.

Após a definição do funcionamento do protocolo MODBUS, desenvolveu-se o software para implementação no microcontrolador.

Como o objetivo deste projeto é estabelecer a leitura de um controle de temperatura pelo sistema supervisório, juntamente com o *software* MODBUS Escravo, desenvolveu-se a lógica do microcontrolador, utilizando literatura adequada (PEREIRA, 2003), para controle de temperatura no tanque através do acionamento de um resistor.

O fluxograma do *software* MODBUS Escravo com o controle de temperatura pode ser apresentado em três subprogramas. Sendo estes a rotina principal, a interrupção serial e a interrupção do *timer 3*, que podem ser visualizados nas figuras de 3, 4 e 5 respectivamente.

A rotina principal é o ponto de partida para o *software*, sendo assim, é nela que são declaradas as variáveis e configurados os periféricos. Neste caso configurou-se a porta serial para a comunicação

MODBUS e o conversor A/D para leitura da temperatura.

Após os procedimentos iniciais o microcontrolador realiza a leitura da temperatura e atua no resistor, responsável pelo aquecimento do tanque, conforme o fluxograma.

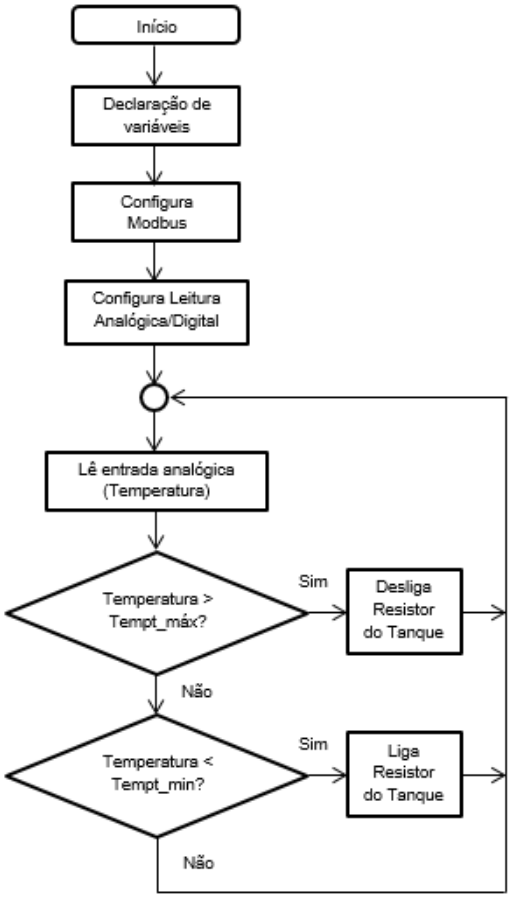

Figura 3. Rotina principal

A rotina principal continua até que o supervisório (mestre) realize um pedido através da porta serial, que gerará a interrupção serial e inicia sua respectiva rotina, representada na figura 4.

O primeiro passo realizado pelo *software* ao receber um *byte* é verificar se a mensagem é para o seu endereço. Caso não seja, o valor recebido é ignorado e configura-se o *timer 3* para temporizar o tempo de 3,5 char. Se a mensagem for para o seu endereço, o escravo é configurado para receber uma sequência de *bytes* (quadro), onde cada *byte* é salvo em sua respetiva posição de memória e o *timer 3* é configurado para temporizar 1,5 char.

Após o fim da recepção de um *byte* o programa retorna à rotina principal aguardando um novo *byte* ou a interrupção do *timer 3*, o que caracteriza o fim do quadro.

Assim, existem somente duas possibilidades para o *software* entre na rotina de interrupção do *timer 3* (figura 5). A primeira é devido a temporização de 3,5 char (quando o quadro não era para este endereço), onde a única ação a ser tomada é preparar o escravo para a recepção de um novo quadro.

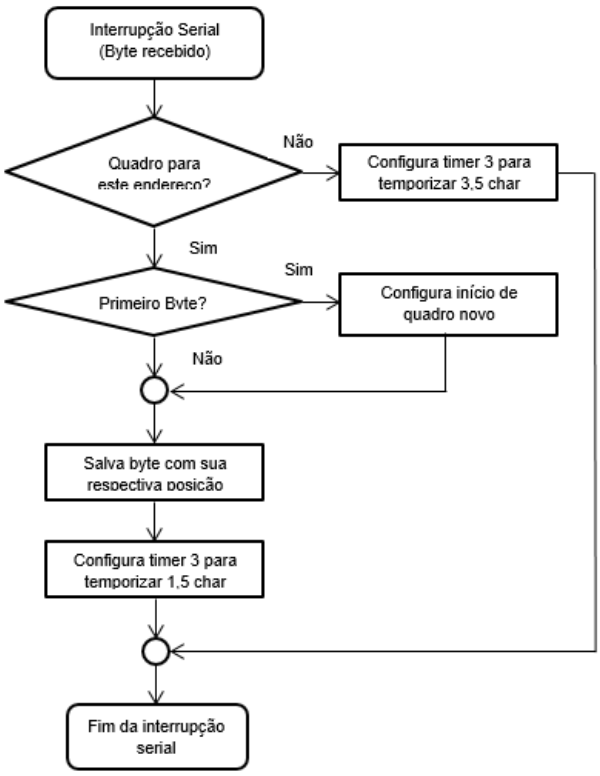

Figura 4. Rotina de interrupção serial.

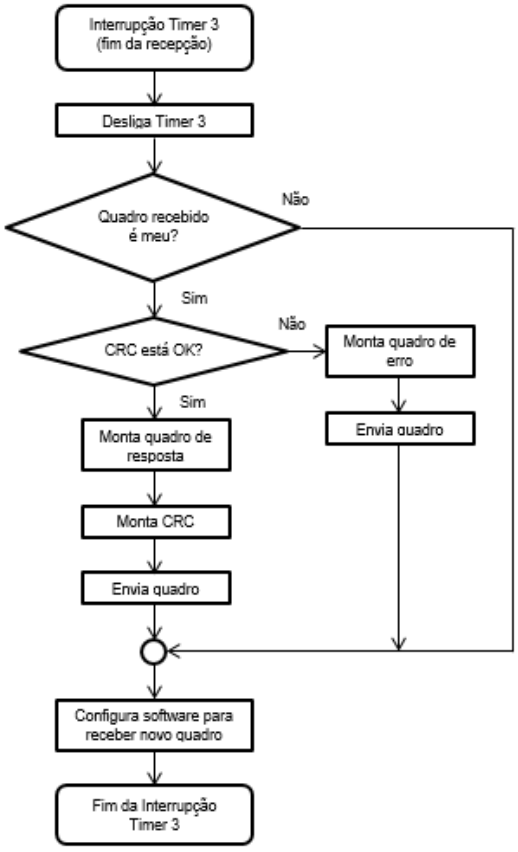

Figura 5. Rotina de interrupção timer 3

Na segunda condição, quando a mensagem recebida é do endereço em questão, primeiro é verificada a integridade do quadro pelo CRC, caso algum erro foi detectado o mesmo deve ser indicado ao mestre.

Se a mensagem está correta, o escravo deve verificar o que foi requisitado pelo mestre e montar a mensagem de resposta com o CRC equivalente, dessa forma o quadro de resposta é enviado. A finalização é feita configurando o escravo para aguardo de uma nova mensagem.

Finalizado o código do Escravo MODBUS o sistema supervisório é configurado. Antes disso, um breve embasamento teórico para ambientação do software ScadaBR deve ser feito.

Sistemas SCADA (*Supervisory Control And Data Acquisition* – Controle Supervisório e aquisição de dados) servem como interface entre operador e processo.

Algumas funções são comuns a muitos sistemas SCADA, como geração de gráficos e relatórios com o histórico do processo, detecção de alarmes e registro de eventos em sistemas automatizados, entre outros.

O ScadaBR é uma aplicação multiplataforma em Java onde o software é executado a partir de um servidor (como configuração padrão, o Apache TomCat). Na execução do aplicativo, pode-se acessá-lo a partir de um navegador de internet.

Para a utilização do sistema supervisório ScadaBR como Mestre MODBUS, bem como interface de leitura para gerenciamento do processo em questão, fez-se necessário a configuração do programa de acordo com os mesmos parâmetros utilizados no microcontrolador.

Neste caso, utilizou-se como velocidade de transmissão 9600 bps, 8 *bits* de dados, 1 *stop bit*, sem *bit* de paridade e com modo de transmissão RTU.

Com a finalização do programa do escravo e a configuração do mestre, o último passo é a implementação física da comunicação.

Para tanto, utilizou-se um conversor USB/serial (RS232) e em sequência um CI conversor serial/TTL (MAX232) para que fosse possível a comunicação entre o sistema supervisório instalado no PC e o programa no PIC. A figura 6 mostra um diagrama de blocos ilustrativo da configuração utilizada.

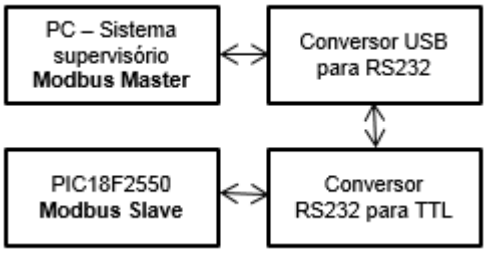

Figura 6. Esquema da configuração utilizada.

O protocolo MODBUS possui várias funções (*function code*), que podem ser de leitura de status, escrita, detecção de erros, entre outros. Para que um dispositivo com este protocolo possa ser comercializado é necessário possuir um pacote mínimo de funções, pré-determinadas no documento oficial (MODICON, 1996).

O escravo aqui desenvolvido possui somente uma das funções deste pacote, função 03 (*Read Holding Register*), que permite a leitura de vários registradores internos do mesmo.

O *query* (requisição do mestre) da função 03 necessita informar a posição do primeiro registrador e a quantidade destes que deverá ser respondido pelo escravo. No *software* desenvolvido é possível realizar a leitura de até três registradores.

O registrador da posição 0 corresponde ao valor da entrada analógica do PIC (temperatura do tanque, implementada na prática como um potenciômetro). A posição 1 e 2, são respectivamente a temperatura mínima e máxima do controlador.

3. Resultados e Discussão

Alguns pontos são de relevante aspecto na análise da comunicação obtida neste artigo, bem como os problemas identificados durante o decorrer do mesmo.

Sabe-se que o mestre deve enviar um sinal (de acordo com o protocolo explicado anteriormente), isto é, campo de endereço (01h), função (03h), posição do registrador inicial (00h), e número de registradores (03h), juntamente com os 16 *bits* do CRC. A figura 7, mostra este sinal sendo enviado no experimento físico montado.

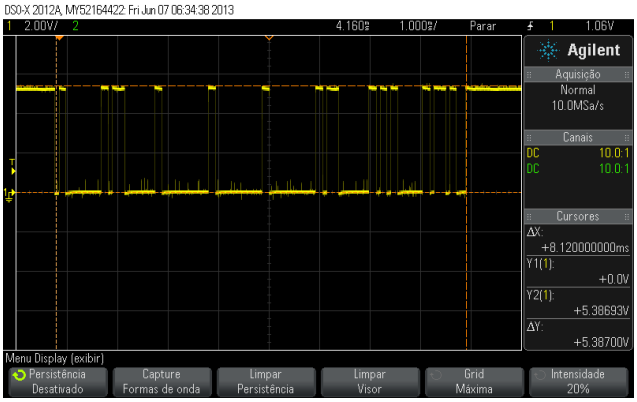

Figura 7. Sinal serial transmito pelo mestre para o escravo, obtido em osciloscópio.

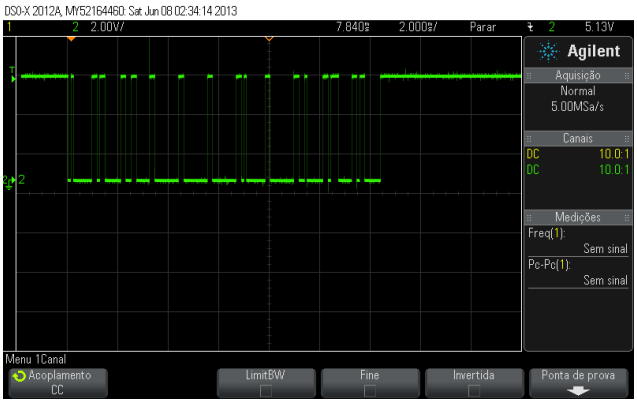

Figura 8. Sinal do escravo sendo transmitido para o mestre como resposta.

Para que exista comunicação, o escravo deve enviar sua resposta para o mestre, de acordo com o que foi solicitado. A figura 8 mostra este sinal sendo transmitido na implementação física, onde os valores dos registradores lidos foram fixados em 00h, 01h e 02h

respectivamente para fins práticos.

A figura 9 mostra os dois sinais simultaneamente, onde primeiramente o mestre faz o pedido que é respondido pelo escravo.

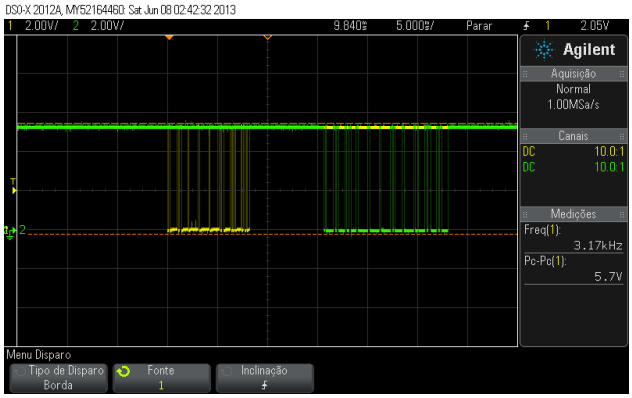

Figura 9. Sinal do mestre (amarelo) e escravo (verde) simultaneamente.

Já na figura 10 pode-se visualizar a leitura dos valores acima descritos pelo ScadaBR, comprovando o funcionamento.

Figura 10. Valores lidos no ScadaBR.

Além de problemas na implementação do *software*, o qual foi construído exclusivamente a partir da documentação original, isto é, sem a utilização de bibliotecas já desenvolvidas, as maiores dificuldades ocorreram no processo de condicionamento do sinal físico. Esta dificuldade pode ser ilustrada pela figura 11.

Na figura 11 pode-se ver claramente a interferência do transmissor na linha do receptor (primeira parte do sinal amarelo), mas que não caracterizou um problema, sendo que a comunicação é por característica *half-duplex*.

Outro problema verificado é a falta do sinal negativo característico do padrão EIA-232. Este último sim ocasionou instabilidade nas leituras realizadas, sendo que por vezes o mestre não reconhecia a resposta. A solução não foi encontrada, porém, visto que o CI MAX232 foi conectado de acordo com seu respectivo *datasheet* (TEXAS INSTRUMENTS INCORPORATED, 2004) e substituído por um segundo componente, especula-se que o problema possa estar no conversor USB/Serial utilizado.

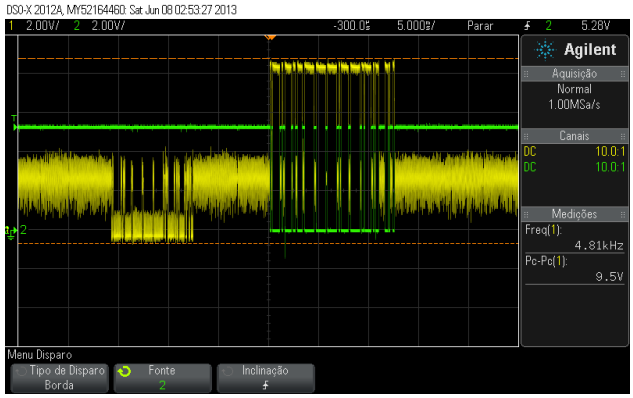

Figura 11. Sinal TTL (verde) convertido a EIA-232 (amarelo) aplicado ao *Receiver* do microcontrolador.

Uma característica que deve ser ressaltada com relação a este projeto, é o seu custo de implementação. Pode-se por exemplo utilizar um dispositivo como o desenvolvido para diversas automatizações que necessitem do protocolo MODBUS, onde outras funções do mesmo podem ser adicionadas com relativa facilidade.

4. Conclusão

Este artigo mostra que a aplicação de uma rede MODBUS para automatização dos mais variados processos pode ter custos bem reduzidos.

Com a aplicação de outras funções existentes no protocolo é possível efetuar escritas através do supervisório, aumentando a gama de aplicações e possibilidades de controle do dispositivo desenvolvido.

A comunicação se mostrou satisfatória mesmo com o sinal físico não operando de forma ideal (RS232 sem a parte negativa do sinal), mostrando a robustez do sistema.

Como projetos futuros pretende-se programar funções de escrita e de checagem de erros para o escravo, bem como o levantamento de custos para implementação deste sistema em um processo industrial.

Referências

SCADABR. **Manual do Software: Sistema Open-Source para Supervisão e Controle.** 2010.

MORAES, Cicero Couto de; CASTRUCCI, Plínio de Lauro. **Engenharia de automação industrial**. Rio de Janeiro: Ltc S.a., 2001.

MICROCHIP. **PIC18F2455/2550/4455/4550** Data Sheet: 28/40/44-Pin, High-Performance, Enhanced Flash, USB Microcontrollers with nanoWatt Technology, 2009. Disponível em: <http://ww1.microchip.com/downloads/en/DeviceDoc/39632e.pdf>. Acesso em: 10 maio 2013.

MODBUS (Org.). **Modbus over Serial Line**: Specification & Implementation guide. V 1.0, 2002. Disponível em: <http://www.modbus.org/>. Acesso em: 03 abr. 2013.

MODICON. **Modbus Protocol: Reference Guide.** North Andover, Massachusetts, 1996.

PEREIRA, FABIO. **Microcontroladores PIC: Programação em C.** São Paulo: Érica Ltda, 2003.

TEXAS INSTRUMENTS INCORPORATED. **MAX232, MAX232I: Dual EIA-232 drivers/receivers.** Disponível em: <http://www.ti.com/lit/ds/symlink/max232.pdf>. Acesso em: 05 jun. 2013.

Nome completo: Talita Tobias Carneiro
Filiação institucional: Universidade Tecnológica Federal do Paraná – UTFPR
Departamento: Coordenação de Eletrônica – COELE
Função ou cargo ocupado: Estudante de graduação
Telefones para contato: (42) 9938-7695
e-mail: talitatobias@outlook.com

Nome completo: Daniel Koslopp
Filiação institucional: Universidade Tecnológica Federal do Paraná – UTFPR
Departamento: Coordenação de Eletrônica – COELE
Função ou cargo ocupado: Estudante de graduação
Telefones para contato: (42) 9933-7161
e-mail: koslopp@gmail.com

Nome completo: Flavio Trojan
Filiação institucional: Universidade Tecnológica Federal do Paraná – UTFPR
Departamento: Departamento Acadêmico de Eletrônica – DAELE
Função ou cargo ocupado: Professor do Curso de Engenharia Eletrônica
Endereço completo para correspondência: Av. Monteiro Lobato Km 4, s/n - Ponta Grossa – PR
Telefones para contato: (42) 3220-4825
e-mail: trojan@utfpr.edu.br

Nome completo: Murilo Leme
Filiação institucional: Universidade Tecnológica Federal do Paraná – UTFPR
Departamento: Departamento Acadêmico de Eletrônica – DAELE
Função ou cargo ocupado: Professor do Curso de Engenharia Eletrônica
Endereço completo para correspondência: Av. Monteiro Lobato Km 4, s/n - Ponta Grossa – PR
Telefones para contato: (42) 3220-4825
e-mail: muriloleme@utfpr.edu.br