Thiago Astolpho Maziero

SISTEMA MULTIPROTOCOLO CONTROLADOR E SUPERVISÓRIO DE CÉLULA DE CARGA

Trabalho de Conclusão de Curso apresentado

à Escola de Engenharia de São Carlos, da

Universidade de São Paulo

Curso de Engenharia de Computação

ORIENTADOR: Evandro Luís Linhari Rodrigues

São Carlos

2011

Ficha catalográfica preparada pela Seção de Tratamento
da Informação do Serviço de Biblioteca – EESC/USP

Maziero, Thiago Astolpho

M476s Sistema multiprotocolo controlador e supervisório de célula de carga. / Thiago Astolpho Maziero ; orientador Evandro Luís Linhari Rodrigues -- São Carlos, 2011.

Monografia (Graduação em Engenharia de Computação) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2011.

AGRADECIMENTOS

Agradeço primeiramente aos meus pais, Valmir e Nanci, pelo apoio e educação não só durante a graduação mas por toda minha vida.

Agradeço ao Prof. Dr. Evandro pelo apoio durante o desenvolvimento deste projeto e a todos os professores, que me forneceram o conhecimento necessário para desenvolver este.

Agradeço aos meus amigos e colegas da universidade pela convivência durante estes quase cinco anos.

RESUMO

Sensores são dispositivos utilizados na medição de grandezas físicas, para isto eles devem transformar tal grandeza em algum sinal analógico ou digital. Utilizando estes dados é possível supervisionar ou até mesmo controlar processos. O objetivo deste trabalho é criar um sistema autônomo capaz de controlar um processo, através de saídas a relê utilizando um sensor do tipo célula de carga, além de oferecer a possibilidade de supervisioná-lo tanto em rede local quanto em redes de longa distância. Para isto, utilizou-se uma rede RS485 com o protocolo MODBUS. Assim, é possível a transferência dos dados do controlador a um servidor *web* que irá disponibilizá-los na internet, possibilitando o acesso aos dados de qualquer navegador. O microcontrolador utilizado foi o AT89S53de 8 *bits*. Foi utilizado o conversor analógico digital LTC2440 de 24 *bits* que possibilita interpretação do sinal gerado pela célula de carga e uma memória EEPROM para armazenar as configurações do controlador. O sistema desenvolvido se mostrou funcional, possibilitando o controle do processo através das saídas a relê presentes neste, utilizando os dados recebidos da célula de carga, além da comunicação com o servidor que tornou possível a supervisão deste de qualquer computador com internet.

Palavras-Chave: Célula de carga, EEPROM, MODBUS, microcontrolador, SPI, LabView, Servidor web.

ABSTRACT

Sensors are devices used for measuring physical quantities, in order to do that they must convert these quantities in an analog or digital signal. Using this data it is possible to monitor or even control processes. The project's objective is, using a sensor called load cell**,** to create an autonomous system capable of controlling a process, using relays, and also offers the possibility to supervise it, both in local and long distance networks. To do that an RS485 network under MODBUS protocol is used, therefore it is possible to transfer data from the controller to a web server that will host it on the internet, enabling the data access from any browser. The AT89C53 microcontroller was used along with the LTC2440, which is an analog to digital converter, that will allow the interpretation of the signal generated by the load cell and an EEPROM memory was used to store the settings of the controller. The developed system proved functional, allowing control of the process through the relay outputs, using the data received from the load cell, and the communication with the server that made possible the supervision through any computer with internet access.

Key-Words: Load cell, EEPROM, MODBUS, microcontroller, SPI, LabView, web server.

LISTA DE FIGURAS

LISTA DE TABELAS

Sumário

1.INTRODUÇÃO

Desde o início das trocas comerciais o homem sentiu necessidade de quantificar suas mercadorias para que estas pudessem ser mais justas. O avanço da ciência desde então tem contribuído para que estas tarefas se tornassem mais simples, tanto no processo de fabricação quanto na medição de quantidades para venda.

Uma destas grandezas pode ser a força exercida sobre uma célula de carga que responderá ao estímulo gerando uma tensão, depois será interpretada pelo controlador que baseado nestas leituras irá decidir como agirá no sistema.

Assim, para que se consiga melhor desempenho em sistemas de produção é necessária a implementação de sistemas de controle automatizados, os quais devem possuir a capacidade de medir as grandezas presentes no sistema, interpretá-las e então agir sob este, o controlando.

Porém, também pode haver a necessidade de monitorar o processo a distância. Assim existem diversos protocolos de comunicação que possuem diferentes nichos de aplicação, estes então possibilitariam o monitoramento do sistema sem a necessidade de um funcionário estar presente.

1.1 JUSTIFICATIVA

Atualmente o uso de célula de carga como transdutores de força se tornou muito comum, tanto no setor comercial, em balanças, até no setor industrial, utilizada na indicação de grandezas e controle de processos.

Por isto é importante que existam sistemas capazes de interpretar os sinais gerados por células de carga e baseado nestes dados controlar um processo. Além disso, cada vez mais, tem-se necessitado ou desejado supervisionar tais processos a distância.

Assim, foi desenvolvido este trabalho de conclusão de curso, para que fosse possível realizarem-se tais tarefas. Ou seja, controlar e supervisionar um processo que possui uma célula de carga como sensor.

O projeto visa criar um sistema cujo objetivo é tornar possível a supervisão e/ou controle de outro sistema que possua uma célula de carga. Este sistema a ser controlado deverá disponibilizar um sinal da ordem de milivolts o qual passará por um conversor A/D para que então seja interpretado por um microcontrolador.

O microcontrolador deverá, então, baseado neste sinal e em suas configurações controlar o processo. Este também estará conectado a uma rede RS485, que possibilitará a transmissão destas informações para um computador.

No computador estarão instalados dois *softwares*, o primeiro foi desenvolvido em *LabView* e sua função é comunicar com controlador através da rede RS485, podendo assim, tanto ler dados quanto escrevê-los, desta forma é possível configurá-lo via computador e supervisionar o processo. A segunda função deste *software* é enviar, periodicamente, parte destes dados coletados ao outro programa que estará executando. O segundo *software* é um programa escrito em *Java* que a partir dos dados recebidos funcionará como um servidor *web,* assim ele hospedará um *site* que exibirá estes dados de forma gráfica. A figura 1 mostra um diagrama do sistema.

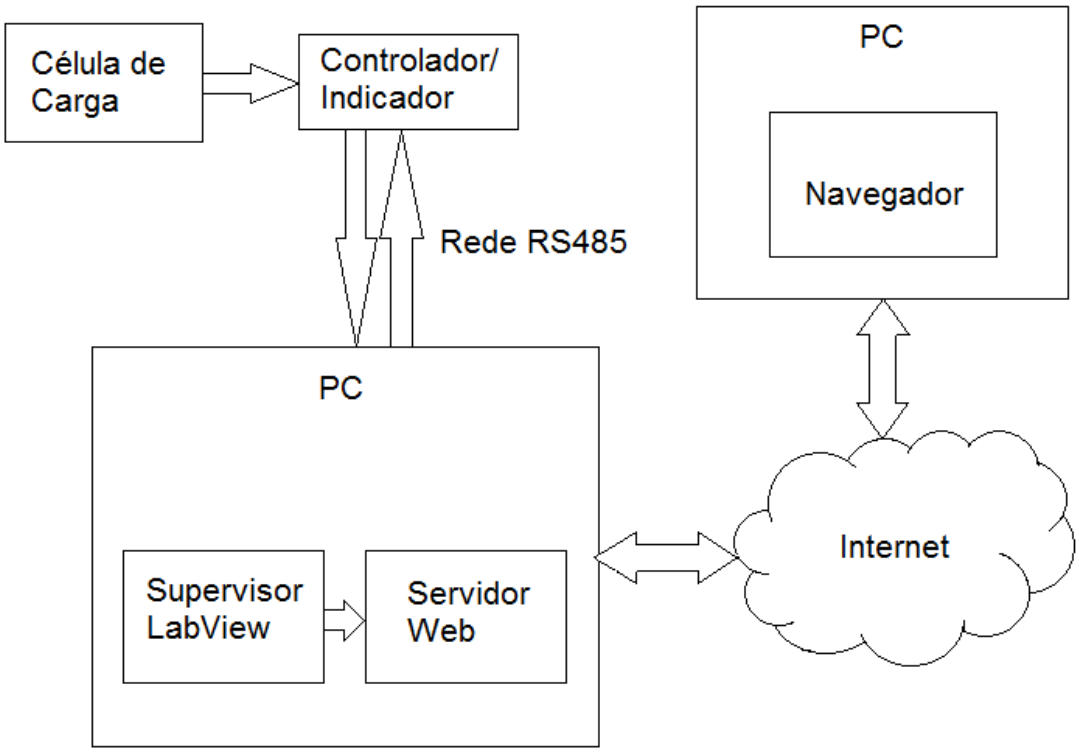

Figura 1 - Diagrama do sistema proposto.

1.2 OBJETIVO

Programar um microcontrolador **AT89S53,** para controlar um processo, baseando-se nos dados coletados de uma célula de carga. Deverá possuir a capacidade de se comunicar via rede serial RS485, implementar um servidor web para obtenção

dos dados do controlador via rede serial para permitir a monitoração de qualquer lugar com acesso a internet.

1.3 ORGANIZAÇÃO DO TRABALHO

O trabalho foi divido em cinco capítulos, cujas descrições estão a seguir:

- O capítulo 2 contém informações sobre o funcionamento dos protocolos utilizados na implementação deste trabalho, entre eles: SPI, I2C, MODBUS, TCP/IP.
- No capítulo 3 serão descritas as etapas de desenvolvimento do trabalho.
- No capítulo 4 são descritos os testes realizados e os resultados obtidos ao realizá-los.
- No capítulo 5 são analisados os resultados obtidos a partir dos testes realizados, descritos no capítulo anterior.

2. FUNDAMENTAÇÃO TEÓRICA

2.1 CÉLULA DE CARGA

Célula de carga é um dispositivo, que por meio da variação ôhmica de um sensor do tipo extensômetro ou *stain gage*, figura 2, transforma uma deformação de um corpo em uma saída de tensão.

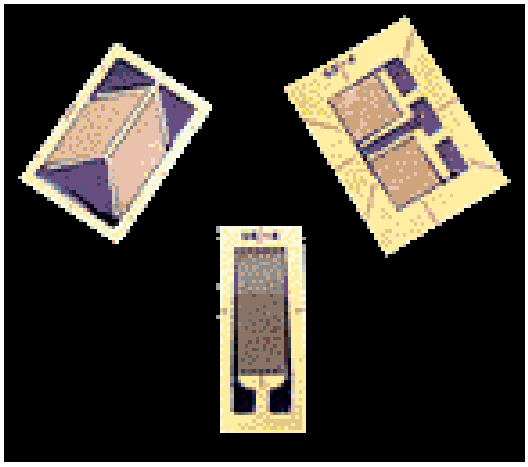

Figura 2 - Extensômetro(retirado de Célula de Carga, Carer, Maurício e Carraro, Edver).

A célula é formada por um ou mais extensômetros formando uma ponte de *Wheatstone*, figura 3, desta forma o desbalanceamento da ponte, devido à variação da resistência do sensor, gerará uma tensão e por meio da medição desta pode-se obter o valor da força aplicada a célula de carga.

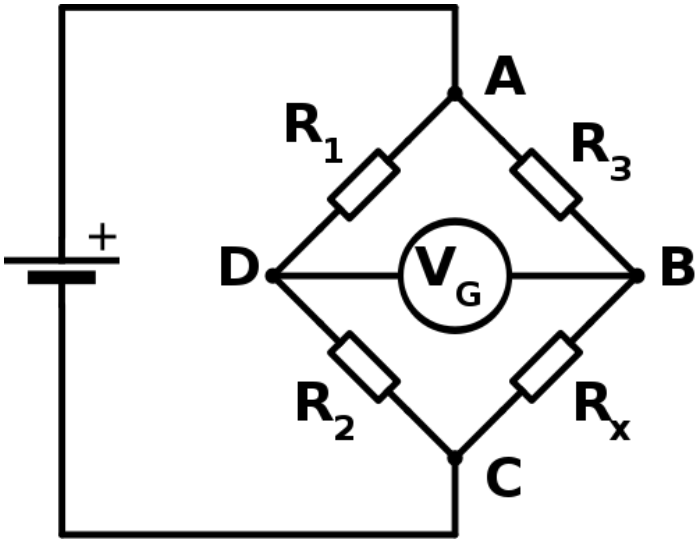

Figura 3 - Ponte de *Wheatstone*.

Os extensômetros, que formam a ponte de *Wheatstone*, são colados ao corpo da célula de carga, uma peça metálica(alumínio, aço, liga de cobre-berílio), que se deformará juntamente aos extensômetros. Desta forma qualquer deformação sofrida pelo corpo da célula será transmitida aos extensômetros. O corpo da célula deve ser cuidadosamente projetado, tal que sua forma e características transmitam o mais fielmente possível as deformações sofridas aos sensores. Outro cuidado que deve-se tomar é devido às deformações que variações na temperatura podem causar erro nas leituras. Este problema é contornado introduzindo-se resistências no circuito de *Wheatstone* que possuam o efeito inverso. Desta forma compensando os efeitos causados pela temperatura. A figura 4 exibe uma célula de carga.

Figura 4 - Exemplo de célula de Carga.

2.2 PROTOCOLO MODBUS

Este protocolo de comunicação funciona utilizando arquitetura mestre/escravo, desta forma o mestre sempre iniciará um ciclo de comunicação fazendo uma requisição ao escravo que deverá respondê-la. Como mostra a figura 5.

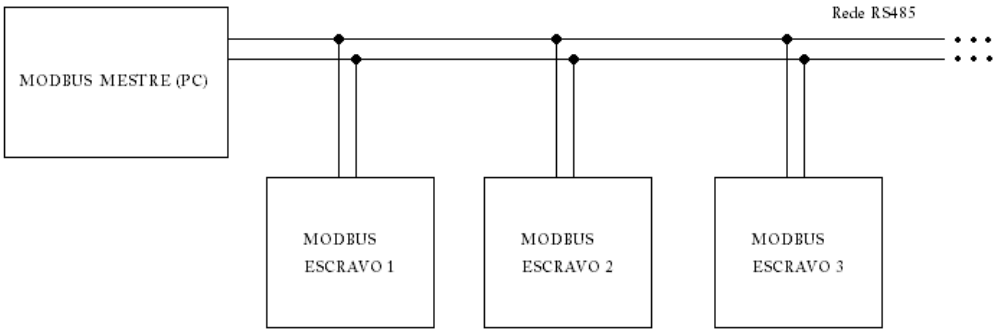

Figura 5 - Rede MODBUS.

O protocolo suporta um mestre e 127 escravos em uma mesma rede, como uma RS485, porém devido a limitações físicas desta o número de escravos pode cair. A cada novo equipamento inserido na rede a impedância resultante irá diminuir, chegando a um ponto que impossibilite a comunicação. Um pacote *MODBUS* é composto por quatro estruturas: Endereço do escravo, código da função, dado e checagem de erro (*CRC*), estes campos são *big-endian,* ou seja, o *byte* mais significativo é enviado primeiro, figura 6.

Figura 6 - *Frame* MODBUS.

O protocolo possui endereçamento de 16 *bits* possibilitando o acesso a 65536 endereços e estes dados, também, serão de 16 *bits*.

Ele possui diversas funções (mais informações *MODBUS APPLICATION PROTOCOL SPECIFICATION*, 2006) porém para uma comunicação eficiente são necessárias pelo menos duas funções que são as funções 4 e 6, respectivamente, *Read Input Registers* e *Preset Single Registers* com estas é possível ler e escrever em qualquer endereço de memória que esteja acessível ao protocolo. Na figura 7 o diagrama de uma transação *MODBUS*.

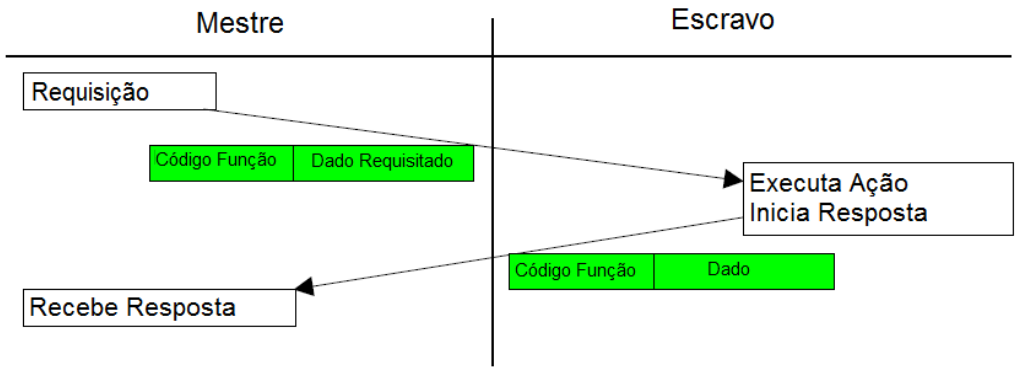

Figura 7 - Diagrama de transação MODBUS(adaptado de *Modbus Application Protocol Specification*, 2006).

A figura 8 mostra os pacotes que mestre e escravo trocarão durante uma transação cuja função é de leitura(função quatro).

Mestre:

Endereço Escravo	Função	Endereço Registrador Alto	Endereço Registrador Baixo	Quantidade Alto	Quantidade Baixo	CRC

Escravo:

Endereço Escravo	Função	Quantidade Bytes	Dado Alto	Dado Baixo	CRC

Figura 8 - *Frames* trocados durante leitura.

A figura 9 mostra os pacotes que mestre e escravo trocarão durante uma transação de escrita (função seis), note que neste caso a resposta dada pelo escravo é apenas um eco da mensagem enviada pelo mestre, funcionando como um *ack* da operação, ou seja, um indicador que a operação foi realizada com sucesso.

Mestre:

Endereço Escravo	Função	Endereço Registrador Alto	Endereço Registrador Baixo	Valor Dado Alto	Valor Dado Baixo	CRC

Escravo:

Endereço Escravo	Função	Endereço Registrador Alto	Endereço Registrador Baixo	Valor Dado Alto	Valor Dado Baixo	CRC

Figura 9 - *Frames* trocados durante escrita.

2.3 PROTOCOLO I²C

O protocolo I²C foi originalmente desenvolvido pela *Philips Semiconductors,* este possui dois canais de comunicação e velocidade de transmissão de baixa a média. O protocolo realiza uma comunicação serial *chip-to-chip* utilizando apenas dois canais, diferentemente do método mais utilizado antes de seu desenvolvimento que era a comunicação paralela de 8 *bits*.

Ele é do tipo mestre/escravo, ou seja, a comunicação será iniciada por um mestre que fará uma requisição de leitura ou escrita. Um dispositivo pode assumir qualquer uma das condições, mestre ou escravo, dependendo da necessidade da aplicação.

Muitos *chips* podem ser conectados a rede I²C, e para que isto seja possível cada um destes deve possuir um endereço único. Sempre que um mestre enviar uma mensagem esta deve conter o endereço do escravo desejado.

Para isto são utilizados apenas dois canais de comunicação bidirecionais, um canal de dados(SDA) e um canal de *clock*(SCK). Ambos devem funcionar em sincronia para que a comunicação seja efetuada com sucesso.

Assim para se escrever e ler informações desta deve-se seguir os seguintes frames, figuras 10 e 11 respectivamente:

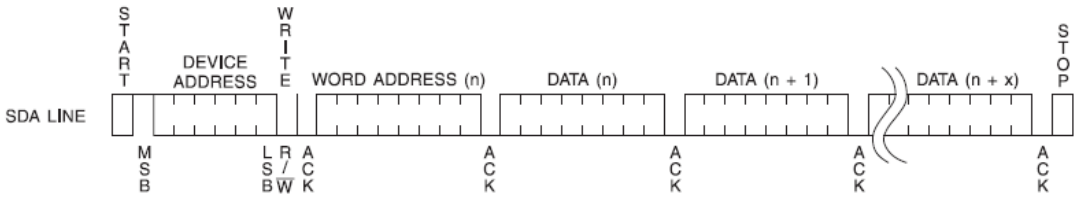

Figura 10 - Bytes a serem enviados para escrita na EEPROM(retirado *de AT24C16 datasheet, ATMEL*).

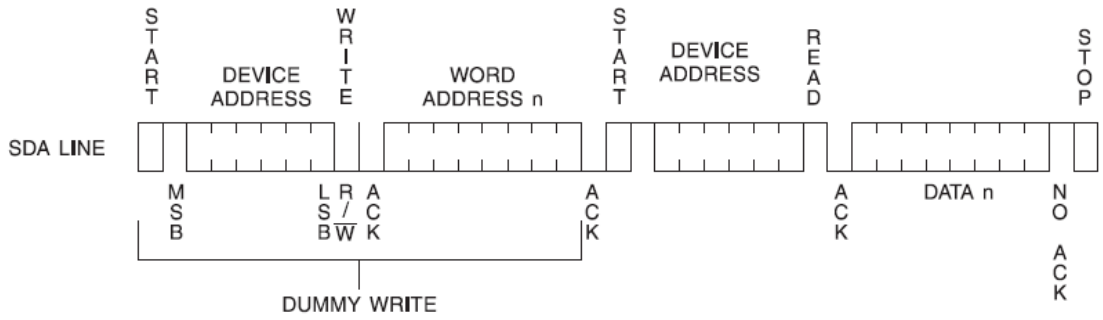

Figura 11 - Bytes a serem enviados para leitura na EEPROM retirado *de AT24C16 datasheet, ATMEL*).

Nota-se nas figuras 10 e 11, condições especiais utilizadas para iniciar(*START*), parar(*STOP*) e confirmação(*ACKNOWLEDGE*), elas estão expostas nas figuras 12 e 13.

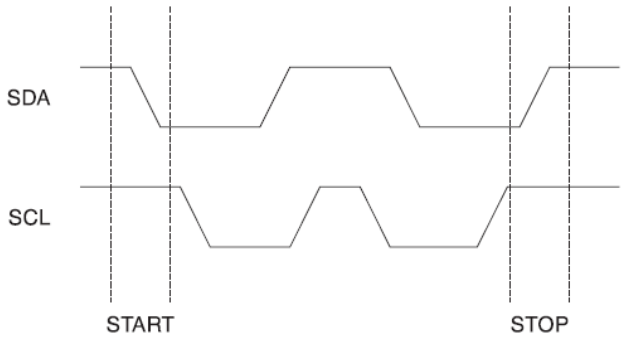

Figura 12 - Bytes de *START* e *STOP* retirado *de AT24C16 datasheet, ATMEL*).

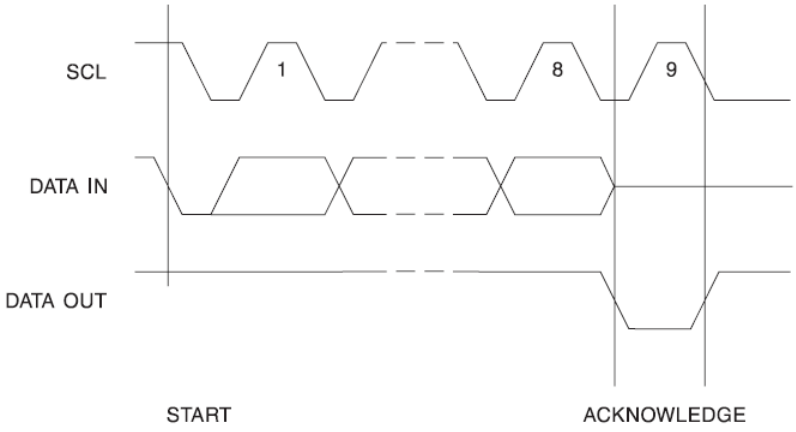

Figura 13 - Byte de *ACKNOWLEDGE* retirado *de AT24C16 datasheet, ATMEL*).

2.4 PROTOCOLO SPI

O *SPI* (Serial Peripheral Interface Bus) é um protocolo de comunicação serial síncrona desenvolvido pela Motorola que opera em modo *full duplex*, ou seja, é possível receber e enviar dados simultaneamente.

Os dispositivos se comunicam no formato mestre/escravo, para isto são utilizados quatro sinais: *SCK*(*clock* serial controlado pelo mestre), *SDO*(saída serial de dados do mestre), *SDI*(entrada serial de dados do mestre) e *CS*(*chip select,* seleciona o periférico a ser utilizado). Os sinais SPI possuem diversas nomenclaturas que estão representadas na tabela 1.

Tabela 1 - Nomenclatura SPI.

Nomes utilizados	Outros nomes
SCK	SCLK, CLK
SDI	MISO, SOMI, DI, DIN, SI
SDO	DO, SOUT, SO, MOSI, SIMO
CS	SS, nCS, CSB, CSN, nSS, STE

A figura 14 mostra um exemplo de conexão simples entre dois dispositivos com comunicação SPI.

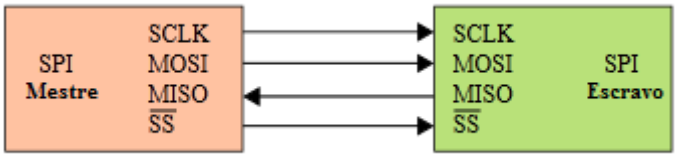

Figura 14 - Comunicação SPI.

O protocolo funciona com um mestre e pelo menos um escravo. Para que seja possível a utilização de mais de um periférico, eles devem possuir a capacidade de aumentar a impedância de seus pinos de saída de dados quando não estiverem selecionados, assim sua saída possui três estados: lógico alto, lógico baixo e alta impedância, sendo denominadas *tri-state*.

O inicio da comunicação dar-se-á quando o mestre tornar o pino $\overline{CS}$ "zero", desta forma a comunicação com o periférico o qual foi selecionado estará ativo. Então a cada pulso efetuado no SCK um novo *bit* pode ser escrito no SDO e um lido do SDI. O significado de cada bit lido e escrito deve variar de acordo com o periférico, figura 15.

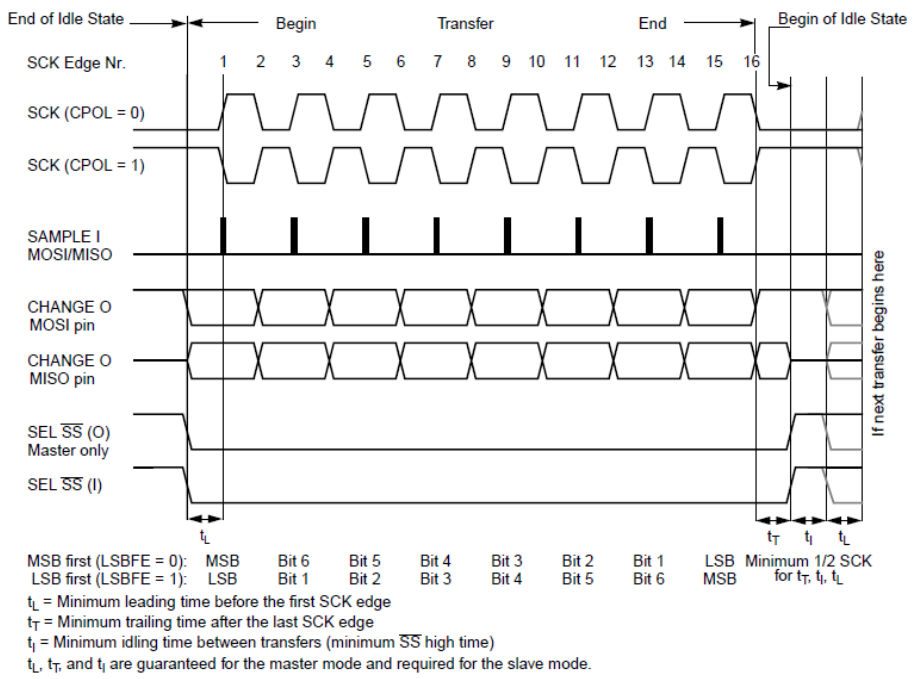

Figura 15 - Diagrama de tempo da comunicação SPI (retirado de *SPI Block Guide* v03.06, Motorola Inc.).

3. MATERIAIS E MÉTODOS

O projeto consiste em criar um sistema cujo objetivo é a utilização de dados recebidos de uma célula para supervisionar e controlar um processo industrial. Assim, este foi divido em três partes:

- **Um controlador/indicador digital –** Que tem como objetivos a obtenção dos dados da célula por meio de um conversor analógico digital, o controle do processo em questão e a transmissão dos dados via rede RS485.
- **Um supervisor em *LabView* –** Que possibilitará ao usuário configurar o controlador com maior facilidade utilizando um computador e periodicamente transmitirá dados ao servidor web para que este disponibilize os dados.
- **Servidor *Web* –** Este utilizará todos os dados recebidos do supervisor para compor um página de internet, possibilitando assim o acesso a esses dados de qualquer computador.

Nota-se que o sistema foi divido em módulos, para que eventuais manutenções ou atualizações sejam efetuadas com maior facilidade. Além disto, outra técnica de engenharia de *software* utilizada foi o uso de máquinas de estado na modelagem do programa, mais detalhes item três deste capítulo.

3.1 O MICROCONTROLADOR

Neste projeto foi utilizado o microcontrolador **AT89S53**, que possui arquitetura baseada em 8051 e é de 8 *bits*. Ele possui as seguintes características, relevantes:

- 12KB de memória *flash* para programa;
- 4V a 6V de tensão de operação;
- *Clock* de 0Hz a 24MHz;
- 256 x 8 *bit* de *RAM* interna;
- 32 pinos de I/O programáveis;
- Três temporizadores/contadores de 16 *bits*;
- Nove fontes de interrupção;
- Canal programável UART;

Com estas características é possível desenvolver as funcionalidades inicialmente propostas.

3.2 CONVERSOR ANALÓGICO DIGITAL

O *ADC* escolhido para este projeto foi o LTC2440, figura 17, devido a sua alta resolução, rápida taxa de amostragem e utilização do protocolo SPI.Ele possui até 24 *bits* de resolução. Este *ADC* foi escolhido por, além da alta resolução, possuir alta taxa de conversão, entrada diferencial e comunicação SPI.

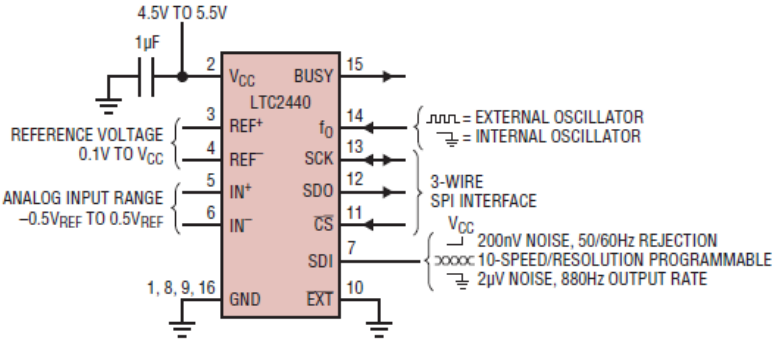

Figura 16 - Pinos do LTC2440(retirado de LTC2440 datasheet, Linear Technology).

A comunicação com o conversor A/D é feita via SPI, porém apenas dois dos três canais são utilizados, além do *chip enable*, já que não é necessário enviar nenhum dado para o conversor. Este canal apenas seria utilizado caso fosse preciso alterar a freqüência com que o A/D realizava a conversão.

Assim, serão lidos 32 *bits*, sendo:

- O primeiro *bit* é um indicador de término da conversão, caso esteja seja “1” a conversão não está terminada;
- O segundo *bit* sempre será “0”;
- O terceiro *bit* é de sinal “1” para sinais positivos e “0” para sinais negativos;
- A partir do quarto *bit* iniciam-se os 24 bits da conversão;
- Os cinco *bits* restantes devem apenas ser utilizados quando feitas médias entre os valores lidos;

A seguir um diagrama com os *bits*, figura 18:

EOC	-	SIG	MSB	...	LSB	...	SUB LSB
Bit 31	Bit 30	Bit 29	Bit 28		Bit 5		Bit 0

Figura 17 - *Bits* a serem recebidos do ADC.

Na realidade precisaremos de 3 *bits* para são estes: o *chip select*, *SDO* e *SCK*. Assim na figura 19 segue o diagrama de fluxo que cada conversão deve seguir.

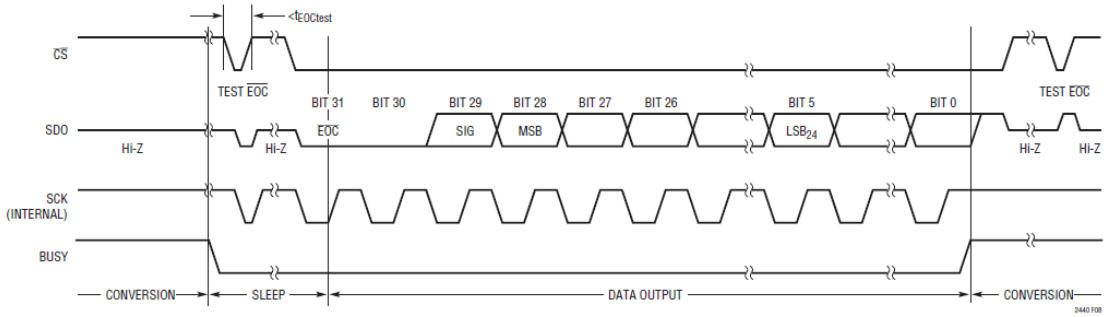

Figura 18 - Diagrama de tempo da comunicação com o *ADC* (retirado de LTC2440 datasheet, Linear Technology).

3.2 MEMÓRIA EEPROM

Foi utilizada a memória AT24C164 que possui 16KB, organizadas em oito blocos de 2KB cada. É possível se efetuar um milhão de ciclos de escrita nesta memória. Esta memória utiliza o protocolo I²C para comunicação com o microcontrolador, seguindo a descrição do item 3 do capítulo 2 , possibilitando que os dados de configuração do equipamento sejam salvos e obtidos novamente ao religá-lo.

Desta forma todos os parâmetros de configuração irão ser gravados na memória *EEPROM* durante o funcionamento do controlador para configurá-las novamente ao religar o equipamento.

3.3 *SOFTWARE DO MICROCONTROLADOR*

O microcontrolador é responsável controlar duas saídas digitais a relê, uma saída analógica (4mA a 20mA) e responder a pedidos de leitura/escrita de um mestre da rede RS485 utilizando protocolo *MODBUS*, estes dois últimos melhores descritos mais a frente neste capítulo. Para efetuar tal controle ele utilizará os dados da célula de carga, duas entradas digitais, os botões e suas configurações, como exibido na figura 19:

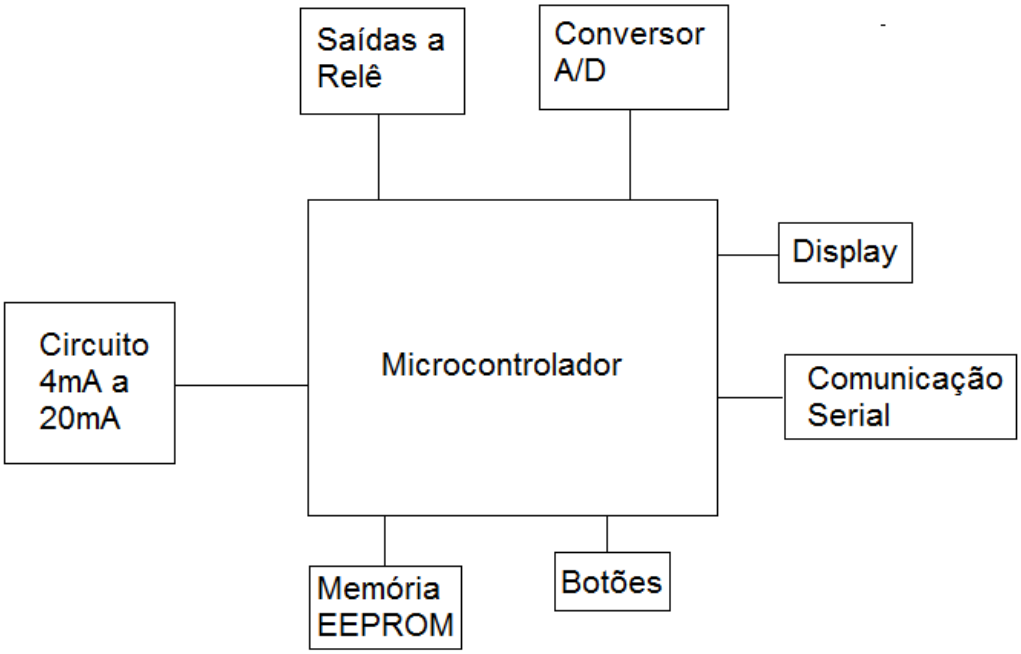

Figura 19 - O controlador.

3.3.1 Telas de Configuração

Para configurar estes parâmetros existe uma máquina de estados que irá gerir os dados a serem exibidos. Sempre que uma *flag* é acionada, geralmente ao pressionar o botão de próxima tela, máquina de estados será ativada e irá se deslocar para o próximo estado. Na figura 20 está o diagrama das telas que esta irá controlar.

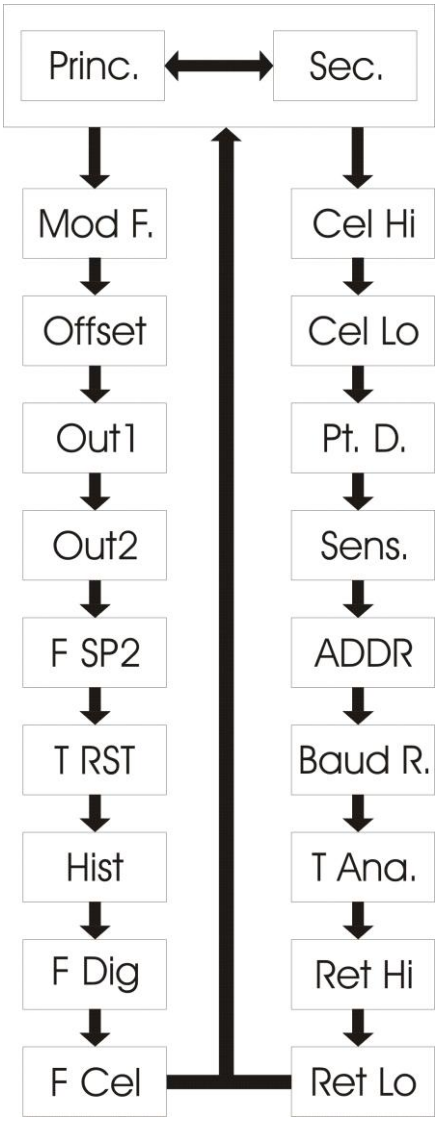

Figura 20 - Diagrama de telas do controlador.

Na figura 20 estão representadas diversas telas de configuração, alguns serão explicados em maior detalhes em sub-capítulos específicos, que são:

- **Princ** – Mostra o valor lido da célula de carga;
- **Sec** – Mostra o valor de TARA/PICO/ACUMULADO;
- **Mod F**. – Seleciona o modo de funcionando do controlador, TARA/PICO/ACUMULADO;
- **Offset** – Valor que será diretamente ao lido da célula de carga;
- **Out1** e **Out2** – Modo de funcionamento dos *Set Points* (Hi, Hi Tp, Lo e Lo Tp);
- **F SP2** – Seleciona se a saída dois será comparada com o valor armazenado em Princ ou Sec;
- **T RST** – Tempo de ativação das saídas utilizado em Hi Tp e Lo Tp;

- **Hist** – Valor de Histerese;
- **F Dig** – Filtro das entradas digitais, para que ruídos não sejam
- **F Cel** – Filtro para estabilização das leituras da célula de carga, calcula uma média entre os últimos valores lidos;
- **Cel Hi** – Valor mais alto lido pela célula de carga;
- **Cel Lo** – Menor valor lido pela célula de carga;
- **Pt. D.** – Quantidade de casas decimais a serem exibidas;
- **Sens.** – Sensibilidade da célula de carga (1mV, 2mV ou 3mV);
- **ADDR** – Endereço da comunicação RS485;
- **Baud R**. – *Baud Rate* da comunicação;
- **T. Ana.** – Seleciona se a saída 4mA a 20mA será comparada com os Set Points ou com os parâmetros Ret Hi e Ret Lo;
- **Ret Hi** – Configuração alta da escala para a saída 4mA a 20mA;
- **Ret Lo** – Configuração baixa da escala para a saída 4mA a 20mA;

3.3.2 Comunicação com *ADC*

Outro ponto importante foi o desenvolvimento da comunicação com o conversor analógico/digital. Esta foi feita utilizando o protocolo *SPI*, porém devido a não necessidade de alterar a freqüência de amostragem do *ADC*, o canal de envio de dados (SDO) para o *ADC* não foi utilizado. O quadro 1 do apêndice, exibe a o código final desenvolvido para esta comunicação.

3.3.3 Saída 4mA a 20mA

Uma das formas de retransmissão de dados é a saída de corrente 4mA a 20mA que poderá ser configurada utilizando os parâmetros Ret Hi e Ret Lo, que determinarão a escala que será transformada na saída 4mA a 20mA.

A saída 4mA a 20mA é obtida a partir de um circuito que transforma freqüência em corrente, o que é feito em dois passos: transformar freqüência em tensão (utilizou-se um LM231*, National Semiconductor*) e depois a tensão em corrente por meio de circuitos do tipo apresentado na figura 21. Para isto utiliza-se um dos *timers* presentes no microcontrolador para gerar as frequências, elas serão obtidas a partir de uma escala configurável e valores calibrados no equipamento. O quadro 2 do apêndice exibe o código que altera a freqüência com que o *timer* funcionará. Ω

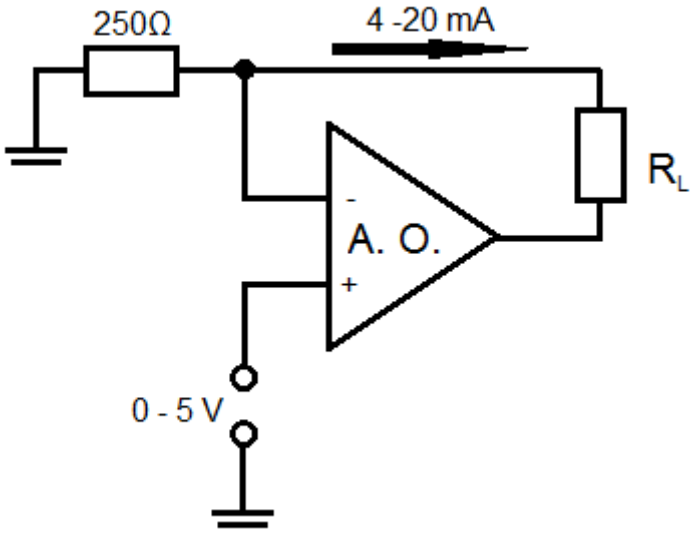

Figura 21 - Circuito para conversão tensão corrente.

3.3.4 Comunicação Serial

O protocolo MODBUS utilizado neste projeto apenas implementa duas das funções disponíveis no protocolo, porém com estas é possível alterar e ler qualquer registrador das configurações do dispositivo. Assim o *frame* do protocolo será recebido *byte* a *byte*, então este terá sua integridade verificada através do campo de checagem de erros do protocolo.

Então será verificada qual função foi requisita pelo mestre, quatro ou seis são as desenvolvidas. Assim que identificada a função, o *frame* de resposta será montado e enviado *byte* a *byte* segundo as especificações do item 2 do capítulo 2.

3.3.5 Funcionamento do controlador

A primeira atitude a se tomar é configurar as características da célula de carga ajustando as variáveis "Cel Lo" e "Cel Hi" para a escala a ser exibida e sua sensibilidade, S CEL, assim o valor indicado resultará numa conversão da tensão lida para a escala desejada.

O sistema possui três pares de *set points* configuráveis para controlar as saídas digitais, apenas um par estará ativo durante o processo, a função destes pares é facilitar a transição entre processos. Assim quatro funções, para cada saída, são possíveis aqui:

- Hi – Saída estará ativa quando acima do *set point*;
- Lo – Saída estará ativa quando abaixo do *set point*;

- HiTp – Saída estará ativa, durante um determinado tempo, configurável, quando acima do *set point*;
- LoTp – Saída estará desativada, durante um determinado tempo, configurável, quando acima do *set point* e ativa no restante;

O tempo de controle descrito acima pode ser configurável na variável *"tReSet"*.

No controle das saídas digitais outro fator importante é a variável "HIst" que fará com que a saída funcione segundo a figura 22:

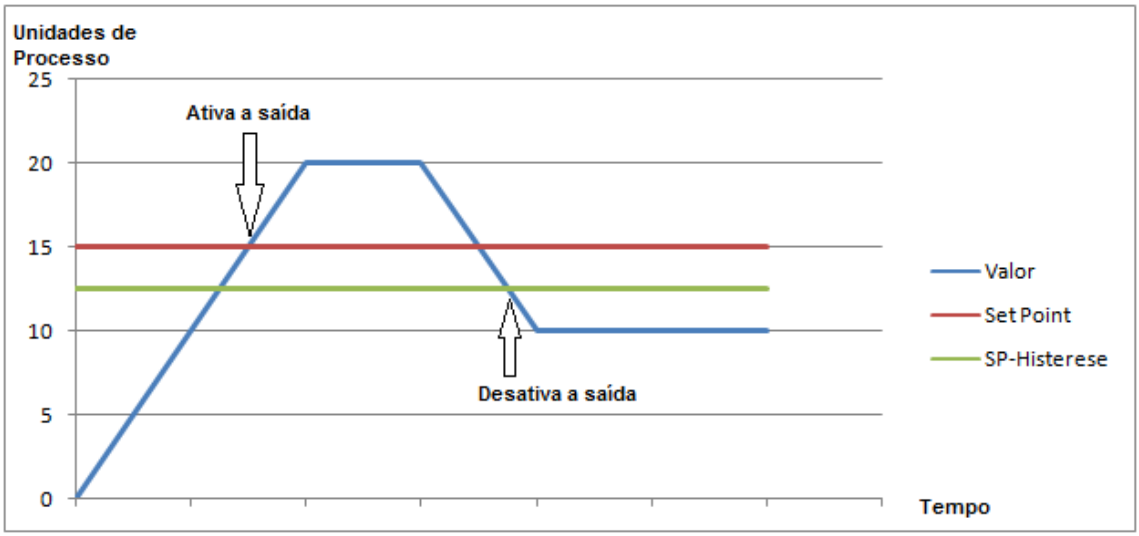

Figura 22 - Gráfico da histerese.

Na figura 24, a saída será acionada ao se tornar maior do que o *set point*, porém não será desligada ao ficar menor que este, isto apenas acontecerá quando o valor do processo for menor do que o valor do *set point* menos o valor de histerese.

O controlador possui duas telas principais, na primeira, chamada “princ”, sempre será mostrado o valor lido da célula de carga convertido para a escala desejada, já o valor indicado pela segunda tela, chamada “sec”, dependerá da função de funcionamento escolhida para o controlador:

- Tara – Neste caso o controlador funcionará como balança comum que possui tara e o valor desta será indicado na segunda tela;
- Pico – Neste modo o controlador armazenará o maior valor medido e este pico será indicado na segunda tela;
- Acumulador – Neste modo ao se pressionar um dos botões o valor que estiver mostrado, na primeira tela, será somado ao valor que estiver indicado na segunda e o resultado será mostrado na segunda tela;

A saída analógica possui duas configurações uma de mínimo e outra de máximo,para qualquer valor lido da célula abaixo ou igual mínimo resultará em uma saída de 4mA no intervalo entre estes valores a saída aumentará linearmente até alcançar o máximo, ponto a partir do qual a saída será 20mA. Conforme mostrado na gráfico da figura 23:

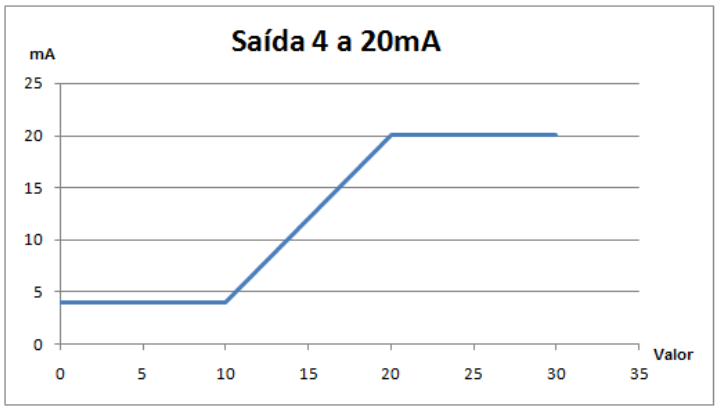

Figura 23 - Funcionamento da saída 4mA a 20mA.

Na figura 23, os valores de mínimo e máximo são de 10 e 20 unidades do processo, respectivamente. Assim para valores menores que o mínimo há uma saída de 4mA e para valores maiores que o máximo uma saída de 20mA.

Na comunicação serial RS485 são necessárias duas configurações *baud rate* da transmissão, para utilizar o software supervisor configurar 9600, e o endereço do equipamento, com estas configuradas basta abrir o supervisor indicar a porta com que a rede esta conectada e o endereço do equipamento.

O protocolo escolhido para a transmissão foi o MODBUS, porém apenas duas das funções deste foram desenvolvidas são as funções 4 e 6, respectivamente, *Read Input Register* e *Write Single Register*, com estas já é possível escrever e ler qualquer um dos endereços de memória do microcontrolador que seja acessado de forma indireta.

3.4 SUPERVISOR *LABVIEW*

Este supervisor tem como objetivo facilitar a configuração do controlador e possibilitar sua supervisão em uma rede local. O supervisor acessa o controlador via rede RS485, atuando como mestre do protocolo MODBUS. Para isto foi utilizado um *VI*

(*Virtual Instrument,* que são programas ou subprogramas em *LabView* e são assim chamados devido a sua aparência e operação que imitam instrumentos físicos, como osciloscópios e multímetros, *Introduction to NI LabView, National Instruments*), que implementa as funções de leitura e escrita do protocolo, representada na figura 24.

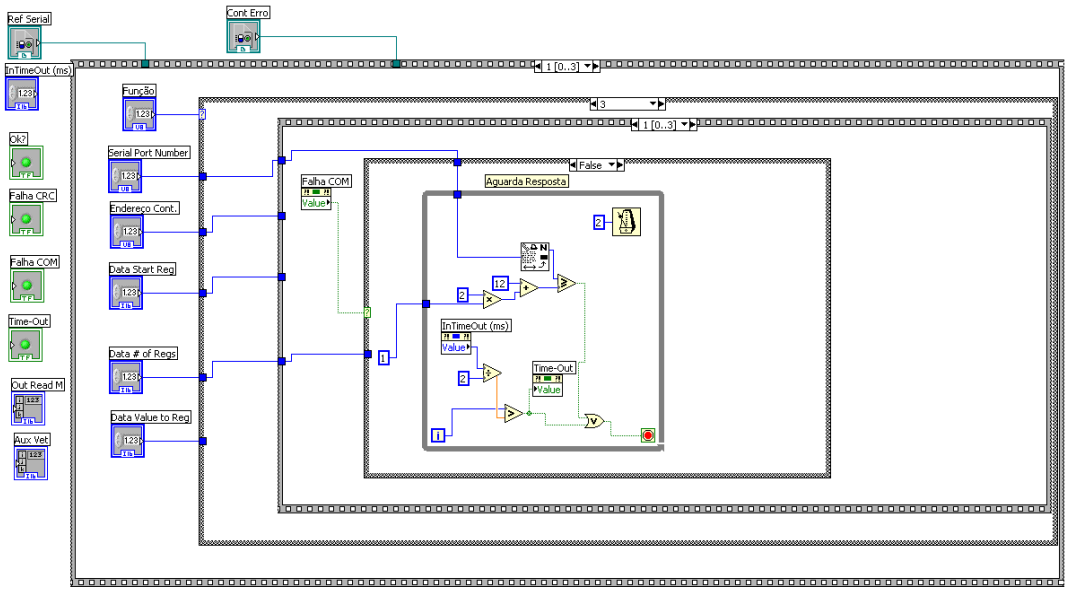

Figura 24 - *VI* de comunicação MODBUS.

Para utilizá-la é necessário que seja configurada a comunicação (porta *COM, baud rate, stop bit, paridade, etc.)*. Para isto é necessária a chamada de outros *VIs*, como se pode ver na figura 25.

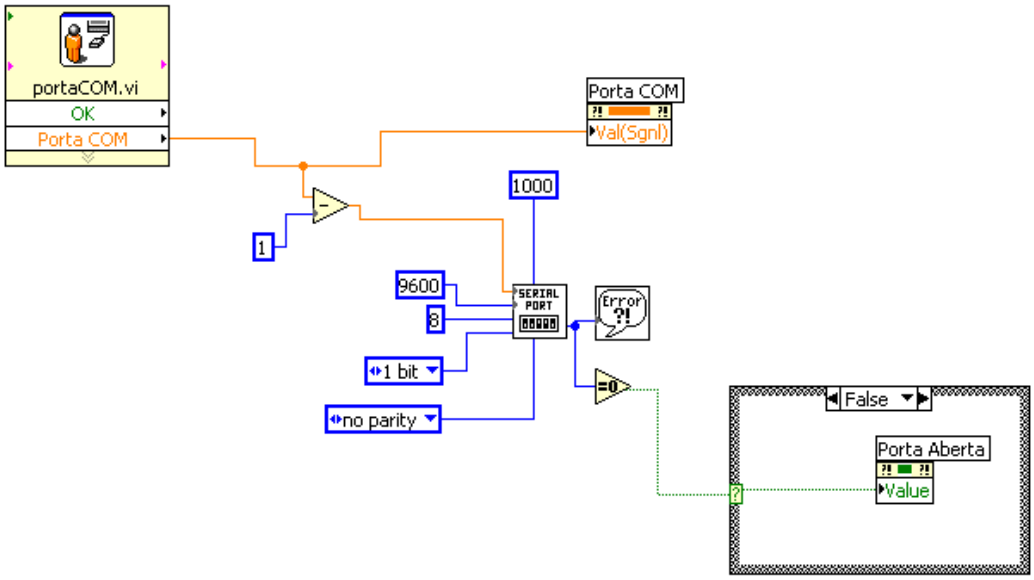

Figura 25 - Abertura da porta COM para comunicação.

Para utilizar o *VI* da figura 21, basta selecionar os seguintes parâmetros: endereço do controlador, função, endereço do registrador, quantidade de registrador (caso de leitura), valor a ser escrito (caso de escrita).

A outra função deste supervisor é o envio de dados ao servidor *web*, descrito no item 3.5, para isso ele deverá periodicamente ler dados como: valor do processo, valor auxiliar do processo, *set points* e estado das saídas a relê. Lidos estes valores ele os enviará através da pilha de protocolos TCP/IP, utilizando a porta 8000, para o servidor *web*. Então deverá esperar por uma resposta deste que deve conter o endereço do controlador cujos dados devem ser exibidos no *website*. Na seqüência Atualizando o supervisor receberá uma resposta do *webserver* com o endereço do controlador para as próximas leituras.

3.4.1 Operação

Primeiramente deve-se configurar a “porta com” que o supervisor irá utilizar para acessar a rede RS485, em seguida deve-se configurar o endereço do equipamento com o qual irá se comunicar.

Então caso a comunicação esteja funcionando corretamente, o programa irá mostrar o valor de todas as configurações e possibilitará a atualização destas através *software*, nas figuras 26 e 27 estão representadas as telas de configuração.

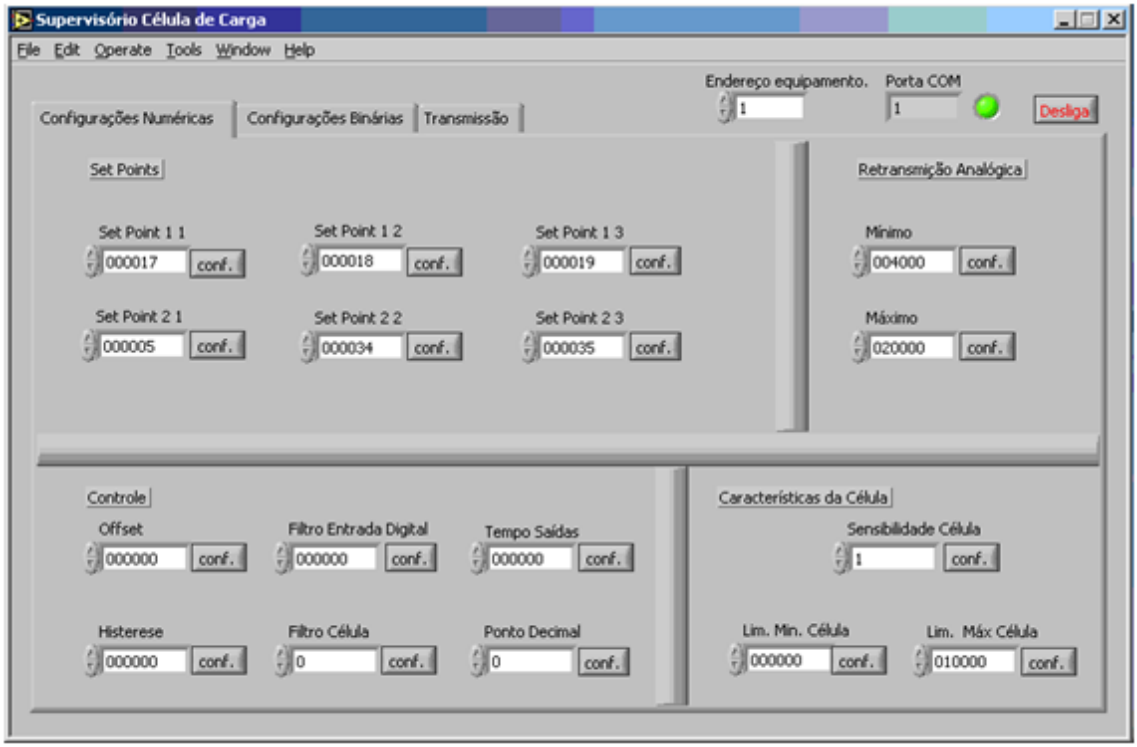

Figura 26 - Tela de configurações numéricas.

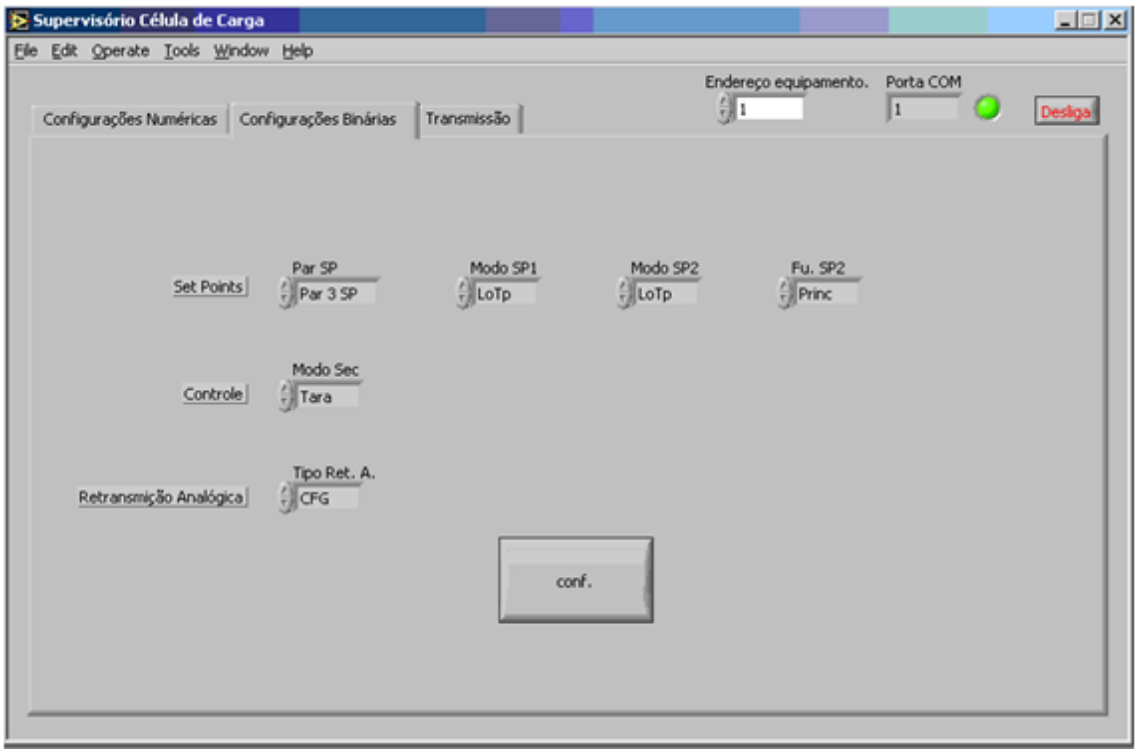

Figura 27 - Tela de configurações binárias.

A segunda função deste software é repassar informações para o *servidor web*, o qual será descrito na próxima sessão. Assim, utilizando um *socket* enviará um pacote

com as seguintes informações: Valor atual, valor tara/acumulador/pico, *set point* 1, *set point* 2, estado relê 1 e estado relê 2. Com estes valores será montada uma *string* que periodicamente é atualizada e enviada ao servidor *web*. Como mostrado na figura 28.

10.65 5.32 20.00 30.00 0 0

Figura 28 - Exemplo de *string* transmitida ao *servidor web*.

Desta forma os dados serão repassados ao servidor *web* que irá manipulá-los. Para iniciar esta transmissão deve-se ir à aba "transmissão", figura 29.

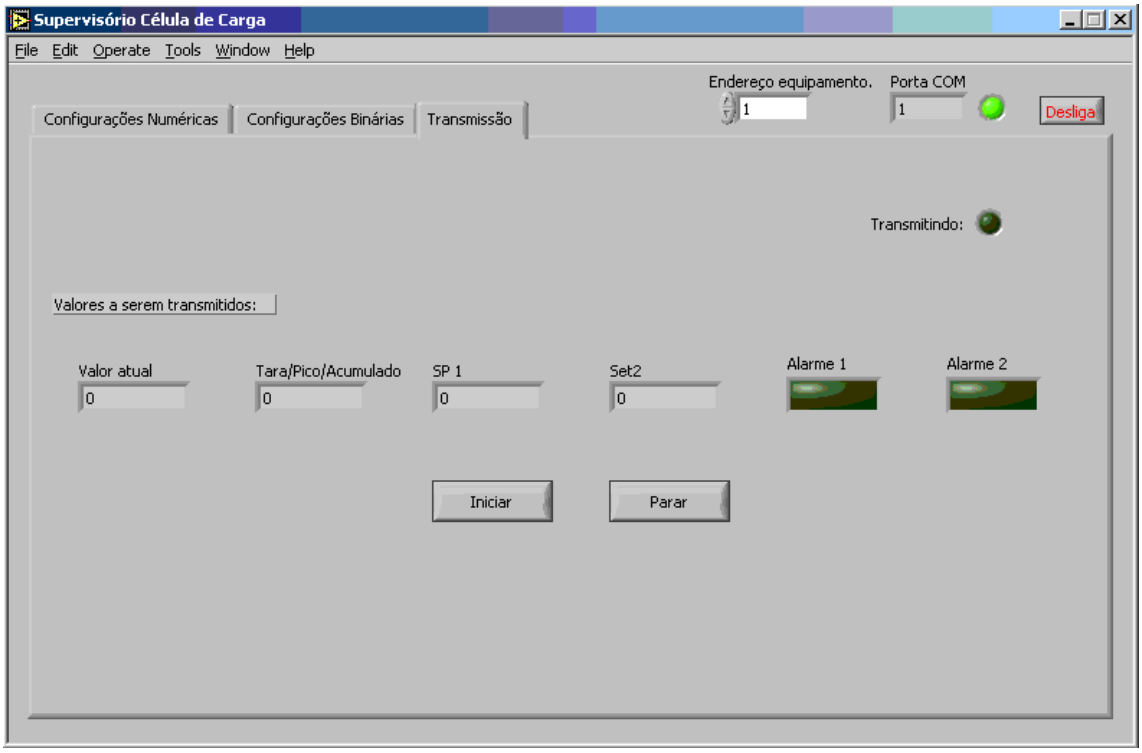

Figura 29 - Tela de transmissão ao *servidor web*.

3.5 SERVIDOR *WEB*

Utilizando os dados recebidos do supervisor *LabView* o servidor *web* irá hospedar um *website* que irá exibir os dados recebidos. Para isso ele irá traçar um gráfico e uma tabela com os últimos cento e um pontos recebidos, número decidido por questões estéticas do gráfico que será exibido.

Para isso foram desenvolvidas duas *threads,* que são diferentes fluxos de código dentro de um mesmo processo. A primeira irá aguardar pela recepção das informações que o supervisor irá transmitir para o servidor e atualizará o objeto que guarda as informações dos últimos pontos recebidos. A segunda é o servidor *web* propriamente dito, ele irá esperar por uma requisição vinda de um navegador e irá responder com o *website*.

3.5.1 Thread *comLabView*

Esta thread aguarda por uma conexão na porta 8000 e deverá receber uma *string* corretamente editada pelo supervisor *LabView*. Caso esta *string* não esteja correta a *thread* irá considerar que a comunicação não está funcionando corretamente, o que acarretará na indicação de comunicação desligada no *website*. Os dados são salvos em um objeto da classe Valores. Esta classe tem a capacidade de guardar cento e um valores de processo e de tara/acumulador/pico. Porém apenas salvará os últimos valores recebidos das saídas digitais (relês), já que não estes não são necessários para a construção do gráfico.

O quadro 3 do apêndice contém o trecho do código que insere os dados, quando corretamente formatados, no objeto em questão.

3.5.2 Thread server

Esta *thread*, utilizando os dados armazenados no objeto da classe Valores, ao receber um requisição via web irá substituir certas sequências de caracteres no código HTML, por estes dados, assim atualizando-o, para que possa ser enviado ao cliente. Além disto, utilizará métodos estáticos da classe *GeraGraf* que irá também a partir dos pontos armazenados criará um gráfico. Para isso ela deve desenhar cada uma das retas utilizadas, exemplos dos métodos utilizados estão no quadro 4 do apêndice.

Para que o gráfico seja gerado deve-se invocar o método estático (que pode ser invocado sem a necessidade de existir um objeto instanciado da classe a que pertence) *paint* da classe *GeraGraf*. Este método requer cinco parâmetros para seu funcionamento:

- Dois vetores de mesmo tamanho que contém os valores a serem indicados *set points*;
- O tamanho dos vetores;
- Os valores dos *set points*;

Então o método irá criar um objeto do tipo *BufferedImage*, neste serão definidos o tamanho e o modelo de cores (no caso RGB, o qual utiliza um número inteiro para representar as cores: vermelho, verde e azul). Com este objeto será possível gerar outro objeto, da classe *Graphics2D*, que está conectado ao primeiro. O objeto da classe Graphics2D torna possível traçar retas e escrever utilizando os métodos *draw* e *drawString*, respectivamente. Finalmente será criado um novo arquivo contendo o gráfico que substituirá o anterior, caso exista, e utilizando o método estático *write* da classe ImageIO a imagem será convertida ao formato *gif*.

3.5.3 Funcionamento

Nas figuras desta seção está exibida a forma com que os dados armazenados pelo servidor serão exibidos no *website,* além disto, os dados exibidos nestas figuras foram simulados utilizando um programa em *LabView* para enviar uma senóide ao servidor. A figura 30 mostra o website sem nenhum dado armazenado.

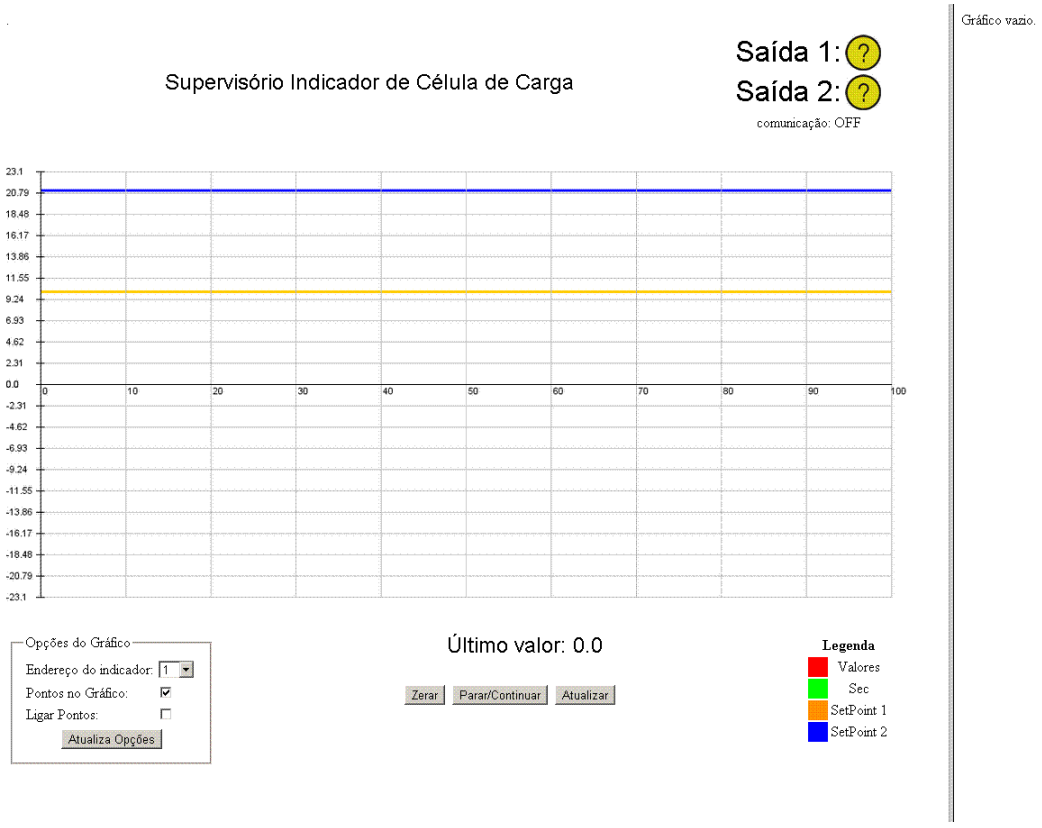

Figura 30 - Página sem dados.

Nota-se que na figura 30 é o *site* informa que a comunicação com o supervisor não está funcionando, o mesmo procedimento é efetuado caso a comunicação do supervisor com o controlador falhe e o servidor já possua dados armazenados.

Assim que a comunicação foi ativada no supervisor *LabView* o servidor começará a inserir os dados no gráficos, um novo ponto sempre será inserido na posição zero, para que isto ocorra todos os outros pontos serão deslocados. Portanto o ponto mais novo será o de número zero e portanto quanto maior sua numeração, mais antigo ele será. A figura 31 mostra o *site* já com dados inseridos.

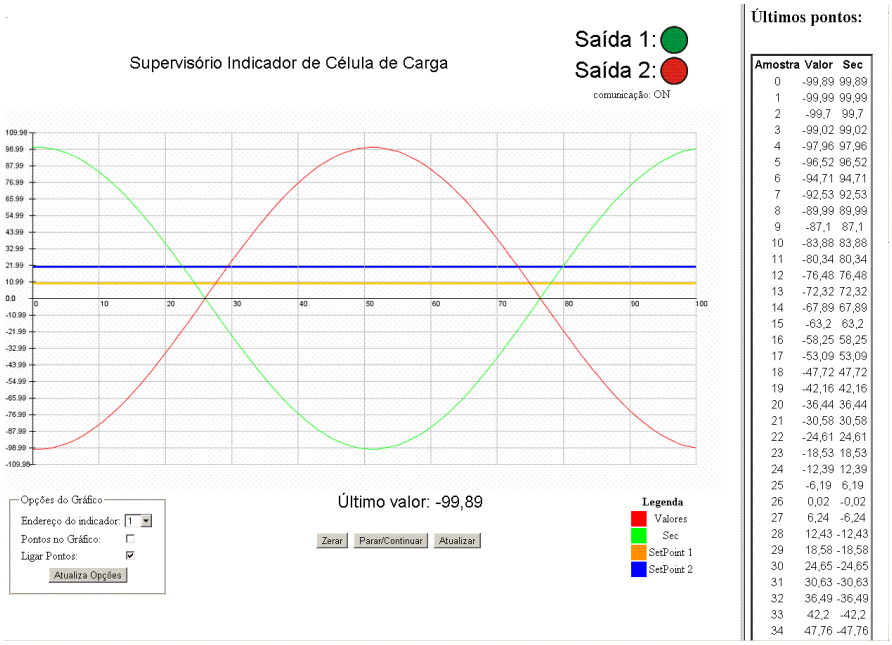

Figura 31 - Página com dados e ligando os pontos.

Na figura 31, a opção ativada seria para mostrar linhas que conectassem os pontos armazenados já na figura 32 a opção escolhida é de pontos, nesta apenas os pontos serão mostrados no gráfico.

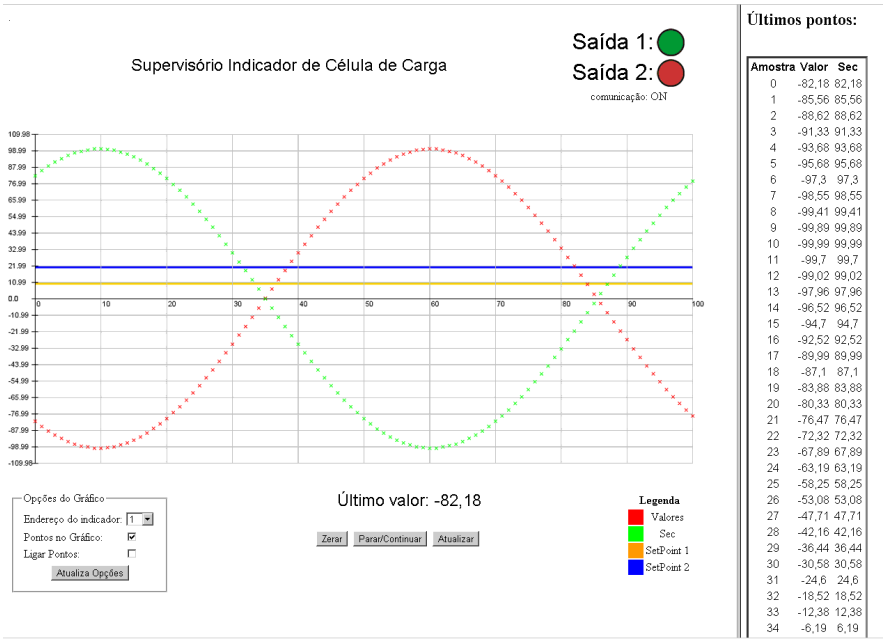

Figura 32 - Página com dado apenas com pontos.

As opções das figuras 31 e 32 podem ser ativadas simultaneamente.

Para o funcionamento do servidor *web* foi programado o protocolo *http* porém nem todos os métodos deste foram utilizados. Apenas os métodos *HEAD* e *GET* foram programados, já que com estes dois métodos foi possível desenvolver as funcionalidades desejadas ao *site*. Estas são:

- Escolher entre traçar linhas, pontos ou ambos no gráfico;
- Escolher o endereço do controlador a ser exibido os pontos, note que ao trocar o endereço o gráfico será reinicializado;
- Zerar os pontos exibidos no gráfico;
- Parar e voltar exibir os pontos recebidos do supervisor *LabView*, enquanto estiver “parado” os dados recebidos serão ignorados, ao voltar a exibir ele inserirá o próximo ponto que receber;

4. RESULTADOS E DICUSSÕES

4.1 COMUNICAÇÂO SERIAL

Para a comunicação serial utilizou-se o software *Modscan* que irá efetuar consecutivas leituras de algum endereço do controlador. Foram utilizados os parâmetros tamanho do cabo (zero, 10m, 20m, 50m, 70m) e quantidade de leituras (cem leituras). Cada teste foi executado três vezes. A tabela 2 mostra quantas leituras corretas cada ensaio obteve.

Tabela 2 - Leituras corretas no teste da serial.

Ensaio	Zero	10m	20m	50m	70m
1	100	100	99	100	100
2	100	100	100	100	99
3	100	100	100	100	100

Observando a tabela 2, nota-se que não ocorre perda significativa na comunicação RS485.

Um problema foi entrado durante o teste funcional do sistema, o *website* indicava perda de comunicação apesar de o supervisor LabView estar funcionando corretamente. Isto deve-se ao fato da taxa de comunicação estar lenta para a taxa de atualização requerida pelo servidor *web*.

4.2 SAÍDA 4mA a 20mA

Para a saída 4mA a 20mA foi testada sua resposta para um escala qualquer configurada no controlador e a resposta da saída medida com um multímetro, assim obteve-se o resultado observado na tabela 3.

Tabela 3 - Resultado para a saída 4mA a 20mA (esperado e Saída em mA).

Percentual da escala	Esperado	Saída	Desvio
0	4	3,98	0,13%
10	5,6	5,57	0,19%
20	7,2	7,17	0,19%
30	8,8	8,78	0,13%
40	10,4	10,38	0,12%
50	12	11,97	0,19%
60	13,6	13,57	0,19%
70	15,2	15,17	0,19%
80	16,8	16,77	0,19%
90	18,4	18,36	0,25%
100	20	20,03	0,19%

Na tabela 3 desvio significa o erro da saída com relação ao fundo de escala, ou seja, é o módulo da diferença entre Esperado e Saída dividido por 16 (diferença entre 4mA e 20mA). Na figura 33 é possível se observar a linearidade desta saída (como já sido previsto na figura 23).

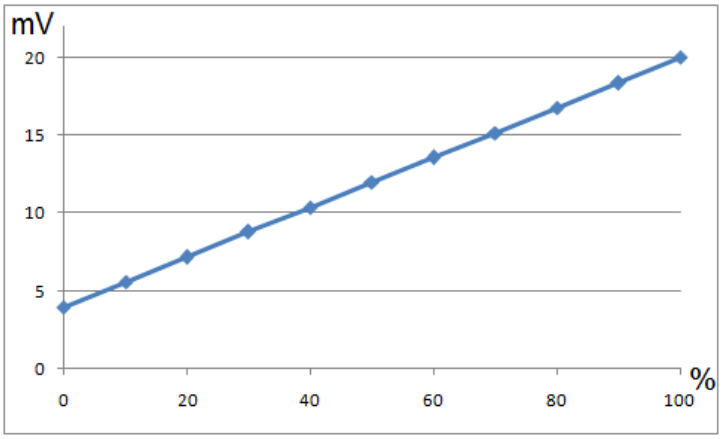

Figura 33 - Linearidade da Saída 4mA a 20mA.

4.3 CONFLITO NAS THREADS

Durante o desenvolvimento do servidor *web* foi encontrado um conflito no acesso aos dados armazenados no objeto da classe Valores. Este problema foi observado em função de ambas a threads acessarem o objeto ao mesmo tempo. Ou seja, enquanto a thread *server* estivesse lendo os dados a thread comLabView poderia alterá-los, o que tornaria parte dos dados exibidos no *website* desatualizados.

Este problema foi solucionado utilizando-se um semáforo, que é uma variável especial utilizada no controle de acesso a recursos compartilhados. Esta variável irá bloquear uma thread que tente acessar os dados caso a outra thread esteja acessando-os. São utilizados os seguintes métodos (observar suas utilizações no quadro 3 do apêndice):

- **semaforo.acquire()**: Que irá bloquear outra thread que tente acessar os dados;
- **semaforo.release()**: Que desbloquear alguma thread que tenha sido bloqueada.

4.4 CONTROLE

Para testar se o controlador/indicador seria capaz de controlar um processo foi utilizado um potenciômetro para simular a entrada da célula de carga e os resultados

do controle exibidos no website. Para estes testes utilizou-se as seguintes configurações:

- *Set point* 1 em 50,00;
- Saída 1 no modo Hi;
- *Set point* 2 em 40,00;
- Saída 2 no modo Lo;

Assim, a comunicação com o *website* foi iniciada com valor lido do potenciômetro acima do *set point* 1, tornando sua saída ativa, e a saída 2 desligada, figura 34.

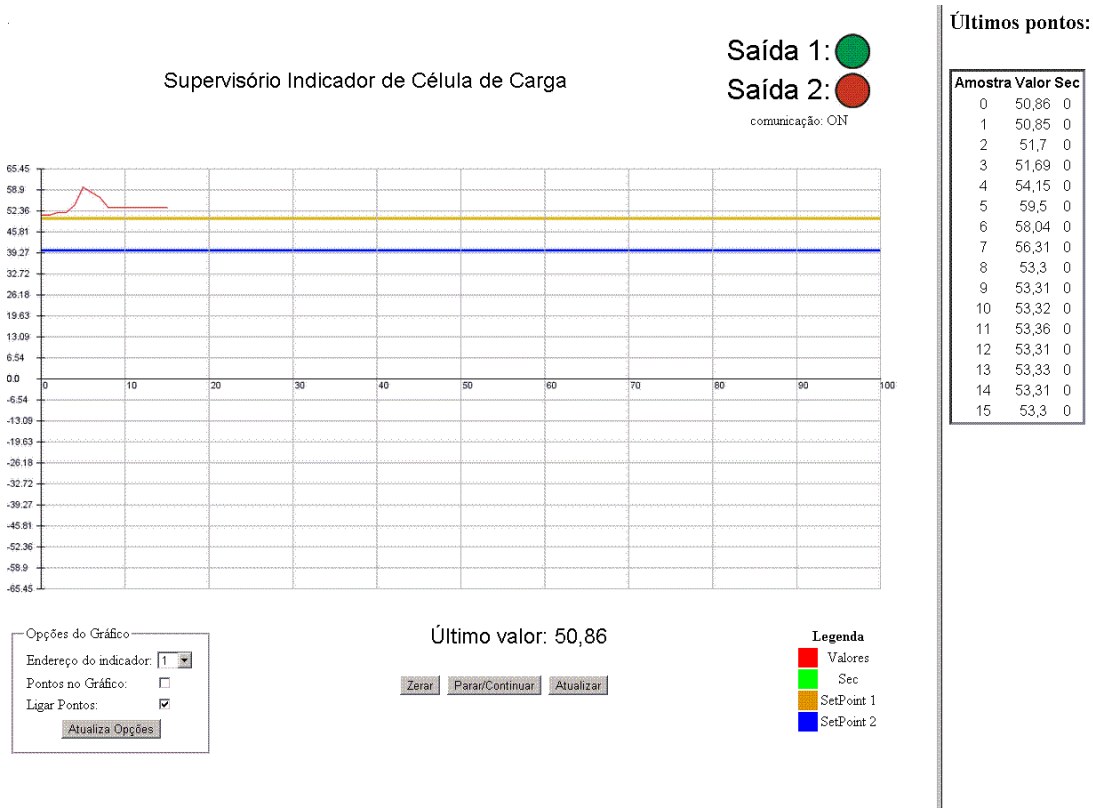

Figura 34 - Página com saída a relê 1 ativa.

Em seguida reduziu-se o valor lido, de tal forma que ficasse entre os valores de *set point* 1 e 2, fazendo com ambas saída estejam desligadas, figura 35.

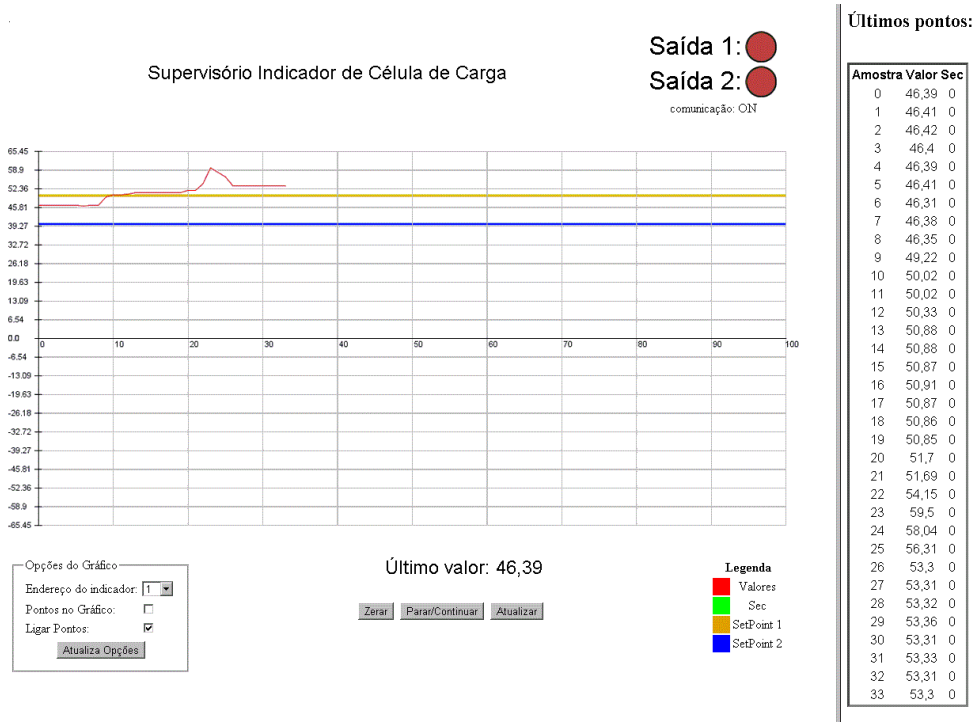

Figura 35 - Saída a relê desligadas.

Finalmente, reduziu-se o valor lido fazendo com que ficasse abaixo do *set point* 2, desativando a respectiva saída, figura 36.

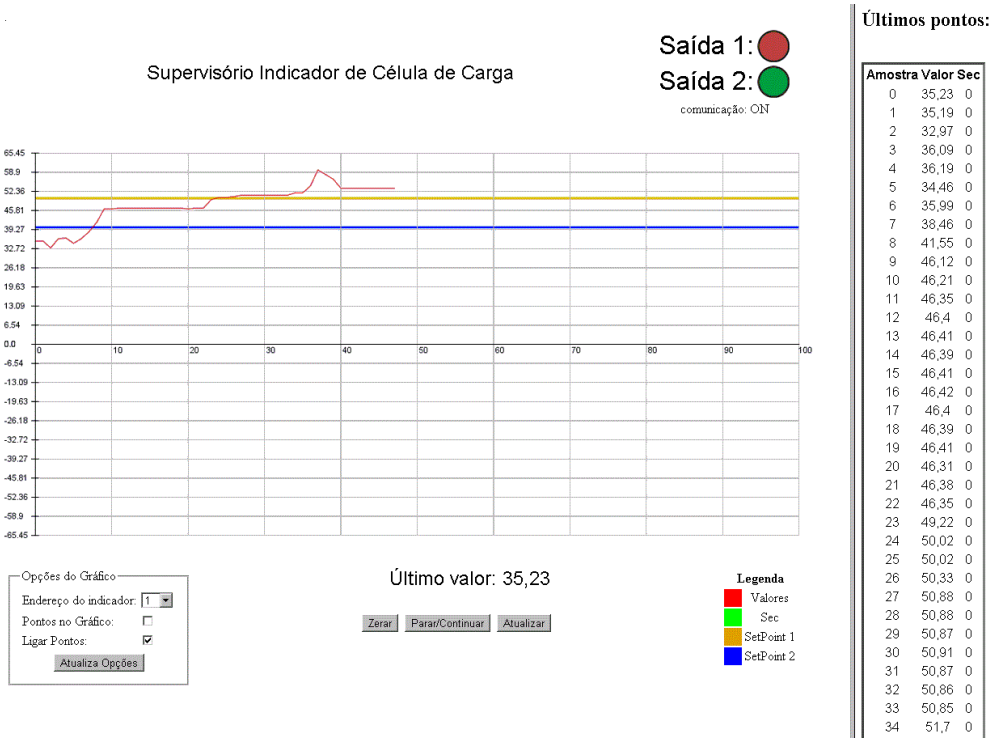

Figura 36 - Saída 1 desliga e Saída 2 ativada.

5. CONCLUSÃO

O objetivo deste projeto foi o desenvolvimento de um sistema capaz de controlar e supervisionar um processo baseado nos dados recebidos de um sensor do tipo célula de carga.

O controlador mostrou-se capaz de utilizar as saídas a relê para controlar um processo, já que elas são ativadas e desativadas conforme foram programadas para funcionarem.

A comunicação RS485 funcionou com baixa perda de dados, conforme pode-se observar na Tabela 2, mesmo assim notou-se uma deficiência nessa comunicação em função da baixa taxa de comunicação utilizada entre o controlador e o supervisor *LabView* (9600bps), o que limita a atualização dos pontos no *website*. Como uma solução para esse problema, o microcontrolador poderia ser trocado para outro que possibilitasse o uso de um cristal de maior freqüência e conseqüentemente ofereceria maior taxa de comunicação. Uma segunda solução seria manter o microcontrolador e alterar o valor do cristal do oscilador utilizado no projeto para um valor adequado às taxas de comunicações exatas.

A saída 4mA a 20mA funcionou com um erro máximo de 0,25%, mostrando-se com boa precisão e apropriada para o tipo de implementação.

Devido a necessidade da utilização de transdutores de força para o funcionamento de processos industriais, células de carga tornaram-se essenciais em tais processos. Assim para torná-los mais eficientes é necessária a automação destes sistemas o que resultará na diminuição da presença do homem nestes processos.

Além disto, como a presença do homem vem diminuindo cada vez mais no chão de fábrica, torna-se necessário que tais processos possam ser monitorados a distância para que eventuais falhas possam detectadas com maior eficiência. O servidor web foi capaz de suprir tal necessidade disponibilizando os dados transmitidos a ele.

REFERÊNCIAS BIBLIOGRAFICAS

1. **Technology, Linear.** LTC2440 datasheet. Disponível em: <http://cds.linear.com/docs/Datasheet/2440fd.pdf>. Acesso em: 4 de novembro de 2011.

2. **Corporation, ATMEL**. AT24C16 datasheet. Disponível em: <http://www.atmel.com/dyn/resources/prod_documents/doc3256.pdf> . Acesso em: 4 de novembro de 2011.

3. **Inc., Motorola.** SPI Block Guide V03.06. Disponível em: <http://www.ee.nmt.edu/~teare/ee308l/datasheets/S12SPIV3.pdf>. Acesso em: 4 de novembro de 2011.

4. **Carer, Maurício e Carraro, Edver**. Célula de carga. Disponível em: <http://hermes.ucs.br/ccet/demc/vjbrusam/inst/cel61.pdf> . Acesso em: 4 de novembro de 2011.

5. **Instruments, National**. Introduction to NI LabView. Disponível em: <http://www.ni.com/gettingstarted/labviewbasics/>. Acesso em: 4 de novembro de 2011.

6. **Semiconductor, National**. LM231 datasheet. National <http://www.national.com/ds/LM/LM231.pdf>. Disponível em:[Acesso em: 4 de novembro de 2011.]

7. **Modbus-IDA**. MODBUS Application Protocol Specification V1.1b. Disponível em: <http://modbus.org/docs/Modbus_Application_Protocol_V1_1b.pdf>. Acesso em: 4 de novembro de 2011.

APÊNDICE

QUADRO 1 - Comunicação SPI com ADC

```
        clr     CS_ADC
        clr     SCK
        setb    SCK
        jb      SDO,FIM_CONV   ;Se 1 a conversao nao terminou
        clr     SCK
        setb    SCK
        jb      SDO,FIM_CONV   ;SEGUNDO BIT EH ZERO
        clr     SCK
        setb    SCK
        mov     C,SDO          ;Recebe o sinal da conversão
        mov     SINALADC,C
        mov     X,#MSB         ;PRIMEIRO BYTE
        LCALL    RECEBE_BYTE_AD
        mov     X,#MLSB        ;SEGUNDO BYTE
        LCALL    RECEBE_BYTE_AD
        mov     X,#LSB         ;TERCEIRO BYTE
        LCALL    RECEBE_BYTE_AD
        clr     SCK            ;Quatro últimos bytes jogados fora
        setb    SCK
        clr     SCK
        setb    SCK
        clr     SCK
        setb    SCK
        clr     SCK
        setb    SCK
        jb      SINALADC, FIM_CONV    ;Caso sinal é negativo inverte os bits
        mov     A,MSB
        cpl     A
        mov     MSB,A
        mov     A,MLSB
        cpl     A
        mov     MLSB,A
        mov     A,LSB
        cpl     A
        mov     LSB,A
FIM_CONV:
        setb    CS_ADC
        ret
RECEBE_BYTE_AD:
        mov     Y,#08h         ;Ler 8 bits
        clr     A
LE_BIT_AD:
        clr     SCK            ;Pulso de clock
        setb    SCK
        mov     C,SDO          ;Le o bit
        mov     ACC.0,C        ;Salva o bit na posicao zero do acumulador
        RL      A              ;Rotacionado o acumlador
        djnz    Y,LE_BIT_AD    ;Vai ler próximo bit
        mov     @R0,A          ;Salva o byte lido
FIM_RECEBE_BYTE_AD:
        ret
```

QUADRO 2 - Cálculo da frequência para saída 4mA a 20mA

```
PUT_4A20:
        mov     X, #DIVISOR
        mov     @X, RET_4A20
        inc     X
        mov     @X, RET_4A20+1
        inc     X
        mov     @X, RET_4A20+2
        inc     X
        mov     @X, #00
        mov     DIVIDENDO, #0ffh
        mov     DIVIDENDO+1, #0ffh
        mov     DIVIDENDO+2, #0ffh
        mov     DIVIDENDO+3, #07fh
        lcall   DIV32
        mov     A, QUOCIENTE+2
        jnz     J1_PUT_4A20
        mov     A, QUOCIENTE+3
        jnz     J1_PUT_4A20
        clr     C
        clr     A
        subb    A, QUOCIENTE
        mov     RCAP2L, A
        clr     A
        subb    A, QUOCIENTE+1
        mov     RCAP2H, A
        sjmp    FIM_PUT_4A20
J1_PUT_4A20:
        mov     RCAP2L, #00
        mov     RCAP2H, #00
FIM_PUT_4A20:
        ret
```

QUADRO 3 - Trecho da Thread comLabView

```
String tmp = input.readLine(); //read from the stream
output.writeBytes(v.getEnd().toString());
System.out.println(tmp);
tmp = tmp.replace(",",".");
String[] vString = tmp.split(" +");
if( vString.length != 7 )
        v.comm=false;
else{
                semaforo.acquire();
                v.insere(vString[1],vString[2]);
                v.setSaida1(Double.parseDouble(vString[2]));
                v.setSaida2(Double.parseDouble(vString[3]));

                if(vString[4].compareToIgnoreCase("1") == 0){
                        v.setAlarme1(true);
                }else{
                        v.setAlarme1(false);
                }if(vString[5].compareToIgnoreCase("1") == 0){
                        v.setAlarme2(true);
                }else{
                        v.setAlarme2(false);
                }
                v.comm=true;
                semaforo.release();
```

QUADRO 4 - Classe GeraGraf

```
public class GeraGraf {
    final static int maxCharHeight = 15;
    final static int minFontSize = 6;

    static BufferedImage grid;
    static FontMetrics fontMetrics;
    static Graphics2D gc;
    static int offset = 40;
    static boolean fPoints=false,linhas=true;;

    static FontMetrics pickFont(Graphics2D g2,
                        String longString,
                        int xSpace) {
        boolean fontFits = false;
        Font font = g2.getFont();
        FontMetrics fontMetrics = g2.getFontMetrics();
        int size = font.getSize();
        String name = font.getName();
        int style = font.getStyle();

        while ( !fontFits ) {
            if ( (fontMetrics.getHeight() <= maxCharHeight)
                 && (fontMetrics.stringWidth(longString) <= xSpace) ) {
                fontFits = true;
            }
            else {
                if ( size <= minFontSize ) {
                    fontFits = true;
                }
                else {
                    g2.setFont(font = new Font(name,
                                               style,
                                               --size));
                    fontMetrics = g2.getFontMetrics();
                }
            }
        }
        return fontMetrics;
    }
    static    public    void    paint(Double    vetorOri1[],Double    vetorOri2[],    int
tamanhoReal,Double saida1, Double saida2) {
        int width = 1080, height = 580;

        int gridWidth = (width-offset*2) /(vetorOri1.length-1);
        int gridHeight = height ;
        Double celMax = 100.0, celMin = 0.0;
        grid = new BufferedImage(width,height,BufferedImage.TYPE_INT_RGB  );
        gc = grid.createGraphics();
        gc.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
                            RenderingHints.VALUE_ANTIALIAS_ON);
        gc.fill(new Rectangle2D.Double(0, 0, width, height));
        fontMetrics = pickFont(gc, "Filled and Stroked GeneralPath",
                               width);
        int x = 0;
        int y = 7;
        int rectWidth = gridWidth ;
        int altGraf = height/2 - offset;
        int stringY = (gridHeight)/2 + fontMetrics.getAscent();
        int rectHeight = stringY - fontMetrics.getMaxAscent() - 2;

        Double vetorRes1[] = new Double[vetorOri1.length];
```

```
        Double vetorRes2[] = new Double[vetorOri2.length];
        Double maior = 0.0;
        for (Integer i=0;i<vetorOri1.length;i++){
               if( maior < Math.abs(vetorOri1[i]) )
                      maior = Math.abs(vetorOri1[i]);
        }for (Integer i=0;i<vetorOri2.length;i++){
               if( maior < Math.abs(vetorOri2[i]) )
                      maior = Math.abs(vetorOri2[i]);
        }if(maior < Math.abs(saida1))
               maior = Math.abs(saida1);
        if(maior < Math.abs(saida2))
               maior = Math.abs(saida2);
        maior*=1.1;

        for (Integer i=0;i<vetorOri1.length;i++){
               vetorRes1[i] = height/2 - vetorOri1[i]*altGraf/maior;
        }
        for (Integer i=0;i<vetorOri1.length;i++){
               vetorRes2[i] = height/2 - vetorOri2[i]*altGraf/maior;
        }
        saida1 = height/2 - saida1*altGraf/maior;
        saida2 = height/2 - saida2*altGraf/maior;

        //Desenho a base do gráfico
        gc.setPaint(Color.black);
        for (Integer i=0;i<=10;i++){
               //Marcas da abcissa
               gc.draw(new   Line2D.Double(offset-5,   height/2+i*altGraf/10,   offset+5,
height/2+i*altGraf/10));
               gc.draw(new   Line2D.Double(offset-5,   height/2-i*altGraf/10,   offset+5,
height/2-i*altGraf/10));
               //Valores na abcissa
               gc.drawString(Double.toString(((int)((-(maior*i)/10)  *  100))/100.0),  0,
height/2+i*altGraf/10+fontMetrics.getAscent()/3);
               gc.drawString(Double.toString(((int)(((maior*i)/10)  *  100))/100.0),  0,
height/2-i*altGraf/10+fontMetrics.getAscent()/3);
               gc.setPaint(Color.lightGray);
               //Divisoes de altura cinza
               gc.draw(new  Line2D.Double(offset+5,  height/2+i*altGraf/10,  width-offset,
height/2+i*altGraf/10));
               gc.draw(new  Line2D.Double(offset+5,  height/2-i*altGraf/10,  width-offset,
height/2-i*altGraf/10));
               gc.setPaint(Color.black);
        }
        //Desenha os SP
        gc.setPaint(Color.orange);
        gc.draw(new     Line2D.Double(offset,     saida1.intValue()-1,     width-offset,
saida1.intValue()-1));
        gc.draw(new      Line2D.Double(offset,      saida1.intValue(),      width-offset,
saida1.intValue()));
        gc.draw(new     Line2D.Double(offset,     saida1.intValue()+1,     width-offset,
saida1.intValue()+1));
        gc.setPaint(Color.blue);
        gc.draw(new     Line2D.Double(offset,     saida2.intValue()-1,     width-offset,
saida2.intValue()-1));
        gc.draw(new      Line2D.Double(offset,      saida2.intValue(),      width-offset,
saida2.intValue()));
        gc.draw(new     Line2D.Double(offset,     saida2.intValue()+1,     width-offset,
saida2.intValue()+1));
        gc.setPaint(Color.black);

        // Desenho as curvas e numeracao do abcissa
        gc.setPaint(Color.black);
        for(Integer i=0;i<vetorRes1.length-1;i++){
```

```
                if( i%10 == 0 ){              //Retas verticais da grade
                        gc.setPaint(Color.black);
                        gc.drawString(i.toString(), x+offset+3, stringY);
                        gc.setPaint(Color.lightGray);
                        gc.draw(new         Line2D.Double(x+offset,         height-offset,
x+offset,offset));
                }
                if(i+1<tamanhoReal){          //Até o ultimo ponto lido
                        if(linhas){
                                gc.setPaint(Color.red);
                                gc.draw(new    Line2D.Double(x+offset,vetorRes1[i],    x    +
gridWidth +offset, vetorRes1[i+1]));
                                gc.setPaint(Color.green);
                                gc.draw(new    Line2D.Double(x+offset,vetorRes2[i],    x    +
gridWidth +offset, vetorRes2[i+1]));
                        }
                }
                x += gridWidth;
        }
        //Ultima linha vertical da grade
        gc.setPaint(Color.lightGray);
       gc.draw(new Line2D.Double(width-offset, height-offset, width-offset,offset));
       gc.setPaint(Color.black);
        gc.drawString(((Integer)(vetorRes1.length-1)).toString(), x+offset, stringY);
        gc.draw(new Line2D.Double(0+offset, height/2, width-offset, height/2));
        gc.draw(new Line2D.Double(0+offset, offset, 0+offset, height-offset));

        if(fPoints){
                for(int i=0;i<tamanhoReal;i++){
                        gc.setPaint(Color.red);
                        drawMark((1.0)*(i*gridWidth),vetorRes1[i]);
                        gc.setPaint(Color.green);
                        drawMark((1.0)*(i*gridWidth),vetorRes2[i]);
                }
        }try{
                File file = new File("graf.gif");
                ImageIO.write(grid, "gif", file);
        }catch(IOException e){
        }
    }
    private static void drawMark(Double X, Double Y){
            gc.draw(new Line2D.Double(offset+X-2, Y-2, offset+X+2, Y+2));
            gc.draw(new Line2D.Double(offset+X+2, Y-2, offset+X-2, Y+2));
    }
    public static void setFPoints(){
            fPoints=true;
    }
    public static void clearFPoints(){
            fPoints=false;
    }
    public static boolean getFPoints(){
            return fPoints;
    }
    public static void setLinhas(){
            linhas=true;
    }
    public static void clearLinhas(){
            linhas=false;
    }
    public static boolean getLinhas(){
            return linhas;
    }
}
```