CENTRO UNIVERSITÁRIO DE BRASÍLIA – UniCEUB

FAET - FACULDADE DE CIÊNCIAS EXATAS E DE TECNOLOGIA
ENGENHARIA DA COMPUTAÇÃO

JOÃO MARCELO DE MORAES TONELLI

MONITORAMENTO E CONTROLE DE TEMPERATURA E UMIDADE DE AMBIENTES

Brasília

2009

JOÃO MARCELO DE MORAES TONELLI

MONITORAMENTO E CONTROLE DE TEMPERATURA E UMIDADE DE AMBIENTES

Trabalho apresentado ao Centro Universitário de Brasília – UniCEUB, como parte das exigências para conclusão do Curso Engenharia da Computação
Orientadora: Professora Maria Marony S. F. Nascimento

Brasília

2009

TONELLI, João Marcelo de Moraes

MONITORAMENTO E CONTROLE DE TEMPERATURA E UMIDADE DE AMBIENTES.

João Marcelo de Moraes Tonelli – Brasília, 2009.

102 p.

Trabalho de conclusão de curso (Monografia) – Centro Universitário de Brasília, 2009.

Aos meus pais Marcelo e Consolação, que me ensinaram as primeiras lições e, com certeza as mais importantes, que foram a torcida mais fiel a cada novo desafio, além da confiança e apoio que sempre me deram.

A Fernanda, pelo carinho e companheirismo.

A Elisângela pelo amor e carinho e por estar presente nesta importante conquista.

AGRADECIMENTO

A Deus, luz e força durante toda a caminhada; a minha família pela compreensão, apoio e carinho; aos profissionais que protagonizaram junto comigo todos os momentos de aprendizagem e crescimento; aos meus amigos que caminharam junto nessa jornada e aos Professores Francisco Javier e Maria Marony pela colaboração e ajuda, fundamentais para conclusão desse projeto.

"A tecnologia digital é a arte de criar necessidades desnecessárias que se tornam absolutamente imprescindíveis".

Joelmir Beting

RESUMO

A temperatura associada à baixa umidade do ar influencia significativamente na vida do homem, ocasionando uma sensação de desconforto e problemas respiratórios que interferem na saúde e na produtividade das pessoas. Neste projeto é apresentado o desenvolvimento de um protótipo automatizado que faz o monitoramento da temperatura e da umidade do ar em ambientes internos, acionando um dispositivo para ligar e desligar os aparelhos de ar condicionado e umidificador de ar dentro da média ideal para torná-lo agradável e propício para a saúde do homem. O projeto trata da criação de um protótipo dividido em três módulos fisicamente ligados entre si através do protocolo RS-485. Os dados são trocados através do protocolo ModBus do tipo mestre/escravo que permite que somente um dispositivo (mestre) inicie as transações, enquanto os outros dispositivos (escravos) respondem de acordo com o pedido do mestre. O primeiro módulo (CPU Sensor) é responsável pela coleta de dados; o segundo (CPU Principal) solicita a coleta ao CPU Sensor, compara os valores e os envia ao CPU Reles; o terceiro (CPU Reles) é o responsável pelo acionamento e desligamento dos aparelhos de ar condicionado e umidificador de ar. Para construção do protótipo foi utilizado um sensor SHT11, e 03 três microprocessadores da família ATMEGA de 8 (oito) bits programados em linguagem C. Os produtos utilizados atenderam as exigências de controle, distribuição, armazenamento de informações e a necessidade de facilidade para interoperabilidade entre os módulos, visando à eficiência, flexibilidade e confiabilidade dos sistemas.

Palavras-chave: Temperatura. Umidade do ar. Microcontroladores.

ABSTRACT

The temperature associated with low humidity of air influences significantly the human life, causing a feeling of discomfort and respiratory problems which affects health and productivity of people. This project is developing an automated prototype that control the temperature and the air humidity of intern places, setting in motion a device to turn on and turn off the air-conditioners and humidifier air looking for the ideal average for make it pleasant and conducive to human health. The project deals with the creation of a prototype physically divided into three interlinked modules via RS-485 protocol. Data is trade via the Protocol ModBus master/slave type that allows only one device (master) to start transactions, while other devices (slaves) respond in accordance with the request of the master. The first module (CPU_Sensor) is responsible for data collection; the second (CPU_Principal) asks collection to CPU_Sensor, compares the values and sends it to the CPU_AcionaRele; the third (CPU_AcionaRele) is responsible for turn on and shutdown the air-conditioners and humidifier air. The prototype was constructed with a sensor called SHT11, and three ATMEGA family microprocessors 8 (eight) bits programmed in language C. The products reached what was expected by the exigencies of controls, distribution, storage of information and the need to make easy the interoperability between modules, for efficiency, flexibility and reliability of the systems.

Keywords: Temperature. Air humidity. Microcontrollers.

LISTA DE SIMBOLOS/DEFINIÇÕES

ABORL – CCF - Associação Brasileira de Otorrinolaringologia e Cirurgia Cérvico-Facial.

BTU – British Thermal Unid

CRC – Controle de Redundância Cíclica

DATASUS – Departamento de Informática do SUS

EIA – Eletronics Industry Association

PCL – Controladores Lógicos Programáveis

RTU – Remote Terminal Unit

RS – Recommended Standard

SUS – Sistema Único de Saúde

Ω - Ohm

INDICE DE FIGURAS

INDICE DE TABELAS

SUMÁRIO

Capítulo 1

APRESENTAÇÃO

1.1 JUSTIFICATIVA

Diariamente, ouvimos falar nos telejornais em umidade relativa do ar e sua influência em nossa sensação de conforto e bem-estar. De acordo com o dicionário Larousse (2001, p 1005), a palavra umidade é definida como “estado ou qualidade do que é úmido; orvalho; teor de vapor d’água existente na atmosfera”. A quantidade de água ou vapor d’água existente no ar é expressa como umidade absoluta ou umidade relativa. O higrômetro é o aparelho utilizado para medir a umidade do ar.

O ser humano é muito sensível a umidade e sua saúde pode ser comprometida seriamente quando essa umidade está muito baixa ou muito alta. Quando a umidade encontra-se inferior a 30%, problemas respiratórios, alergias, sinusites, asmas e outras doenças tendem a se agravar, além de haver riscos de incêndios em áreas vegetadas. Quando a umidade está muito alta podem surgir fungos, bolores, mofos e ácaros, além de agravar as crises reumáticas e danificar aparelhos eletrônicos.

De acordo com os alertas da defesa civil, amplamente divulgados pela mídia, a umidade relativa do ar ideal para que o ser humano se sinta confortável é por volta de 55% a 80%. Eles aconselham o uso de umidificadores ou desumidificadores para manter esse nível.

No Distrito Federal, principalmente durante os meses de agosto e setembro, a temperatura associada à baixa umidade do ar influencia significativamente na vida de seus habitantes, ocasionando uma sensação de desconforto e problemas respiratórios que aumentam o número de atendimentos nos hospitais locais, além de diminuir sua produtividade em trabalhos intelectuais ou físicos. Sendo assim, é necessário a implementação de recursos tecnológicos que melhorem a qualidade ambiental dos espaços internos de trabalho ou convivência. Durante esse período é comum o uso de umidificadores de ar ou até mesmo o uso de bacias com água fria ou toalhas

molhadas para amenizar o calor no interior das casas, apartamentos ou local de trabalho. De acordo com Aquino Neto e Gionda (2003), é possível perceber ganhos na produtividade sempre a qualidade do ambiente interior melhora.

Esse projeto propõe a construção de um sistema de monitoramento da temperatura e da umidade de ar em ambientes internos. A partir da coleta de dados (variação da temperatura e da umidade no decorrer do tempo), um dispositivo é acionado para ligar o ar condicionado e o umidificador de ar visando manter a temperatura e a umidade dentro da média ideal para que o ambiente fique agradável e de acordo com os percentuais indicados pela Defesa Civil.

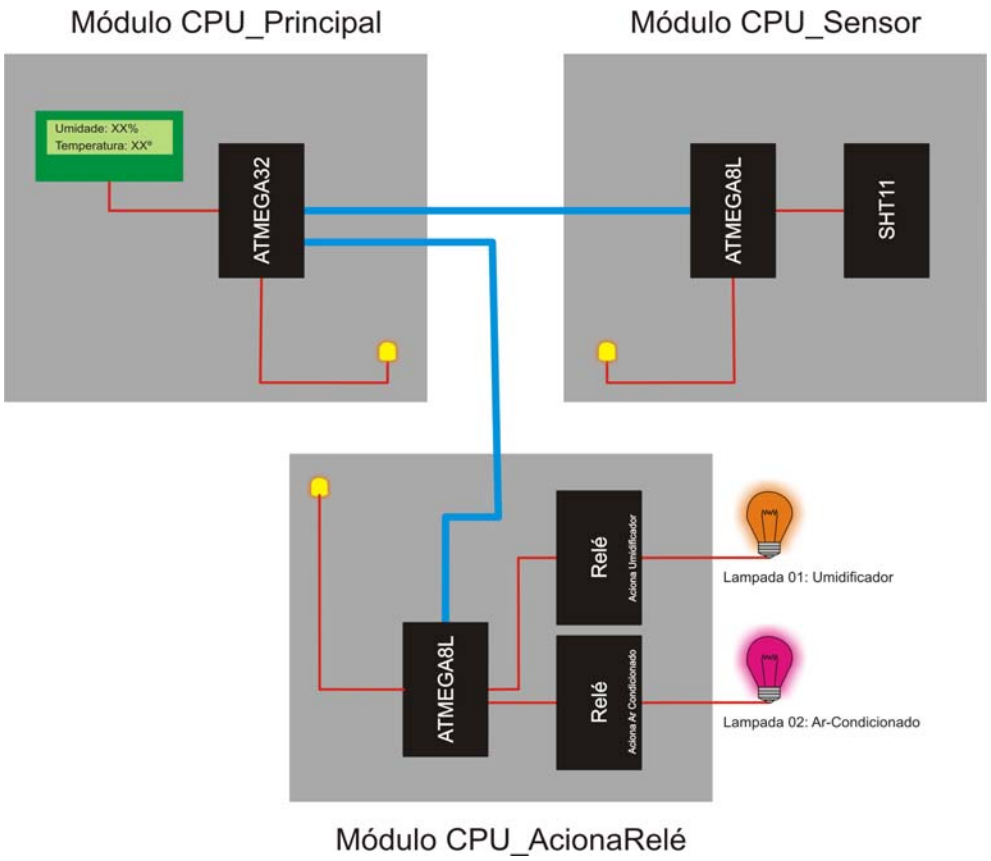


FIGURA 1.1: APRESENTAÇÃO GERAL DO PROTÓTIPO

Para a realização do projeto (figura1.1) foram utilizados 03 (três) microcontroladores Atmel de 8 bits , que são programados em linguagem C.

Cada microcontrolador está presente em um módulo. Cada módulo se comunica um com o outro por meio do padrão MODBUS e do protocolo RS-485 responsável pela comunicação física do sistema. Os dados são coletados através de um sensor SHT11, presente em um dos módulos, que envia os valores, de temperatura e umidade, para o módulo principal que checará os valores, e emitirá um sinal para o outro módulo responsável por ligar e/ou desligar os equipamentos responsáveis pela climatização do ambiente.

1.2 OBJETIVOS

Objetivo Geral

Desenvolvimento de um sistema automatizado visando manter a temperatura e a umidade relativa do ar em ambientes internos dentro dos níveis recomendados pela defesa civil, tornando esses ambientes mais agradáveis e propícios para saúde do homem.

Objetivos Específicos

- Aperfeiçoar os recursos oferecidos pelos aparelhos de ar condicionado e umidificadores de ar utilizados em residências e ambientes de trabalho;
- Demonstrar a viabilidade da utilização de um sistema de microcontrolador para manter estável a temperatura e a umidade relativa do ar em ambientes internos.

1.3 ESTRUTURA DA MONOGRAFIA

O trabalho teórico foi organizado da seguinte forma:

O Capítulo 1 faz uma apresentação geral da monografia trazendo a justificativa e os objetivos para elaboração do projeto.

O Capítulo 2 aborda os aspectos teóricos da umidade relativa do ar, temperatura e instrumentos utilizados para medí-las.

O Capítulo 3 apresenta as especificações técnicas dos componentes utilizados para construção do protótipo bem como os pontos relativos à comunicação dos módulos: o protocolo MODBUS e suas características técnicas, comandos, modos de transmissão (RTU); campo checksum (CRC); o padrão de comunicação RS 485: transceptor MAX 485, sensor SHT11, microcontroladores Atmega (Atmega 8L e Atmega 32 16PC).

O Capítulo 4 traz o processo de implementação do protótipo e apresentação dos testes realizados.

O Capítulo 5 traz as conclusões e sugestões para projetos futuros na área.

Capítulo 2

UMIDADE RELATIVA DO AR, TEMPERATURA E INSTRUMENTOS UTILIZADOS PARA MEDIÇÃO.

2.1 A UMIDADE DO AR

O ar atmosférico contém uma quantidade de vapor d'água que varia de acordo com a temperatura do ar e com a disponibilidade de água na superfície terrestre de determinada localidade. Esse vapor, resultado da evaporação da água do solo, da transpiração das plantas, dos mares, rios e lagos, sobe para a atmosfera pela ação do calor, tornando-se um de seus principais componentes.

De acordo com Ayoade (1998, p. 128), "embora o vapor d'água represente somente 2% da massa total da atmosfera e 4% de seu volume, ele é o componente atmosférico mais importante na determinação do tempo e do clima". Salienta ainda que:

> O vapor d'água é de grande significado por diversas razões, de modo que os meteorologistas e os climatólogos estão interessados em sua quantidade e em sua distribuição no tempo e no espaço. [...] A quantidade de vapor d'água num certo volume de ar é uma indicação da capacidade potencial da atmosfera para produzir precipitação. Pode também absorver tanto a radiação solar quanto terrestre e, assim, desempenha papel de regulador térmico no sistema Terra-atmosfera, exercendo, em particular, grande efeito sobre a temperatura do ar e a temperatura sentida pela pele humana e, conseqüentemente, o seu conforto.

A umidade do ar é o termo empregado pelos cientistas e estudiosos para descrever a quantidade de vapor d'água que existe em certo momento na atmosfera e a quantidade máxima que ela pode conter (em torno de 4%). Entretanto, este termo não abrange as outras formas nas quais a água pode estar presente na atmosfera, seja de forma líquida (gotículas) ou sólida (gelo).

Ayoade (1998) afirma que há várias formas para se medir o conteúdo de umidade da atmosfera, cujos índices de umidade geralmente utilizados são:

1. Umidade absoluta: expressa em gramas por metro cúbico de ar e é a massa total de água num dado volume de ar;
2. Umidade específica: é a massa de vapor d'água por quilograma de ar;
3. Índice de umidade: é a massa de vapor d'água por quilograma de ar seco;
4. Umidade relativa: é a razão entre o conteúdo real de umidade de uma amostra de ar e a quantidade de umidade que o mesmo volume de ar pode conservar na mesma temperatura e pressão quando saturado. É normalmente expressa em forma de porcentagem;
5. Temperatura do ponto de orvalho: é a temperatura na qual ocorrerá saturação se o ar se esfriar a uma pressão constante, sem aumento ou diminuição de vapor d'água;
6. Pressão vaporífica: é a pressão exercida pelo vapor contida na atmosfera em milibares.

Entretanto, a umidade relativa do ar é a mais conhecida pela população. A umidade relativa é a comparação da umidade real, verificada em aparelhos como o higrômetro, e o valor teórico, estimado para aquelas condições, podendo variar de 0% (ausência do vapor de água no ar) a 100% (quantidade de vapor de água que o ar pode dissolver, indicando que o ar está saturado).Esse índice é amplamente influenciado pela temperatura do ar e varia de acordo com as mudanças dessa temperatura.

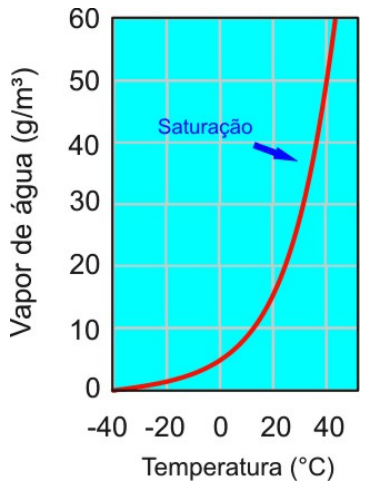


FIGURA 2.2 – PONTO DE SATURAÇÃO

Fonte: br.geocities.com/.../leituras/Umidade12.gif

O ar contém certa quantidade de vapor d'água. Quando atinge esse limite, diz-se que está saturado. A quantidade máxima desse vapor aumenta com a temperatura.

Para Ayoade (1998, p.144), é importante lembrar que:

> Nos trópicos, onde as variações diurnas na temperatura são grandes, há variações consideráveis na umidade ao longo do dia. Em muitas partes dos trópicos úmidos, particularmente nas áreas costeiras, a umidade relativa pode muitas vezes estar próxima de 100% à noite, durante a estação chuvosa. A umidade relativa atinge seu valor mínimo à tarde durante a estação seca, nos interiores continentais dos trópicos. Há também variações sazonais nos valores da umidade relativa nas baixas latitudes. As variações sazonais são mínimas no Equador e crescem com o aumento em latitude.

No Distrito Federal, principalmente durante os meses de agosto e setembro, percebe-se a massa de ar seco que cobre a região, com índices de umidade relativa inferiores a 20%, conforme os alertas da defesa civil divulgados na mídia.

2.1.1 INSTRUMENTOS PARA MEDIÇÃO DA UMIDADE RELATIVA DO AR

HIGRÔMETRO

Higrômetro é o instrumento utilizado para medir a umidade do ambiente. Muito utilizado em estudos relacionados ao clima, pode também ser empregado em ambientes fechados onde a umidade excessiva ou baixa pode causar danos, como por exemplo, em museus, bibliotecas ou laboratórios.

Esse aparelho é composto, em sua maioria, por substâncias com capacidade de absorver a umidade atmosférica, tais como o cabelo humano e sais de lítio. No caso de usar o cabelo, uma mecha de cabelos é colocada entre um ponto fixo e outro móvel e, de acordo com a umidade a que estiver submetido, ela varia de comprimento, arrastando o ponto móvel. O

movimento é transmitido a um ponteiro que se desloca sobre uma escala, onde estão os valores da umidade relativa do ar.

- **Higrômetro de cabelo:** Esse tipo de higrômetro é utilizado para a medição da umidade do ar. A longitude do fio de cabelo varia de acordo com a umidade ambiente. Essa alteração é indicada como umidade relativa por meio de meios mecânicos.

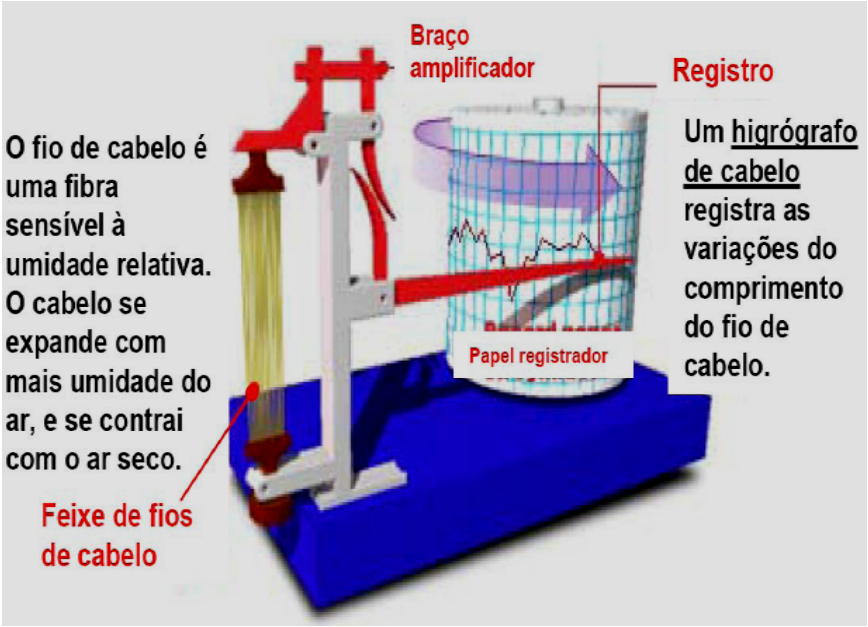


FIGURA 2.3 – HIGRÔMETRO DE MECHAS DE CABELO

Fonte: http://www.dca.iag.usp.br/www/material/humberto

- **Higrômetro de espelho de ponto de orvalho (Dew-Point):** O espelho de ponto de orvalho é um procedimento preciso de medição da umidade relativa do ar, capaz de avaliar a condensação do vapor d'água quando a temperatura abaixa além do ponto de orvalho. A temperatura de uma superfície refletida no espelho se esfria até o ponto onde se inicia a condensação. A temperatura, medida nesse momento por uma termoresistência PT-100, corresponderá à temperatura de ponto de orvalho, da qual se pode calcular a umidade relativa do ar por meio do cruzamento das informações com a medição da saturação e temperatura do ar. Um elemento Peltier é instalado para esfriar e se avalia a superfície refletida, usando-se um procedimento óptico-eletrônico.

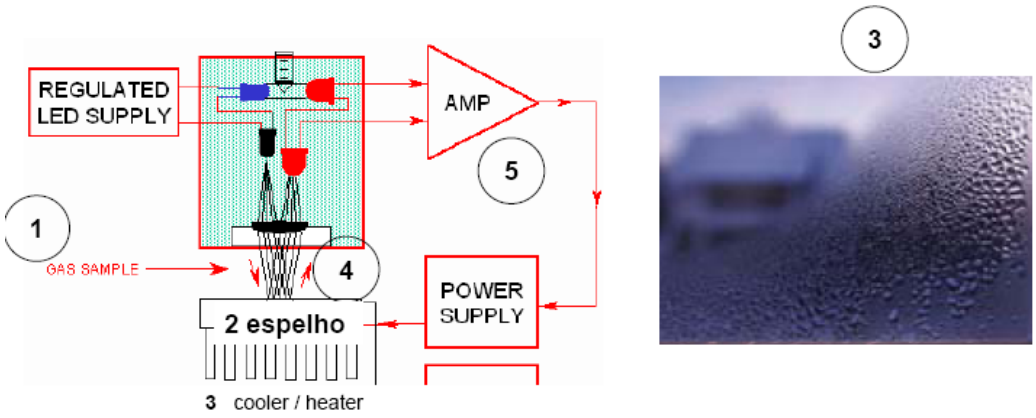


FIGURA 2.4 HIGROMETRO DE ESPELHO DE PONTO DE ORVALHO

Fonte: http://www.dca.iag.usp.br/www/material/humberto

- **Higrômetro de sais de lítio:** baseia-se na variação de condutividade dos sais, que apresentam uma resistência variável de acordo com a água absorvida. Um amperímetro, com uma escala devidamente calibrada fornecerá os valores da umidade do ar.

MEDIÇÃO PSICROMÉTRICA

Além desses modelos há também outras formas de medir a umidade relativa do ar calculando a velocidade de evaporação da água. Nesse caso, utilizam-se dois termômetros idênticos, precisos, expostos ao ar: um com o bulbo descoberto e o outro com o bulbo coberto por uma gaze umedecida. O primeiro medirá o ar ambiente e o segundo, a temperatura do bulbo úmido.

O primeiro termômetro mede o ar ambiente e o segundo a chamada temperatura de bulbo úmido[1]. Os dois termômetros são colocados em uma circulação de ar ou em meio de ar circulante, protegidos de calor radiante. Devido ao calor de evaporação latente, a temperatura do termômetro de bulbo úmido baixa e baixa mais à medida que o ar for ficando mais seco. Depois de 01 ou 02 minutos a temperatura do termômetro de bulbo úmido permanece constante e os valores de medição dos dois termômetros (bulbo úmido e bulbo seco) podem ser lidos.

[1] Fonte: http://www.lufft.com.br/tecnologia/medi%c3%a7%c3%a3%20de%20conceitos%20b%c3%a1sicos.pdf

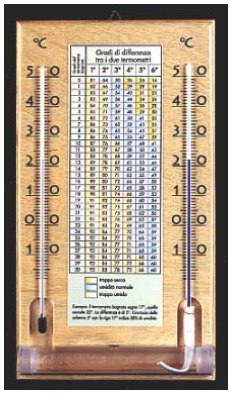


FIGURA 2.5 - PSICROMETRO

Fonte: http://pt.wikipedia.org/wiki/psicr%c3%b4metro

COMPARAÇÃO DAS DIFERENTES TÉCNICAS DE MEDIÇÃO

COMPARAÇÃO DAS DIFERENTES TÉCNICAS DE MEDIÇÃO

TÉCNICA	Medição Mecânica	Psicrômetro	Sensor Capacitivo	Higrômetro de espelho de ponto de orvalho
VANTAGENS	Operação simples; Baixo custo	Preciso; Estabilidade	Grande precisão; Simplicidade; Livre de manutenção; Resposta rápida; Boa relação custo/benefício.	Precisão altíssima; Confiabilidade; Estabilidade; Tempo rápido de resposta.
DESVANTAGENS	Tempo de resposta alto; Custo de manutenção alto; Faixa de medição limitada; Inércia.	Manutenção contínua do sistema; Erros de leitura.	É necessário a compensação de temperatura.	Custo alto para aquisição; Custos para manutenção (limpeza do espelho).

TABELA 1: Quadro comparativo de técnicas de Medição

2.2 TEMPERATURA

O tipo de clima de muitos lugares requer o uso de equipamentos para refrigeração, principalmente, no verão visando garantir a temperatura ambiente ideal capaz de gerar conforto, produtividade, saúde e bem-estar das pessoas.

Segundo a Organização Meteorológica Mundial[2] – OMM (1997), o conforto térmico é um conceito subjetivo associado à sensação térmica considerada agradável ao ser humano. De acordo com esta organização, em termos físicos, "está associado ao estado em que o indivíduo encontra-se em equilíbrio térmico com o ambiente, de forma que seu corpo não necessita despender energia para elevar ou reduzir a temperatura interna". Enfatizam ainda que o conforto térmico dependerá de variáveis do ambiente, tais como a temperatura, a umidade relativa do ar, a velocidade de deslocamento do ar, a pressão barométrica, o calor irradiado por outros corpos próximos além das variáveis de cada pessoa (seu peso, se está em repouso ou em atividade, entre outras).

De acordo com o pneumologista Stelmach (2008), o ideal é que a temperatura esteja entre 18° e 23°, pois abaixo disso o ser humano necessita defender-se contra uma perda exagerada de seu próprio calor, resultando em mal estar, incomodo e falta de conforto. Afirma ainda que no ambiente de trabalho há uma correlação entre a produtividade do homem e a temperatura que o rodeia. O ambiente demasiado quente ou demasiado frio não contribui para a eficiência esperada no trabalho, por isso é fundamental controlar a temperatura ambiente.

Segundo Ayoade (1998, p. 50) a temperatura é um dos elementos mais discutidos do tempo atmosférico. Pode ser definida em termos do movimento de moléculas, de modo que, quanto mais rápido for seu deslocamento, mais elevada será a sua temperatura. Normalmente, é definida em termos relativos tomando-se por base o grau de calor que o corpo possui. Conforme esse autor:

> [...] a temperatura é a condição que determina o fluxo de calor que passa de uma substancia para outra. O calor desloca-se de um corpo que tem uma temperatura mais elevada para outra que tem a temperatura mais baixa. A temperatura de um corpo é determinada pelo balanço entre a radiação que

[2] Disponível em www.inmet.gov.br/html/clima/clima.html. Acesso em 20 de agosto de 2009.

> chega e a que sai e pela sua transformação em calor latente e sensível, entre outros. A temperatura de um corpo é, portanto, o grau de calor medido por um termômetro. Várias escalas são usadas para expressar as temperaturas, tais como: Fahrenheit, a Centígrada, a Kelvin ou a escala de temperatura absoluta.

Grande número de países expressa a temperatura em escala Centígrada ou Celsius.

2.2.1 Termômetro

São os instrumentos utilizados para medir a temperatura dos sistemas físicos. Os mais comuns são aqueles que se baseiam na dilatação do mercúrio. Alguns determinam o intervalo da temperatura mediante o aumento da pressão de um gás ou pela curvatura de uma lâmina bimetálica; outros ainda empregam efeitos elétricos, traduzidos pelo aparecimento de correntes elétricas quando ao ponto de solda de dois metais diferentes é aquecido.

TIPOS DE TERMÔMETROS

- **Termômetro de mercúrio:** É o modelo mais comum que existe. Consiste basicamente de um tubo capilar (fino como um fio de cabelo) de vidro, fechado a vácuo e um bulbo (espécie de bolha arredondada em uma extremidade contendo mercúrio). O mercúrio, como todos os materiais, dilata-se quando aumenta a temperatura. Por ser extremamente sensível, ele aumenta de volume quando ocorre qualquer variação de temperatura, mesmo próxima à do corpo humano. Quando aquecido, o volume de mercúrio se expande no tubo capilar do termômetro. Essa expansão é medida pela variação do comprimento, numa escala graduada que pode ter uma precisão de 0,05°C; quando se observa a variação de temperatura.

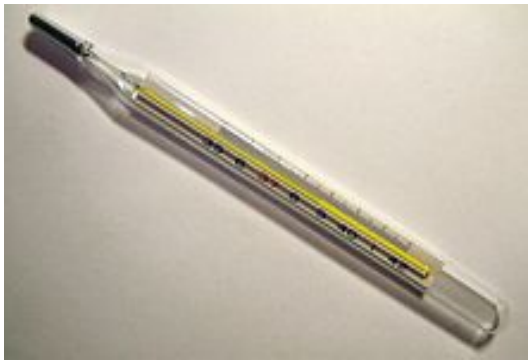


FIGURA 2.6 – TERMÔMETRO DE MERCÚRIO

Fonte: www.instrumentação.net/.../img/termometro6.jpg

- **Termômetros digitais:** São os instrumentos mais utilizados pelas empresas. São destinados a medir a temperatura em processos e produtos diversos que necessitam apenas de uma medição esporádica. Exemplos de aplicações de termômetros digitais para medição de temperaturas em: fundições; alimentos em restaurantes e indústrias; processos químicos; estruturas; fornos; produtos diversos. Em geral, os termômetros digitais podem ter aplicação industrial ou não, para monitoração constante e precisa de temperaturas de determinados equipamentos que sejam sensíveis a alterações de seu funcionamento em função de sua temperatura e/ou ambientes que necessitam de cuidados com a temperatura (conservação de alimentos em freezer de supermercados; laboratórios biológicos que cultivam bactérias ou outras espécies). Esse equipamento é também utilizado na versão com interface de raio infravermelho (INFRAERD), para versão esporádica de temperatura sem contato físico com o objeto (indústrias, formula 1, dentre outros).

FIGURA 2.7 – TERMÔMETRO DIGITAL CLÍNICO

Fonte: ansnafisica.blogspot.com/2009/03/termômetros.html

FIGURA 2.8 – TERMÔMETRO DIGITAL PARA CONSERVAÇÃO DE ALIMENTOS

Fonte: www.koboldmessring.com/.../prid/138/index.html

- **Termômetro infravermelho:** também denominado pirômetro óptico, é um dispositivo que mede a temperatura sem contato com o corpo ou meio do qual se pode conhecer a temperatura. A unidade de infravermelho é sensibilizada pela energia emitida, refletida e transmitida, que for focalizada no detector. O circuito eletrônico converte a energia recebida em uma leitura que é exibida no visor do termômetro[3]. Geralmente, este termo é aplicado a instrumentos que medem temperaturas superiores a 600° Celsius. É utilizado tipicamente, para medição da temperatura de metais incandescentes em fundições.

FIGURA 2.9 – TERMOMETRO INFRAVERMELHO

Fonte: www.imagemrio.com.br/descricao.asp

[3] Fonte: Manual de instruções do termômetro digital infravermelho TD-920

- **Modelos de termômetros por contato**: utilizam pontas sensoras, geralmente intercambiáveis, com modelos diferentes de sensores para cada aplicação.

FIGURA 2.10 - TERMOMETRO POR CONTATO

Fonte: www.impac.com.br/pirometro_optico_lutrn_tm93

- **Termômetros bimetálicos:** São também conhecidos como termômetros metálicos. Baseiam-se no fenômeno da deformação termodinâmica; esse efeito acontece quando uma barra de metal é ligada a outra de coeficientes diferentes, a corrente ao atravessar (ou ser aquecida por chama) irá aquecer o conjunto de forma desigual, resultando dilatações diferentes provocando o arqueamento da barra que poderá ser usado, tanto para abrir ou fechar válvulas; ligar ou desligar circuitos elétricos ou registrar a quantidade de corrente que atravessa a barra. Os do primeiro tipo podem ser construídos de forma semelhante aos termômetros a liquido: uma barra, retilínea ou não, ao dilatar-se, move um ponteiro registrador. Os mais usados e precisos termômetros desse tipo exploram a diferença de dilatabilidade entre materiais como latão e partes de carros, ferro e cobre, dentre outros. Para isso, constroem-se lâminas bimetálicas de forma espiralada que se curvam conforme aumenta ou diminui a temperatura. Nesse movimento, a lâmina arrasta, em sua extremidade, um ponteiro que percorre uma escala graduada ou registra graficamente a variação de temperatura num papel em movimento (termógrafo).

FIGURA 2.11 – TERMOMETRO BIMETÁLICO

Fonte: www.cwapor.com.br/

2.3 INSTRUMENTO PARA VISUALIZAR A TEMPERATURA E A UMIDADE DO AR

- **Termohigrômetro:** É um instrumento que tem dupla função: indica a temperatura e a umidade relativa do ar. O seu campo de aplicação é muito vasto, podendo ser em transporte de alimentos; armazéns de perecíveis; frigoríficos; hospitais, laboratórios, sala de computadores, entre outros.

FIGURA 2.12 - TERMOHIGRÔMETRO

Fonte: infoelec.net/osc/índex.php\/oscsid=dbad0fb4e61

2.4 O CLIMA E A UMIDADE RELATIVA DO AR

A umidade do ar está diretamente ligada à variação de temperatura e a existência de reservatórios de água tais como rios, mares ou lagos nas diferentes regiões do planeta, alterando o índice de umidade de cada localidade de acordo com a presença desses recursos naturais.

Segundo Indriunas (s/d), é nas zonas temperadas que há maior variação da umidade do ar e é onde as estações do ano são mais definidas. Salienta ainda que:

> O relevo e as correntes de ar influenciam de forma marcante a distribuição das chuvas, o índice pluviométrico e, de modo geral, a umidade. No Brasil, as regiões mais úmidas são a amazônica e as litorâneas, decrescendo quanto mais se adentra ao território. O Centro-Oeste é o local que apresenta, junto com uma parte do Sudeste, alguns dos menores índices de umidade relativa do ar do país. A Região Sul possui índices mais altos devido ao seu relevo mais plano, o que facilita a penetração do ar úmido.

As baixas umidades do ar interferem significativamente na agricultura e na pecuária que necessitam de uma grande quantidade de água para seu desenvolvimento. Ainda nesse aspecto, as florestas sofrem constantes ameaças de incêndio, muitas vezes ocasionadas pela prática de queimadas e raios que atingem a vegetação seca.

Outros problemas decorrentes da baixa umidade estão relacionados ao bem estar do homem, tanto em sua saúde física quanto em sua produtividade.

2.5 A INFLUÊNCIA DA UMIDADE DO AR NA VIDA DO HOMEM

O ar contém uma quantidade de vapor d'água que varia de acordo com a temperatura: quanto mais quente o ar, mais vapor d'água ele reterá. Quando ocorre uma baixa umidade significa que o ar está mais seco e poderia reter uma quantidade de água maior naquela temperatura, ocorrência normalmente acentuada no inverno.

Quando a umidade do ar está abaixo da média, o homem sofre, principalmente, com a diminuição da hidratação das vias aéreas e dos olhos. Essa agressão às mucosas que revestem as fossas nasais e vias respiratórias como um todo propicia crises de bronquite e asma, infecções virais e bacterianas. A pele fica ressecada e sensações de desanimo e cansaço se tornam freqüentes.

De acordo com a Organização Mundial de Saúde – OMS (2008), os valores da umidade relativa do ar ideais variam de 55% a 80%. Segundo essa organização,

é fundamental que o homem observe alguns procedimentos que minimizam a influência nociva provocada pela baixa umidade:

- Estado de atenção – umidade do ar entre 20 e 30% - tomar muita água; evitar atividades físicas ao ar livre no horário entre 11h e 15 h; proteger-se do sol em local sombreado ou em áreas com vegetação; utilizar toalhas molhadas, bacias com água ou umidificadores;
- Estado de alerta – umidade do ar entre 12 e 20% - seguir as mesmas orientações anteriores; não fazer atividades físicas entre o horário de 10 h e 16 h; umedecer os olhos com soro fisiológico e evitar aglomerações em locais fechados.
- Estado de emergência – umidade do ar abaixo de 12% - além das orientações já citadas, umedecer os ambientes internos no período de 10 h às 16 h; suspender aulas, cinemas, teatros ou qualquer outra atividade que exija aglomeração de pessoas; interromper qualquer atividade ao ar livre, tais como prática de esportes ou serviços externos.

DOENÇAS DO APARELHO RESPIRATÓRIO

De acordo com os dados fornecidos pelo Ministério da Saúde (2008), a estiagem é um dos principais fatores responsáveis pelo agravamento de quadros de doenças respiratórias, como a asma e a rinite alérgica. Segundo esses dados, a baixa umidade do ar, comum no período da seca, representa um grande tormento para as pessoas portadoras de doenças respiratórias, embora nesse período também aumente a incidência de diarréia virótica, viroses e doenças de pele.

No Distrito Federal e nos estados de Goiás, Mato Grosso e Tocantins e algumas regiões dos estados do Pará, Mato Grosso do Sul, Minas Gerais, Bahia, Maranhão e Piauí, a seca se manifesta com mais intensidade e, conseqüentemente, o número de atendimentos médicos aumentam consideravelmente. Nesses locais, o Ministério da Saúde tem procurado capacitar profissionais de saúde para diagnosticar e tratar essas doenças, inclusive com campanhas de conscientização à população alertando-a para cuidados que amenizem as crises e diminuam o índice de mortalidade. Recomendam a utilização de aspirador de pó para limpeza da casa, ingestão de líquidos, evitar o fumo, manter os ambientes arejados e usar umidificadores, bacias com água ou toalhas molhadas. Recomendam ainda o uso de cremes hidratantes com a pele limpa para evitar lesões, já que esta área fica vulnerável durante esse período. O uso de chapéu, roupas leves, calçados confortáveis constituem como indicações fundamentais.

Segundo o DATASUS (BRASIL, 2008), a asma representou no ano de 2004, a terceira causa de hospitalização pelo SUS, responsabilizaram-se por 396.505 internações em todo o país. No Distrito Federal, em 2005, 4.160 pessoas asmáticas foram internadas, correspondendo a 3% do número de internações.

A PRODUTIVIDADE DO HOMEM

Estudos comprovam que a produtividade do trabalho do homem é muito menor quando o índice de umidade relativa do ar está muito baixo. O organismo humano gasta energia em excesso para alcançar o conforto térmico, quer seja eliminando água por meio da respiração, do suor, da urina ou da transpiração. Conseqüentemente, sua produtividade diminui ou adquiri qualidade inferior.

De acordo com Fanger (2000, apud GIODA & AQUINO NETO, 2003), os trabalhadores de escritórios têm sua produtividade influenciada, significativamente, pela umidade do ar, podendo ser até 6,5% maior em ambientes cujo ar está em melhores condições. Há, portanto, grande incentivo econômico para que a qualidade do ar, em ambientes internos, seja melhorada.

2.6 UMIDIFICADORES DE AR

Umidificador de ar é o equipamento que ajuda a manter a umidade em ambientes fechados, em um nível confortável. De acordo com informações fornecidas pela ABORL – CCF[4], esse aparelho ajuda a minimizar a ardência nos olhos, a irritação na garganta, sangramentos no nariz, conseqüências diretas do ar seco e comuns em determinadas épocas do ano.

Entretanto, segundo essa associação, por ser um aparelho relativamente caro, as pessoas que mais se beneficiam com sua utilização, são aquelas que têm rinite ou asma, já que o ar seco agrava consideravelmente esses problemas.

Salientam ainda que sua aquisição deve ser criteriosa. Segundo eles, o modelo mais indicado é o umidificador sônico, que controla melhor a umidade do

[4] Folha de São Paulo, 24/07/2008

ambiente, uma vez que, os aparelhos mais simples podem deixar o ambiente excessivamente úmido, favorecendo a proliferação de fungos e piorando os quadros das pessoas que possuem alergia respiratória.

FIGURA 2.13 – UMIDIFICADOR DE AR ULTRASÔNICO

Fonte: http://www.mariscal8.com.br/img/v5100meio2.jpg

APRESENTAÇÃO DO EQUIPAMENTO[5]

O umidificador produz uma névoa fria e devolve a umidade do ar em ambientes de clima seco.

- Material: plástico;
- Funções: umidificador;
- Indicação: lugares de clima seco;
- Vantagens: umedece o ambiente com névoa fria;
- Tecnologia: ultra-sônico;
- É silencioso;
- Possui regulador de intensidade e indicador de nível da água e filtro desmineralizador para manter o ar puro.

FICHA TÉCNICA DO PRODUTO

- Alimentação: 110/220 volts;
- Consumo aprox. de energia: 30 watts;
- Capacidade aprox. do reservatório: 05 litros;

[5] Os dados variam de acordo com a marca do produto

- Horas aprox. de uso: 20 horas;
- Tamanho aprox. do ambiente: 20 m$^{2;}$
- Peso aprox. do produto: 3 Kg;
- Peso aprox. da embalagem: 3,2 Kg;
- Dimensões aprox. do produto: (LxAxP) – 25 x 20 x 28 cm;
- Dimensões aprox. com embalagem: (LxAxP) – 20 x 25 x 30 cm.

O INTERIOR DE UM UMIDIFICADOR

O tipo mais comum de umidificador é o umidificador evaporativo. É um modelo simples e de valor acessível.

Um reservatório capta a água fria e a joga em um recipiente; a água é absorvida por um recipiente com parede porosa. Um ventilador força o ar passar nesse recipiente que ficará úmido. Conforme o ar passa por esse recipiente, um pouco de água evapora. Quanto mais alta for à umidade relativa, mais difícil será a evaporação da água.

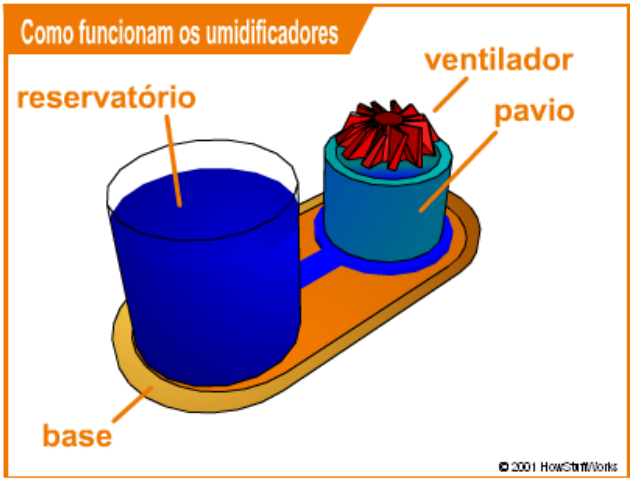


FIGURA 2.14 – O INTERIOR DE UM UMIDIFICADOR

Fonte: http://casa.hsw.uol.com.br/umidificadores

2.7 AR CONDICIONADO

O ar condicionado é um equipamento destinado a climatizar o ar em ambientes fechados, mantendo sua temperatura controlada. Esse aparelho, atualmente mais acessível, permite manter o ambiente ameno e agradável qualquer que seja a estação do ano e as condições climáticas.

No mercado há diferentes tipos de condicionadores de ar, tais como janela (ou parede), portáteis, split e sistema central.

AR CONDICIONADO DE JANELA OU PAREDE

São os mais utilizados, pois são encontrados facilmente no varejo e por possuírem preços mais acessíveis. Estes aparelhos fazem a renovação contínua do ar fresco.

FIGURA 2.15: AR CONDICIONADO DE JANELA

Fonte: www.arcondicionado.ind.br/imagens/ar_condicio

AR CONDICIONADO PORTÁTIL

São os mais práticos porque podem ser utilizados em todos os ambientes onde for necessário fazer a climatização, além de não ter nenhum custo para instalação. Funcionam expulsando o ar quente para o exterior e trazendo o ar frio para o interior, assegurando a renovação do ar.

FIGURA 2.16: AR CONDICIONADO PORTÁTIL

Fonte: www.arcondicionadoportatil.com.br/images/5A8_...

AR CONDICIONADO SPLIT

Este aparelho pode ser fixo ou móvel. Possui duas partes, uma instalada no interior e a outra do lado externo do ambiente. Além de manter o ar do ambiente agradável e de controlar a temperatura, os modelos splits reduzem o ruído da operação pois seu condensador fica do lado externo do ambiente e possuem sistema de filtragem do ar.

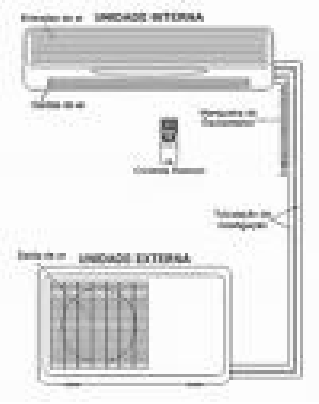


FIGURA 2.17: AR CONDICIONADO SPLIT

Fonte: www.fazfacil.com.br/images/Split.gif

SISTEMA CENTRAL DE AR CONDICIONADO

São os modelos mais recomendados para ambientes comerciais, pois climatizam muitos espaços simultaneamente. Possuem custo maior de aquisição, manutenção e operação. São silenciosos e não ficam visíveis em fachadas.

FIGURA 2.18: SISTEMA CENTRAL DE AR CONDICIONADO

Fonte: www.revistatechne.com.br/.../imagens/i96110.jpg

DIMENSIONAMENTO DO AR CONDICIONADO

De acordo com o manual de instruções do equipamento marca SPLIT[6] (2008), para o correto dimensionamento do ar condicionado é essencial considerar o tamanho do ambiente, o seu isolamento, a sua exposição ao sol (ou sombra) e o número de pessoas que freqüentarão o ambiente.

Geralmente todos os aparelhos possuem termostato que mantém, de forma automática, a temperatura selecionada. Para instalação do aparelho é importante considerar o espaço necessário especificado no manual de instruções e deve ser efetuada por técnicos especializados.

CAPACIDADE DO AR CONDICIONADO

A potência de refrigeração do equipamento é determinada pela unidade de potencia: BTU (British Thermal Unit = Unidade Térmica Britânica).

É essencial que a escolha do aparelho de ar condicionado seja feita de forma correta para garantir o conforto do usuário. Além disso, quando o equipamento possui capacidade abaixo da necessária para o ambiente, o consumo de energia aumenta demasiadamente e a vida útil do compressor torna-se menor.

[6] Disponível em http://www.fazfacil.com.br/manutencao/ar_condicionado.html

MANUTENÇÃO DO AR CONDICIONADO

De acordo com a portaria 3.523/98 do Ministério da Saúde (BRASIL, 1998) é muito importante fazer a manutenção periódica nos aparelhos de ar condicionado, por motivo de:

- Aumento do rendimento;
- Prolongamento da vida útil do equipamento;
- Evita quebras, reduzindo os gastos com trocas de peças;
- Reduz o consumo de energia;
- Proteção contra intempéries (sol, chuva, etc.);
- Manter os aparelhos limpos evita a concentração de ácaros, fungos, mofos e bactérias, mantendo o ar sempre puro.

Capítulo 3

ESPECIFICAÇÕES TÉCNICAS DO PROTÓTIPO

3.1 INTRODUÇÃO

Neste capítulo são abordadas as especificações técnicas dos componentes utilizados para construção do protótipo bem como os pontos relativos à comunicação dos módulos.

De acordo com Carvalho (2009), fornecedores e usuários de equipamentos e sistemas industriais buscam continuamente produtos com arquiteturas próprias, independentes de fabricantes, que tenham alto desempenho, comprovados mecanismos de segurança e que sejam tecnologicamente modernos. Nesta perspectiva, para a construção deste protótipo, procurou-se produtos que atendessem às exigências de controle, a distribuição e armazenamento de informações e de melhor interoperabilidade entre os módulos, fundamentais para a eficiência, flexibilidade e confiabilidade dos sistemas.

O protótipo é dividido em 03 (três) módulos fisicamente ligados entre si através do protocolo RS-485 e os seus dados são trocados através do protocolo ModBus do tipo mestre/escravo que permite que somente um dispositivo (mestre) inicie as transações, enquanto os outros dispositivos (escravos) respondem de acordo com o pedido do mestre. O primeiro módulo, chamado de CPU_Sensor, é responsável pelo coleta de dados, porém isso só é feito depois que o segundo módulo, denominado de CPU_Principal, solicita essa tarefa. Após receber os dados a CPU_Principal realiza as comparações de valores e envia um dado para o terceiro módulo, chamado de CPU_Reles, responsável apenas pelo acionamento dos aparelhos responsáveis por manter a umidade e a temperatura dentro da faixa de valores estipulados inicialmente.

3.2 COMUNICAÇÃO DOS MÓDULOS

Na comunicação de dados e na interligação em rede, protocolo é o padrão que especifica o formato dos dados e as regras que deverão ser seguidas. Uma

rede não funciona sem os protocolos uma vez que são eles que determinam como um programa deve preparar os dados que serão enviados para o estágio seguinte do processo de comunicação. Portanto, segundo Helb (1999, p. 365), o protocolo de comunicação são os conjuntos de regras e convenções e troca de informações entre sistemas, o que significa que se pode construir um dispositivo capaz de trocar informações usando aquele protocolo.

Para construção deste protótipo observou-se as possibilidades e vantagens oferecidas pelos protocolos de comunicação digitais, tais como a diminuição de afiação, facilidade de manutenção, flexibilidade na configuração da rede e, principalmente, diagnóstico dos dispositivos. Os diferentes dispositivos comunicam-se com outros e, de forma cooperativa, realizam tarefas que obedecem ao atendimento dos comandos impostos. Para tanto, optou-se pelo protocolo MODBUS do tipo mestre/escravo que permite que somente um dispositivo (o mestre) inicie as transações. Os outros dispositivos (escravos) respondem de acordo com o pedido do mestre ou de acordo com a tarefa solicitada. O dispositivo periférico processa a informação e envia os dados para o mestre.

PROTOCOLO MODBUS

MODBUS é um protocolo de comunicação de dados criado em 1970 pela Modicon Inc. (atualmente parte do grupo Schneider Electric). Segundo Borges (2007), trata-se de um bus amplamente divulgado e utilizado (variadores de velocidade, robôs, máquinas especiais, autômatos programáveis industriais, dentre outros). Este protocolo é um dos mais antigos protocolos utilizados em redes de Controladores Lógicos Programáveis – PLC – para aquisição de instrumentos e comandar atuadores.

Características Técnicas

O protocolo MODBUS utiliza como meio físico o RS-232, RS-485 ou Ethernet, com mecanismo de controle de acesso mestre/escravo. A estação mestre, normalmente um PCL, envia mensagens solicitando que os escravos enviem os

dados lidos pela instrumentação ou envia sinais a serem escritos nas saídas para o controle dos atuadores.

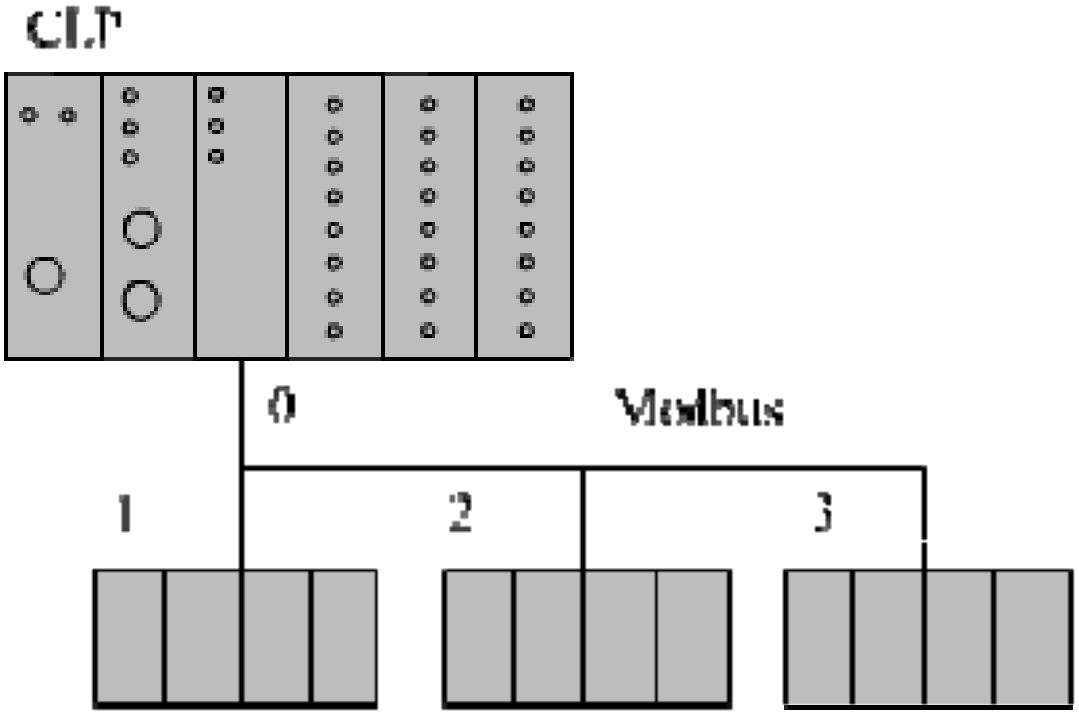


FIGURA 3.19 – MODELO DE REDE MODBUS (1 MESTRE/ 3 ESCRAVOS)

Fonte: HTTP://pt.wikipedia.org/wiki/modbus

A comunicação em protocolo MODBUS obedece a um frame que contem o endereço do escravo, o comando que será executado, uma quantidade variável de dados complementares e uma verificação de consistência de dados.

Comandos do MODBUS

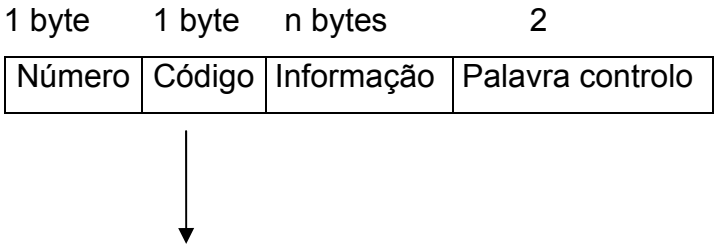
De acordo com Borges (documento técnico n° 02 da Scheneider Eletric, 2007), os principais comandos do MODBUS são:

Numero da função	Descrição
1 ou 2	Leitura de bits
2 ou 4	Leitura de palavras
5	Escrita de bits
6	Escrita de palavras
7	Leitura rápida de 8 bits
8	Diagnóstico de trocas
11	Leitura contador de eventos
12	Leitura Buffer Trace
15	Escrita múltipla de bits
16	Escrita múltipla de palavras

TABELA 2 – Principais comandos de MODBUS

Informação n bytes:

- Valor de bits ou palavras lidas;
- Valor dos bits ou palavras escritas;
- Número de palavras ou número de bits.



Identificação da trama de envio

Modos de transmissão para o padrão MODBUS

Há dois modos de transmissão para o padrão MODBUS:

- ASCII – possui caracteres codificados em 7 bits + 1 bit de paridade. O campo checksum, responsável pela verificação da integridade das palavras, é gerado pelo método LRC;
- RTU – possui caracteres codificados com 8 bits + 1 bit de paridade. O campo de checksum é gerado pelo método CRC.

Por apresentar maior densidade de caracteres que é enviada numa mesma mensagem, aumentando o desempenho de comunicação, na construção deste protótipo optou-se pelo modo de transmissão RTU (Remote Terminal Unit). Portanto, o presente trabalho abordará especificamente este modo de transmissão. No modo RTU o dispositivo configurado para cada palavra de dados é enviado apenas um caracter no padrão hexadecimal. Em relação à formação da palavra de dados que compõe o conjunto de dados (framing) da mensagem, deve-se observar a quantidade de bits por cada palavra, que será sempre igual a 11, independentemente dos parâmetros de comunicação:

- 1 start bit, 8 data bit, sem bit de paridade e 2 stop bit;
- 1 start bit, 8 data bit, 1 bit de paridade e 1 stop bit;
- 1 start bit, 8 data bit, 1 bit de paridade e 1 stop bit.

CAMPO CHECSUM

O protocolo MODBUS possui dois tipos de cálculos de checagem de integridade de dados (checksum): a checagem de paridade (pode ou não ser aplicada para cada caracter da mensagem) e a checagem do framing (tipo LRC ou CRC, para os modos ASCII e RTU, respectivamente). Para ambas, a checagem é gerada ora no mestre ora no escravo antes da transmissão.

No protocolo MODBUS é criado uma estrutura de hierarquia (um mestre e vários escravos), onde o mestre administra o conjunto das trocas de acordo com os seguintes tipos de dialogo: o mestre troca com o escravo e espera a resposta ou o mestre troca com o conjunto de escravos sem esperar a resposta. É importante salientar que dois escravos não podem dialogam simultaneamente.

A configuração de todo mestre para esperar, durante um intervalo de tempo pré-determinado antes de abortar a comunicação, é chamada de timeout. Esse timeout deve ser programado para ser longo o suficiente para dar tempo ao escravo de responder as solicitações de maneira normal. Quando o escravo detecta algum erro de transmissão, a mensagem não será validada e não será enviada nenhuma resposta ao mestre. Nesse caso, o timeout será acionado, permitindo que o mestre gerencie a ocorrência do erro – CRC 16 (modo RTU).

MENSAGEM DE QUADRO MODBUS

De acordo com Carvalho (2009), para marcar o inicio e o fim da mensagem e permitir que o dispositivo receptor determine qual dispositivo está sendo endereçado e quando a mensagem está completa, utiliza-se um quadro de mensagens. Segundo este autor, uma mensagem MODBUS é colocada no quadro e transmitida para o dispositivo, onde cada palavra da mensagem (incluindo o frame) será colocada em um dado de quadro que adiciona um start-bit, stop bit e bit de paridade. No modo RTU a palavra é de 8 bits (na verdade são 11 bits, pois é adicionado o bit de start, stop e paridade no quadro). Ressalta ainda que:

> O modo de mensagens RTU inicia com um intervalo de 3,5 caracter implementado como um caracter múltiplo de taxa de transmissão utilizada pela rede. O primeiro campo transmitido é o endereço do dispositivo; os caracteres seguintes transmitem todos os campos hexadecimais de 0 a 9 e A a F. Um dispositivo de rede monitora a rede, incluindo o intervalo de silencio e quando o primeiro campo é recebido (o endereço) após o intervalo de silencio de 3,5 caracter, o dispositivo decodifica e determina é do dispositivo. Seguindo o último caracter transmitido, um intervalo de tempo similar de 3,5 caracter finaliza o fim da mensagem e pode iniciar uma nova mensagem após o intervalo. A mensagem inteira deve ser transmitida continuamente. Se o intervalo de silencio demorar mais que 1,5 caracter e ocorrer antes de completar o quadro, o dispositivo considera a mensagem incompleta e considera o próximo byte como o endereço da nova mensagem. [...] se a mensagem iniciar 3,5 caracter antes do inicio da nova mensagem, o dispositivo receptor assume que ele está continuando com a mensagem previa, isso irá gerar uma mensagem de erro, assim como o valor final do campo CRC não será valido para combinar a mensagem. (Carvalho, 2009)

FORMATO GERAL DE UMA TRAMA TIPO RTU

START	ENDEREÇO	FUNÇÃO	DADOS	LRC	END
Silencio	2 bytes	2 bytes	N bytes	2 bytes	silencio

Para o envio, são necessárias as seguintes informações:

- Endereço bits, palavras;
- Valores bits, palavras;
- Número de bits;
- Número de palavras.

1 byte	1 byte	n bytes	2 bytes
Número Escravo	Código Função		Controlo CRC

Palavra de controlo: CRC

Quando a mensagem é recebida pelo escravo, este lê a palavra de controlo aceitando ou recusando. Entretanto, em nenhum dos casos, ele não responde.

PADRÃO DE COMUNICAÇÃO RS-485

Considerando que uma aplicação consiste de vários dispositivos em lugares diferentes e um sistema ser composto por diversas unidades, cada uma com determinada função, é necessário que exista um meio de comunicação entre eles. O padrão RS - Recommended Standard - foi criado em 1983[7] e desenvolvido pela EIA (Eletronics Industry Association), promove uma potente forma de comunicação multiponto, muito utilizada para transferência de dados para pequenas quantidades e taxas de até 10 Mbps. Embora seu alcance possa chegar até a 4000 pés, quanto maior a distancia a ser percorrida pelos dados, menor será a taxa de transmissão, conforme figura abaixo:

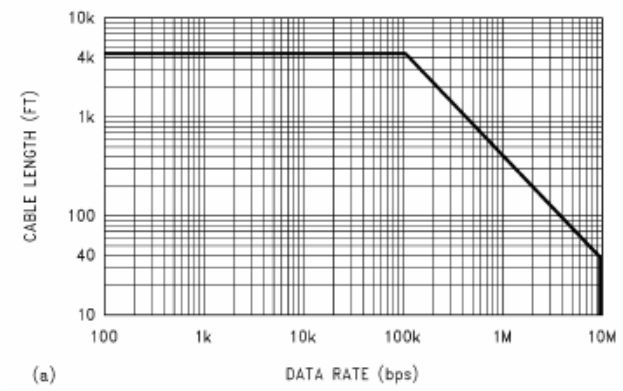


FIGURA 3.20 – DISTÂNCIA X TAXA DE TRANSMISSÃO

Fonte: http://www.cic.unb.br/~bordim/TD/Arquivos/G10_Monografia.pdf

O padrão RS-485 utiliza apenas um par de fios, aqui denominados fios A e B, e possui comunicação de forma diferencial. Quando o fio A for positivo e o B for

[7] Ten Ways to Bulletproff RS-485 Interfaces – National semiconductors – application note 1067.

negativo, obter-se-á o nível lógico 1 e quando o fio A for negativo e o B positivo, o nível lógico será 0, o que significa que o nível lógico é determinado pela diferença de tensão entre os fios, daí o nome de modo de operação diferencial[8].

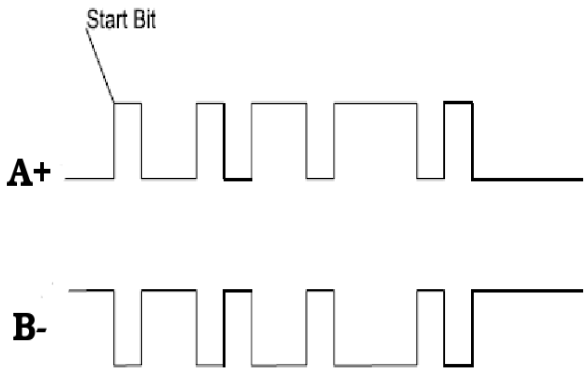


FIGURA 3.21 – PAR DIFERENCIAL

Fonte: RS 485 Professor Victor Leão - http://www.national.com/an/AN/AN-1057.pdf-

O padrão RS-485 foi desenvolvido para atender a necessidade de comunicação multiponto. Seu formato permite conectar até 32 dispositivos, sendo 1 transmissor e 1 receptor[9].

Para aplicações do padrão RS-485, utiliza-se um único PC como mestre da rede e único cabo de rede. Os terminais remotos da rede são tratados por endereçamento. Seu protocolo é do tipo half-duplex, entretanto não há definições ou recomendações para nenhum protocolo de comunicação.

O padrão RS-485 foi utilizado na construção deste protótipo por apresentar as seguintes vantagens:

- Redes locais baratas quando comparadas a outras;
- Flexibilidade de configuração;
- O usuário define, projeta e testa o seu próprio protocolo de comunicação sem a necessidade de pagar royalties aos fabricantes;
- Existe a possibilidade de usar protocolos abertos, bem definidos e testados;

[8] Transceivers and Repeaters Meeting the EIA RS-485 Interface Standard – Nacional semiconductor – application Note 409
[9] Fonte: RS 485 Professor Victor Leão - http://www.national.com/an/AN/AN-1057.pdf-

- Pode-se migrar de um padrão para outro sem perder suas características de pulso.

O padrão RS-485 possui as seguintes características elétricas:

PARÂMETRO	VALOR
Modo de operação	Diferencial
Numero de TX e RX	32 TX 32 RX
Comprimento Máximo	1200m
Taxa máxima de Comunicação	10Mbps
Tensão máxima de Modo Comum	12 à -7 volts
Tensão mínima de Transmissão (carga)	± 1,5 volts
Tensão mínima de Transmissão (sem carga)	± 6 volts
Impedância Mínima de carga	60 Ω
Limite da corrente mínima da saída em Curto circuito (mA)	150 para terra 250 para -7 até 12 volts
Impedância de Entrada do RX	12KΩ
Sensibilidade do RX	± 200mV

TABELA 3 – Características Elétricas do padrão RS-485

3.3 TRANSCEPTOR MAX 485

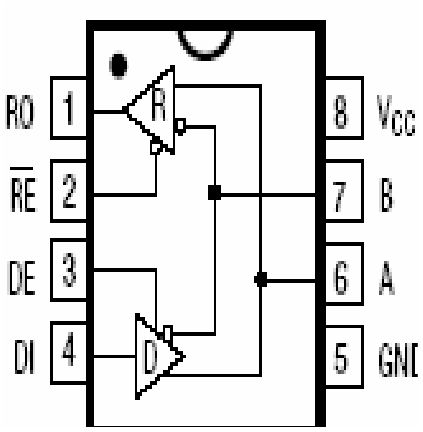


FIGURA 3.22: MAX 485

1. Ro: Saída para recepção;
2. RE: Habilitação da recepção;

3. DE: Habilitação da transmissão;
4. DI: Entrada para transmissão;
5. GND: Alimentação do circuito integrado;
6. A: Entrada não inversora;
7. B: Entrada inversora;
8. Vcc: Alimentação do circuito integrado.

MODO DE OPERAÇÃO

- Conectar RE e DE
- Ativar DE;
- Transmite por DI
- Desabilita DE.

O Max485 é um transceptor de baixa potencia para RS-485. Este utiliza um principio diferencial, que traduz um sinal lógico TTL em dois sinais, denominados de A e B. O sinal A possui a lógica do sinal TTL e o sinal B é complementar. A informação do sinal de entrada é codificada na forma do sinal A-B, sendo a diferença entre os sinais A e B. Caso esta diferença seja superior a 200mV, então, tem-se nível lógico 1. No entanto, se esta diferença for inferior a -200mV, considera-se nível lógico 0.

3.4 SENSOR SHT11

Para fazer as medições das informações de temperatura e umidade foi utilizado o sensor digital SHT11, fabricado pela indústria alemã SENSIRION. Este sensor é um circuito integrado de alta precisão, cuja saída digital é de fácil leitura e interpretação.

O SHT11 é um sensor de umidade e temperatura alojado num único chip, funciona a 3 v e possui uma saída digital calibrada. Possui como elemento sensível um polímero capacitivo para a umidade relativa e um sensor de temperatura do tipo “band gap”. Cada sensor SHT11 é individualmente calibrado numa câmara de

umidade de precisão, sendo os coeficientes programados numa memória OTP. Por possuir uma interface série de 2 fios, pode ser facilmente integrado em qualquer bus série[10].

Além disso, é muito adequado para ser utilizado em sistemas microcontrolados por possuir baixo consumo de energia e seu preço ser muito competitivo em relação a outros disponíveis no mercado. Ambos os elementos de medição estão conectados a um conversor analógico-digital de 14 bits e a um circuito de interface serial contido dentro do mesmo chip.

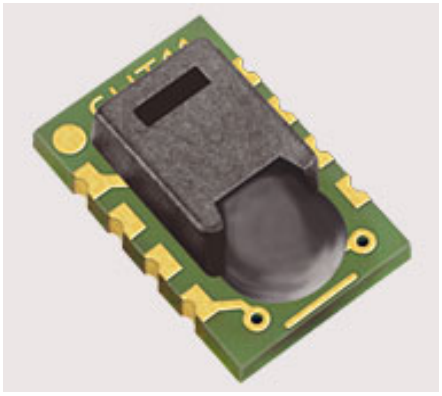


Figura 3.23: Sensor SHT11

Fonte: Datasheet SHT11 - Sensirion

Suas principais características são:

Consumo de energia	80 uW (a 12 bits, 3v, medição de 1/s)
RH intervalo operacional	0 – RH de 100%
T operam intervalo	-40 - + 125° C
Tempo de resposta	8s
Saída	Digital (interface de 2 fios)

Tabela 4: Características do Sensor SHT11

Fonte: Datasheet SHT11 - Sensirion

[10] Fonte: http://itodi.est.ips.pt/aabreu/meteobot/relatório.pdf

DIAGRAMA DE BLOCOS

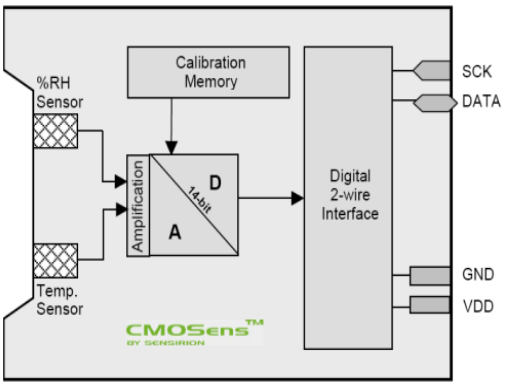


Figura 3.24: Diagrama de Blocos SHT11

Fonte: Datasheet SHT11 - Sensirion

ESPECIFICAÇÕES (INTERFACE)

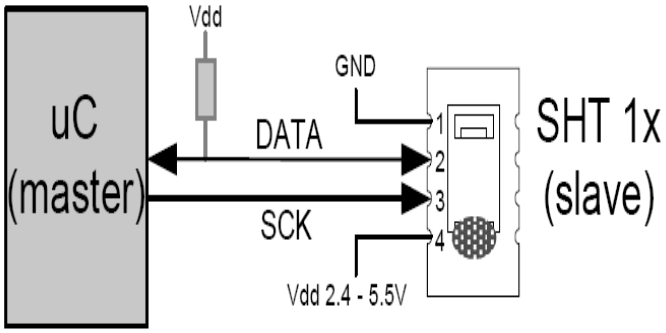


FIGURA 3.25: Pinagem do SHT11

Fonte: Datasheet SHT11 – Sensirion

PIN	Name	Comment
1	GND	Ground
2	DATA	Serial data bidirectional
3	SCK	Serial clock input
4	VDD	Supply 2.4 – 5.5V

TABELA 5 – Descrição dos Pinos SHT11

Fonte: Datasheet SHT11 - Sensirion

O sensor SHT11 se comunica com o microcontrolador através de apenas dois pinos (um para clock e outro para dados). Esses dois pinos foram conectados em duas portas GPIO do microcontrolador e o protocolo é implementado via software.

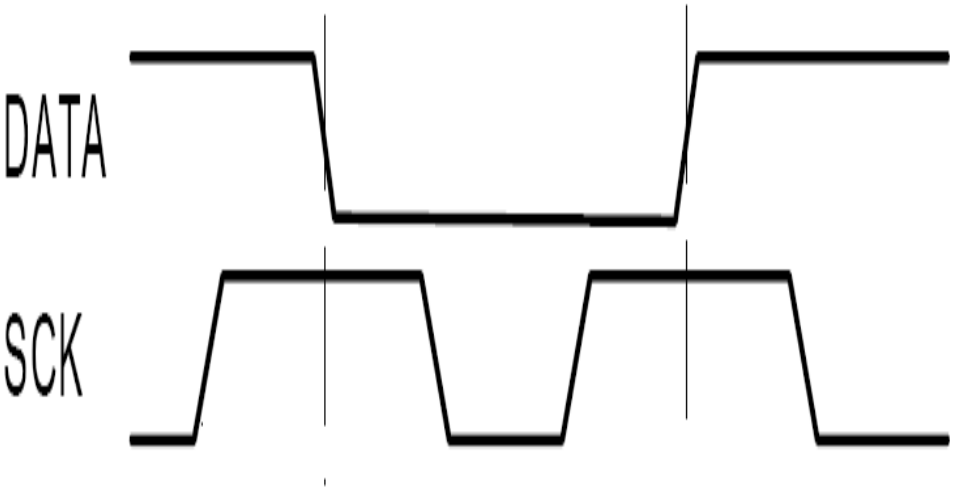


FIGURA 3.26 – Sequencia de início da comunicação.

Fonte: Datasheet SHT11 - Sensirion

CONEXÕES

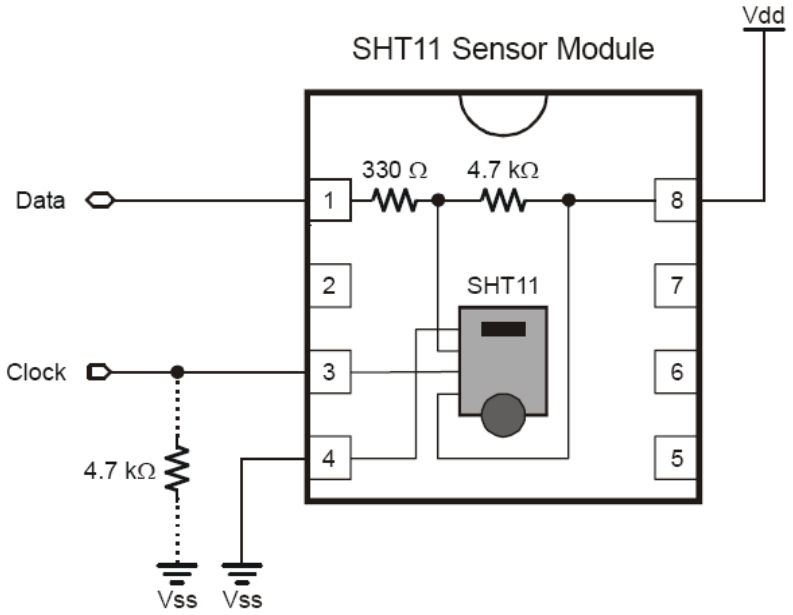


Figura 3.27: Conexões do SHT11

Fonte: Datasheet SHT11 – Sensirion

DISPLAY LCD

Os módulos LCD são interfaces de saída muito úteis em sistemas microprocessados e podem ser gráficos (encontrados com resoluções de 122x32, 128x64, 240x64 e 240x128 dots pixel, geralmente disponíveis com 20 pinos para conexão e a caracter especificados em número de linhas por colunas e encontrados nas configurações previstas, conforme tabela 6).

Para facilitar as leituras durante à noite, esses módulos podem ser encontrados com uma iluminação de fundo – LED backlight. A alimentação deste led faz-se normalmente pelos pinos 15 e 16 para os módulos comuns e 19 e 20 para os módulos gráficos, sendo os pinos 15 e 19 para ligação ao anodo e os pinos 16 e 20 para o carodo. Sua corrente de alimentação varia de 100 a 200m, variando de acordo com o modelo. Esses módulos utilizam um controlador próprio que permite sua interligação com outras placas por meio de seus pinos, onde deve ser alimentado o módulo e interligado o barramento de dados e controle do módulo com a placa do usuário. Para tanto é necessário um protocolo de comunicação entre as partes, envolvendo o envio de bytes de instruções e bytes de dados pelo sistema usuário.

Número de Colunas	Número de Linhas	Quantidade de pinos
8	2	14
12	2	14/15
16	1	14/16
16	2	14/16
16	4	14/16
20	1	14/16
20	2	14/16
20	4	14/16
24	2	14/16
24	4	14/16
40	2	16
40	4	16

Tabela 6: Módulos LDC disponíveis

Fonte: http://www2.eletronica.org/apostilas-e-ebooks/componentes/LCD_30324b.pdf

Descrição dos pinos do módulo (ou display) para conexão deste a outras placas:

Pino	Função	Descrição
1	Alimentação	Terra ou GND
2	Alimentação	VCC ou +5V
3	VO	Tensão para ajuste de contraste
4	RS Selelção	1 – Dado; 0 - Instrução
5	R/W Seleção	1 – Leitura; 0 – Escrita
6	E chip select	1 ou (1→0) – Habilita, 0 –Desabilita
7	BO LSB	Barramento de Dados
8	B1	
9	B2	
10	B3	
11	B4	
12	B5	
13	B6	
14	B7 MSB	
15	A (quando existir)	Anodo p/ LED backlight
16	K (quando existir)	Catodo p/ LED backlight

Tabela 7: Pinagem dos Módulos LCD

Fonte: http://www2.eletronica.org/apostilas-e-ebooks/componentes/LCD_30324b.pdf

Interface com CPU

De acordo com Barbacena e Fleury (1996), os módulos LCD são projetados para conectar-se com a maioria das CPU's disponíveis no mercado, desde que estas atendam as temporizações de leitura e escrita de instruções de dados, fornecido pelo fabricante do módulo. Estes tempos variam em função do clock da CPU do usuário. Geralmente, pode-se conectar o barramento de dados da CPU ao barramento do módulo, mapeando-o convenientemente na placa de usuário e efetuar-se uma operação normal de leitura e escrita sem mais problemas.

Clock da CPU	tAS (MHz)	PW EH (nS)	TH (nS)
08 MHz	325	650	75
10 MHz	250	500	50
12 MHz	200	400	33,3
16 MHz	138	275	12,5

Tabela 8: Relação clock da CPU x Temporização do Módulo LCD

Fonte: http://www2.eletronica.org/apostilas-e-ebooks/componentes/LCD_30324b.pdf

3.5 MICROCONTROLADORES ATMEGA

A família de microcontroladores AVR ATMEGA[11] da Atmel é muito utilizada tendo em vista as suas características e funcionalidades.

ATmega 8L

O microcontrolador ATmega 8L foi escolhido por apresentar as seguintes características:

- Microcontrolador de 8 bits;
- Possui 23 portas de entrada e saída (I/O);
- Memória de programa de 8 kbytes;
- Dois timers de 8 bits;
- Um timer de 16 bits;
- Ad de 10 bits;
- 130 instruções de programação;
- Porta serial;
- Manter/slave SPI serial interface;
- Consumo ativo de 3,6 mA e inativo de 1 mA;
- Programação via serial;
- Baixo custo.

[11] Dados disponíveis em HTTP://www.atmel.com/dyn/rerources/prod_documents/24865.pdf. Acesso em 24/08/09.

ATmega 32 16PC

O ATmega 32 é um microcontrolador CMOS de 8 bits com baixo consumo de energia baseado na arquitetura RISC de AVR. É capaz de executar as instruções em um único ciclo de relógio, o ATmega 32 alcança um desempenho de 1 MIPS por MHz, permitindo ao projetista otimizar consumo de energia ou velocidade de processamento. A ATmega 32 possui as seguintes características gerais:

- 32k bits de memória flash;
- 2k bytes de 1 Ram;
- 1024 bytes EEPROM;
- 2 times/contadores de 8 bits;
- 1 timer/contador de 16 bits;
- 8 canais de 10 bits de ADC, USART, WDT, POR, BOD;
- 4 canais de PWN, perto de 1SP;
- Interface serial 1P para programação dentro do sistema;
- 6 modos para conservar potencia;
- 32 pinos de I/O.

Capítulo 4

IMPLEMENTAÇÃO DO PROTÓTIPO

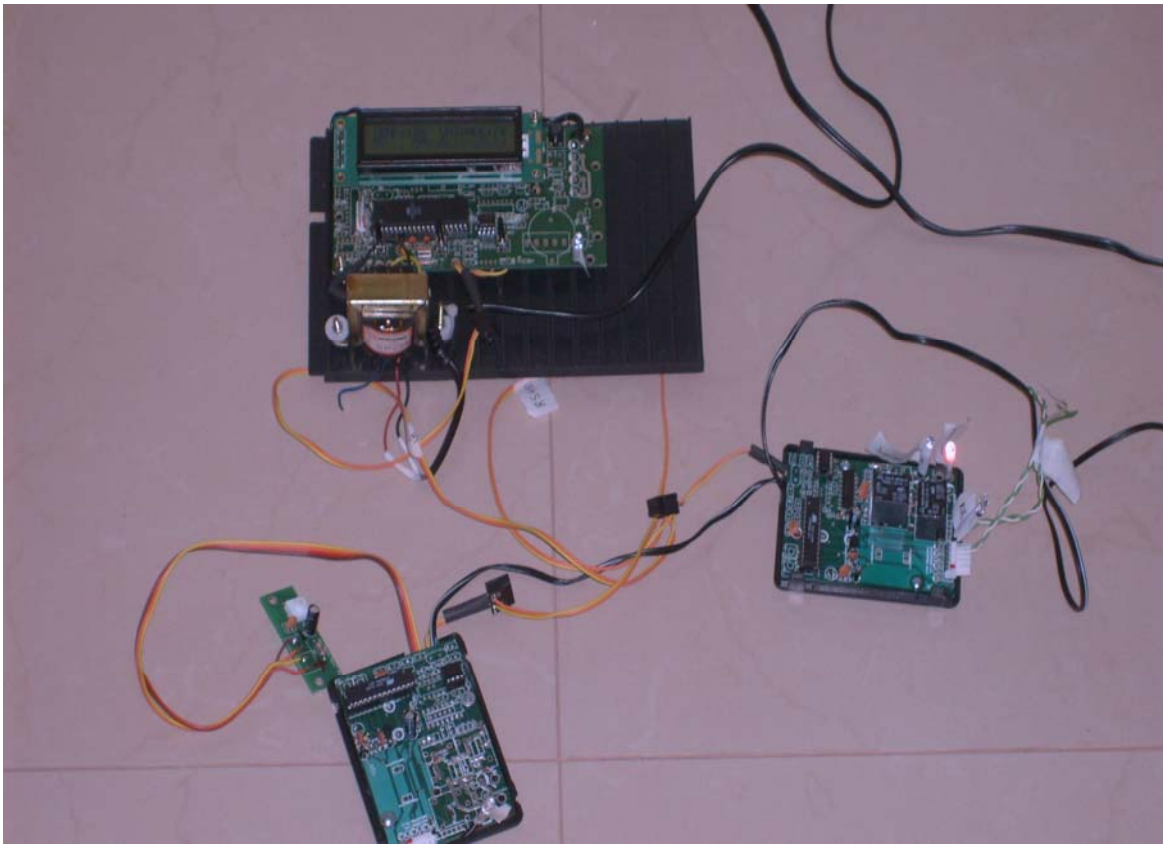


FIGURA 4. 28: PROTÓTIPO

Por desejar-se apenas demonstrar a parte conceitual do projeto, trabalhou-se neste protótipo com as faixas de 55% a 80% para a umidade relativa do ar e 18° a 23° para a temperatura, pois caso se trabalhasse com as faixas recomendadas, conforme relatado na apresentação deste, o tempo de resposta seria bastante elevado na simulação.

O protótipo foi dividido em três módulos microprocessados que se comunicam por meio do protocolo RS 485. Os módulos foram denominados neste projeto de CPU_Principal; CPU_Sensor e CPU_AcionaRele e endereçados, respectivamente, como 1, 2 e 3.

**********Trecho Do Código de Programação**********

```
eeprom char CPU_Principal=1;
eeprom char CPU_Sensor=2;
eeprom char CPU_AcionaRele=3;

/*
******** PROTOCOLO EXCLUSIVO DE COMUNICAÇÃO DA REDE *******
Byte0 = Endereço do CLP de destino
Byte1 = Função (comando): 1=Leitura (solicitação de informação) ou SaídaRele1, 2=Escrita(comando remoto) ou SaídaRele2
Byte2 = Informação (dados): 0=Temperatura ou Desligar relé, 1=Umidade ou Ligar relé, 2=Status (status do relé), 254=CRC Inválido, 255=Comando inválido
Byte3 = Primeiro (High) Byte do digito verificador CRC16 (Cyclic Redundancy Check)
Byte4 = Segundo (Low) Byte do digito verificador CRC16 (Cyclic Redundancy Check)
*/

//##########  VAIAVEIS PARA COMUNICAÇÃO DA REDE  ##########
char tx_buffer[255]; //USADO COMO BUFFER PARA ENVIAR PARA REDE
char FlagBufferNovo=0; //Usado para identificar a chegada de novo Buffer

//***** VARIÁVEIS EM GERAL
char text[20];
char SequenciaComando=1;
char Temperatura;
char Umidade;

//***** FUNÇÕES EM GERAL
void Reiniciar(void);
void crc16(unsigned char *uchMsg, unsigned short usDataLen); //Função para cálculo de CRC
void comunica_rede(unsigned char CLP, unsigned char Funcao, unsigned char Informacao);
void ZeraBuffer(void);
```

**********Fim Do Trecho Do Código de Programação**********

O sistema começa operar a partir da CPU_Principal que, inicialmente, carrega as funções gerais do equipamento (verificação de erros, estabelecimento da conexão, entre outras). Em seguida, essa CPU carrega as funções responsáveis pela leitura da umidade relativa do ar e da temperatura, fazendo a conexão com a CPU_Sensor por meio da porta TX.

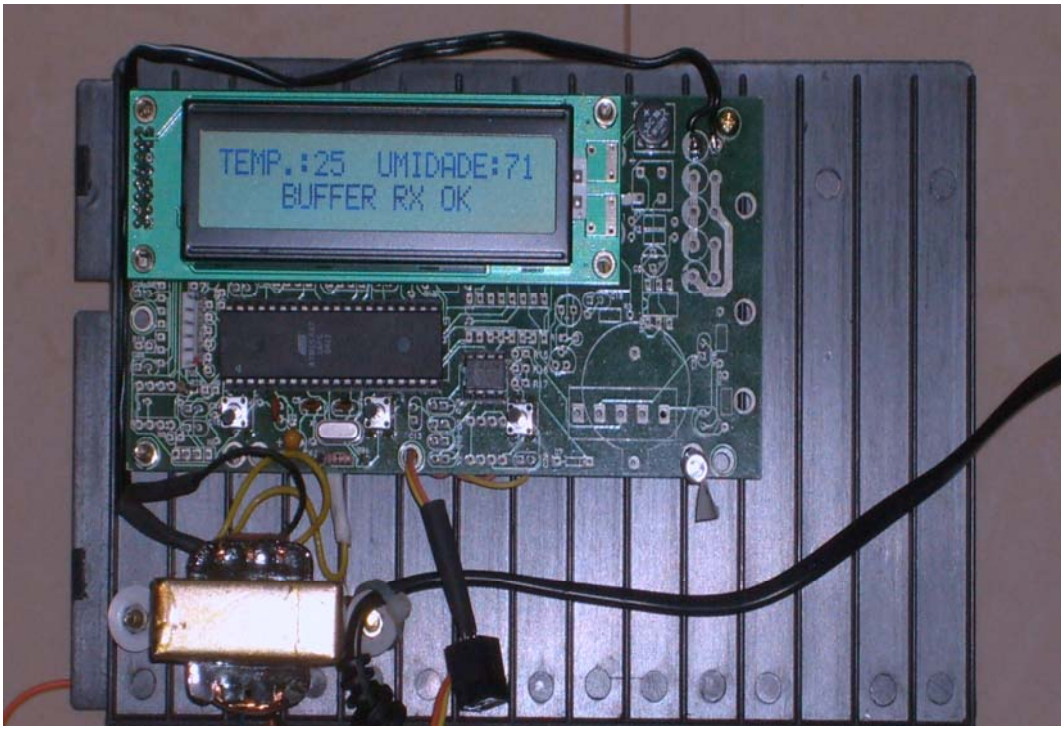


FIGURA 4.29: CPU_Principal

**********Trecho Do Código de Programação**********

```
if (SequenciaComando==1) //Leitura de temperatura
        {
                TX_RX = 1;
                delay_ms(10);
                comunica_rede(CPU_Sensor, 1, 0); //Leitura Temperatura
                delay_ms(10);
                TX_RX = 0;
                lcd_gotoxy(0,1);
                lcd_putsf("TX...    CPU_Sensor");

        }
        if (SequenciaComando==2) //Leitura de umidade
        {
                TX_RX = 1;
                delay_ms(10);
                comunica_rede(CPU_Sensor, 1, 1); //Leitura Umidade
                delay_ms(10);
                TX_RX = 0;
                lcd_gotoxy(0,1);
                lcd_putsf("TX...    CPU_Sensor");
        }
```

**********Fim Do Trecho Do Código de Programação**********

Após o estabelecimento da conexão, a CPU_Sensor verifica as mensagens recebidas da CPU_Principal e se conecta diretamente com o Sensor SHT11 que, através dos comandos fornecidos pelo fabricante no Datasheet do sensor, realiza as medições solicitadas e envia novamente os valores para a CPU_Sensor que responde as solicitações da CPU_Principal, enviando os dados aferidos. Esse procedimento está em consonância à estrutura de hierarquia criada pelo protocolo MODBUS onde o mestre administra o conjunto de trocas com o tipo de dialogo: o mestre troca com o escravo e a espera a resposta.

A CPU_Principal, de posse dos dados coletados e permitindo a visualização dos valores aferidos através de um display de LCD, realiza as comparações entre os valores medidos e os valores estipulados armazenados na memória.

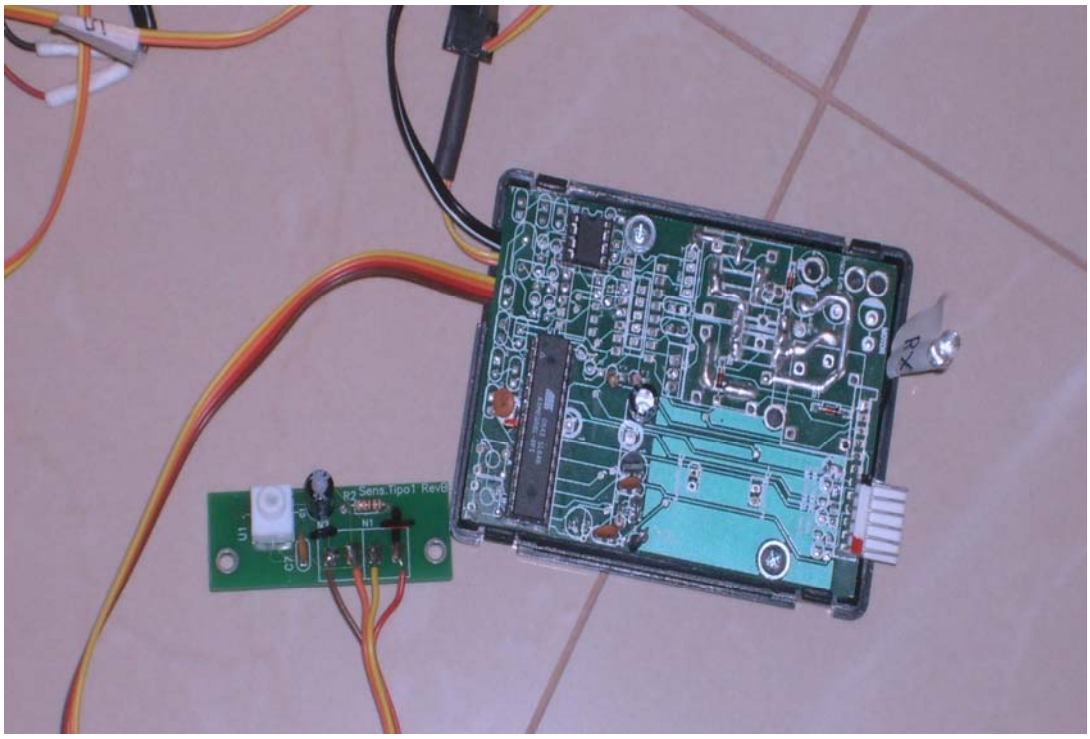


FIGURA 4.30: CPU_Sensor

**********Trecho Do Código de Programação**********

```
while(1)
{
        status = 0b00000000; //Leitura normal
        le_sensirion();
        crc16(&rx_buffer[0],3);//efetua a cálculo de CRC
        if ((rx_buffer[3] == auchCRCLo) & (rx_buffer[4] == auchCRCHi) & (FlagBufferNovo==1))
```

```
{
        FlagBufferNovo=0;
        rx_buffer_overflow=0;
        rx_wr_index=0;
        rx_rd_index=0;
        rx_counter=0;
        if (rx_buffer[0]==CPU_Sensor)
        {
                LedIndicador=1;

                if (rx_buffer[1]==1 & rx_buffer[2]==0) //Leitura Temperatura
                {
                        TX_RX = 1;
                        delay_ms(10);
                        comunica_rede(CPU_Principal,1,(char)temp_val.f);
                                                        //CLP,Funcao,Informacao
                        delay_ms(10);
                        TX_RX = 0;
                }

                if (rx_buffer[1]==1 & rx_buffer[2]==1) //Leitura Umidade
                {
                        TX_RX = 1;
                        delay_ms(10);
                comunica_rede(CPU_Principal,1,(char)humi_val.f);
                                                        //CLP,Funcao,Informacao
                        delay_ms(10);
                        TX_RX = 0;
                }
        }
        ZeraBuffer();
```

**********Fim Do Trecho Do Código de Programação**********

Quando os dados coletados relativos à umidade relativa do ar estiverem abaixo de 55%, a CPU_Principal enviará uma mensagem para a CPU_AcionaRele e indicará que o dispositivo de controle da umidade deve ser ligado. Ao receber esse status a CPU_AcionaRele acionará o dispositivo através de um relé. Quando os dados da umidade relativa do ar atingir 80%, a CPU Principal informará para a CPU_AcionaRele que esta deverá desligar o dispositivo de controle.

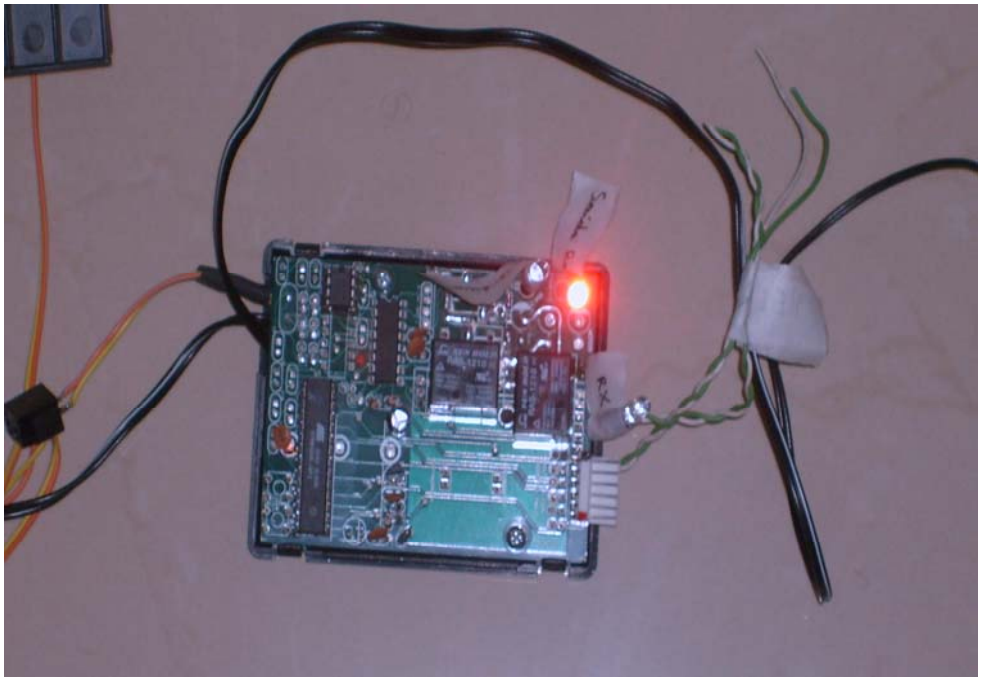


FIGURA 4.31: CPU_AcionaRele

**********Trecho Do Código de Programação**********

```
if (SequenciaComando==3) //Liga/Desliga SaidaRele1 dependendo da umidade
{
        if (Umidade <75) //Liga SaidaRele1
        {
                TX_RX = 1;
                delay_ms(10);
                comunica_rede(CPU_AcionaRele, 1, 1); //Liga SaidaRele1
                delay_ms(10);
                TX_RX = 0;
        }
        else
        {
                if (Umidade >= 90) //Desliga SaidaRele1
                {
                        TX_RX = 1;
                        delay_ms(10);
                         comunica_rede(CPU_AcionaRele, 1, 0); //Desliga SaidaRele1
                        delay_ms(10);
                        TX_RX = 0;
                        }
                        else
                        {
        if (SequenciaComando<=3) //Incrementa a sequencia de comando
        {
                SequenciaComando++;
                }
        else
        {
                SequenciaComando=1;
```

```
}
```

**********Fim Do Trecho Do Código de Programação**********

Para a temperatura, o processo é realizado da mesma maneira. Quando os dados coletados indicam na CPU_Principal que a temperatura ambiente é maior que 23°, esta envia os sinais para a CPU_AcionaRele que o dispositivo de controle da temperatura deverá ser ativado e que este deverá ser desligado quando os dados constantes na CPU_Principal alcançarem a temperatura de 18°.

**********Trecho Do Código de Programação**********

```
if (SequenciaComando==4) //Liga/Desliga SaidaRele2
{
	if (Temperatura > 23) //Liga SaidaRele2
	{
		TX_RX = 1;
		delay_ms(10);
		comunica_rede(CPU_AcionaRele, 2, 1); //Liga SaidaRele2
		delay_ms(10);
		TX_RX = 0;
	}
	else
	{
		if (Temperatura <= 18) //Desliga SaidaRele2
		{
			TX_RX = 1;
			delay_ms(10);
			comunica_rede(CPU_AcionaRele, 2, 0); //Desliga SaidaRele2
			delay_ms(10);
			TX_RX = 0;
			}
			else
			{
	if (SequenciaComando<=3) //Incrementa a sequencia de comando
	{
		SequenciaComando++;
	}
	else
	{
		SequenciaComando=1;
	}
```

**********Fim Do Trecho Do Código de Programação**********

A CPU_Principal, sempre que se comunicar com a CPU_AcionaRele, aguardará uma resposta confirmando se o comando foi aceito com sucesso.

**********Trecho Do Código de Programação**********

```
if (rx_buffer[1]==1 & rx_buffer[2]==0) //Desliga SaidaRele1
{
        TX_RX = 1;
        delay_ms(10);
        comunica_rede(CPU_Principal, 2, 0); //Retorna o mesmo comando para CPU_Principal entender que o comando foi aceito
        delay_ms(10);
        TX_RX = 0;
        SaidaRele1=0;
}

if (rx_buffer[1]==1 & rx_buffer[2]==1) //Liga SaidaRele1
{
        TX_RX = 1;
        delay_ms(10);
        comunica_rede(CPU_Principal, 2, 1); //Retorna o mesmo comando para CPU_Principal entender que o comando foi aceito
        delay_ms(10);
        TX_RX = 0;
        SaidaRele1=1;
        }

if (rx_buffer[1]==2 & rx_buffer[2]==0) //Desliga SaidaRele2
{
        TX_RX = 1;
        delay_ms(10);
        comunica_rede(CPU_Principal, 2, 0); //Retorna o mesmo comando para CPU_Principal entender que o comando foi aceito
        delay_ms(10);
        TX_RX = 0;
        SaidaRele2=0;
}

if (rx_buffer[1]==2 & rx_buffer[2]==1) //Liga SaidaRele2
{
        TX_RX = 1;
        delay_ms(10);
        comunica_rede(CPU_Principal, 2, 1); //Retorna o mesmo comando para CPU_Principal entender que o comando foi aceito
        delay_ms(10);
        TX_RX = 0;
        SaidaRele2=1;
}
```

**********Fim Do Trecho Do Código de Programação**********

O sistema ficará constantemente em execução até ser interrompido pelo usuário.

TESTES E VALIDAÇÃO DOS RESULTADOS

Para fazer os testes foi realizada a monitoração de ambientes utilizando-se o protótipo e o Termohigrômetro TH02 da marca “Impac”.

Os dois monitoradores foram submetidos às mesmas situações e horários, conforme registros abaixo:

Teste 1:

	Protótipo	**Termohigrômetro**
Ambiente	quarto	quarto
Horário	15:30 h	15:30 h
Temperatura	25°C	24,6°C
Umidade	69%	72%

Tabela 9: Dados Coletados no primeiro monitoramento

FIGURA 4.32 Valores Medidos em Temperatura Ambiente

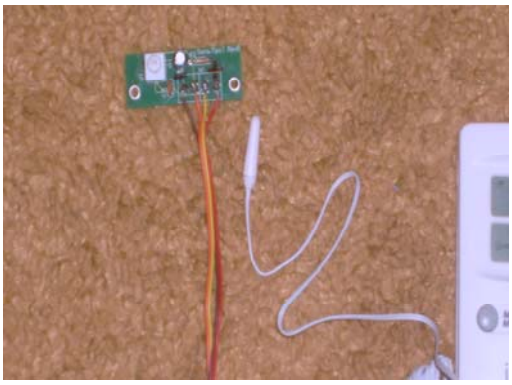


FIGURA 4.33 Sensores do protótipo e do termohigrômetro

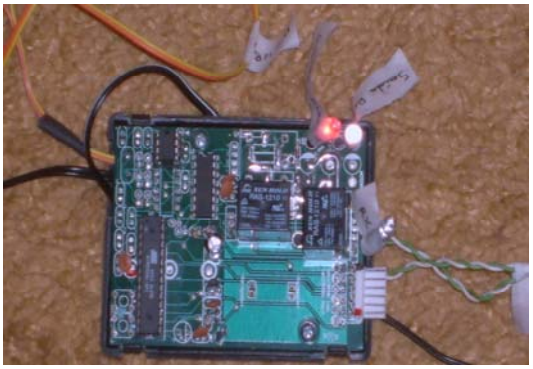


FIGURA 4.34: Funcionamento da CPU_Rele

Observa-se que, em situações iguais, houve uma pequena diferença entre os dados obtidos pelo protótipo e termohigrômetro.

Entretanto, ressalta-se o funcionamento do protótipo que foi programado para ligar ou desligar os aparelhos de ar condicionado e umidificador de ar sempre que os valores estivessem fora da faixa previamente estipulada, ou seja, ligar quando a temperatura estiver acima de 23°C e desligar quando esta estiver a 18° C; ligar quando a umidade estiver abaixo de 55% e desligar quando esta atingir 80% da umidade relativa do ar. Ao fazer a leitura da umidade relativa do ar e da temperatura através do Sensor SHT11 e mostrar os valores lidos na tela do display, constata-se que o indicador luminoso que representa o aparelho de ar condicionado encontra-se aceso (a temperatura estava a 25°C, portanto acima do valor estipulado) e que o indicador luminoso que representa o aparelho umidificador de ar encontra-se apagado (a umidade estava a 69%, portanto dentro do valor estipulado).

Teste 2:

	Protótipo	**Termohigrômetro**
Ambiente	Caixa de isopor com gelo	Caixa de isopor com gelo
Horário	15:45 h	15:45 h
Temperatura	17°C	18°C
Umidade	65%	71%

Tabela 10: Dados Coletados no segundo monitoramento

FIGURA 4.35: Valores medidos em ambiente resfriado

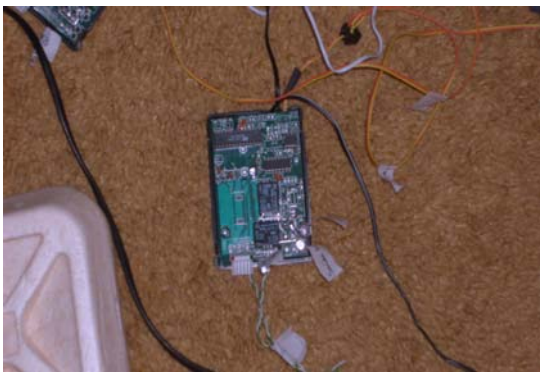


FIGURA 4.36: Funcionamento da CPU_Rele

A diferença nos dados obtidos pelo protótipo e termohigrômetro permaneceu. Neste monitoramento, tanto a temperatura quanto a umidade estão dentro da faixa estipulada, podendo-se constatar que o indicador luminoso que representa o aparelho de ar condicionado desligou-se automaticamente quando a temperatura ficou abaixo de 18°C e que o indicador luminoso que representa o aparelho umidificador de ar ainda permaneceu desligado uma vez que a umidade relativa encontrava-se em 71%.

Teste 3:

	Protótipo
Ambiente	Ambiente desumidificado com a utilização de um secador de cabelo
Horário	16:00 h
Temperatura	37°C
Umidade	43%

Tabela 11: Dados Coletados pelo protótipo, terceiro monitoramento

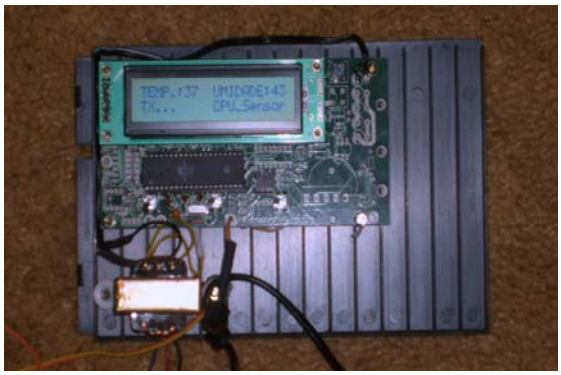


FIGURA 4.37: Valores medidos, ambiente desumidificado

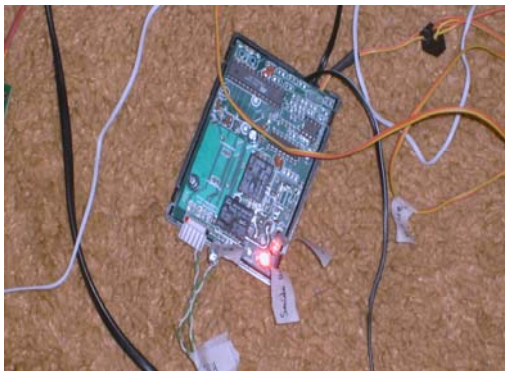


Figura 4.38: Funcionamento da CPU_Rele

Nesta situação, tanto a temperatura quanto a umidade do ar estão fora da faixa estipulada, ou seja, a temperatura está a 37°C (superior a 23°C) e a umidade do ar encontra-se em 43% (abaixo de 55%), podendo-se constatar que os indicadores luminosos que representam tanto o aparelho de ar condicionado quanto o aparelho umidificador de ar foram acionados automaticamente.

Capítulo 5

CONCLUSÃO

O sistema de monitoramento e controle de temperatura e umidade de ambientes desenvolvido neste trabalho mostrou-se eficaz devido aos resultados obtidos na realização dos testes de seu funcionamento, apresentando um comportamento de acordo com o proposto, ratificando a confiabilidade.

Os produtos utilizados na construção do protótipo atenderam as exigências de controle, distribuição, armazenamento das informações e de melhor interoperabilidade entre os módulos. Além disso, ainda há o fator econômico, pois os componentes utilizados foram de custo relativamente baixo e de fácil disponibilidade no mercado brasileiro.

Os dispositivos comunicaram-se, por meio do padrão RS-485, de forma integrada e realizaram as tarefas obedecendo ao atendimento dos comandos impostos através do protocolo Modbus do tipo mestre/escravo. As medições das informações de temperatura e umidade, realizadas pelo Sensor Digital SHT11 foram efetuadas e possibilitaram uma fácil leitura e interpretação dos dados.

Os testes foram realizados em ambiente submetido a variações de temperaturas e umidades que permitiram melhor visualização do sistema. Quando os valores estipulados foram alcançados, os relés ligaram ou desligaram os aparelhos de ar condicionado e umidificador de ar, possibilitando manter a temperatura e a umidade relativa do ar em ambientes internos dentro dos níveis recomendados para saúde e bem estar do homem.

A conclusão deste projeto é realizada de forma provisória, uma vez que, embora eficaz em seu propósito inicial, ainda há outras possibilidades de testes, sugeridas para projetos futuros, tais como: a realização da comunicação dos módulos utilizando tecnologias de redes sem fio, como, por exemplo, rádio freqüência e Bluetooth, bem como o controle e monitoramento de medidas relacionadas a outros segmentos (pressão, nível de combustível, dentre outros).

REFERÊNCIAS BIBLIOGRAFICAS ‘ .

AYOADE, J.O. **Introdução à climatologia para os trópicos.** Tradução de Maria Juraci Zani dos Santos; revisão de Suely Bastos. 5ª ed – Rio de Janeiro: Bertrand Brasil, 1998.

ATMEGA, **Microcontrolador Atmega 8L**, ATMEL, 2004. Disponível em: <http://www.atmel.com/dyn/resources/prod_documents/24865.pdf Acesso em 16 de setembro de 2009.

BARBACENA, Ilton L e FLEURY, Claudio Afonso. **Display LCD.** Disponível em <www2.eletronica.org/apostilas-e-books/componentes/LCD_30324b.pdf> Acesso em 25 de setembro de 2009.

BORGES, Fátima. **Redes de Comunicação Industrial.** Documento técnico n° 2. Schneider Eletric. 2007.

BRASIL, Ministério da Saúde. Programa de Atendimento ao Paciente Asmático. Disponível em: http://www.saude.df.gov.br/ Acesso em 13 set 2008.

_____. **Portaria 3.523/98**. Porque fazer manutenção em seu aparelho? Disponível em: <http://www.aguiarcondicionado.com.br/index.php?pg=dicas> Acesso em 12 de jun de 2009.

CARVALHO, Francisco Manoel. **Introdução a redes de comunicação: protocolos de transmissão.** Disponível em <www.artigonal.com/tecnologia-artigos/introdução-a-redes-de-comunicação-protocolos-de-transmissão> Acesso em 05 de agosto de 2009.

FOUCAULT, Alain. **O CLIMA:** História e devir do meio ambiente. Coleção Perspectiva Ecológica. São Paulo: Instituto Piaget, 1993.

GIODA, A & AQUINO NETO, F.R. **Considerações sobre estudos de ambientes industriais e não industriais no Brasil:** uma abordagem comparativa. Cadernos Saúde Pública, Rio de Janeiro, v. 19, n. 5, set/out – 2003. Disponível em: <http://www.scielo.br/scielo.php?pid=S0102-311X2003000500017&script=sci_arttext> Acesso em 18 de agosto de 2009.

HELB, Gilbert. **Comunicação de Dados.** Tradução da 6ª edição original de Vandenberg de Sousa. Campus, Rio de Janeiro, 1999.

INDRIUMAS, Alexandre. **Como funciona a umidade do ar.** Disponível em <http://ciencia.hsw.uol.com.br/umidade-do-ar5htm> Acesso em 12 de set 2008.

LAROUSSE, Àtica: Dicionário da Língua Portuguesa – Paris: Larousse / São Paulo: Ática, 2001; p. 1005.

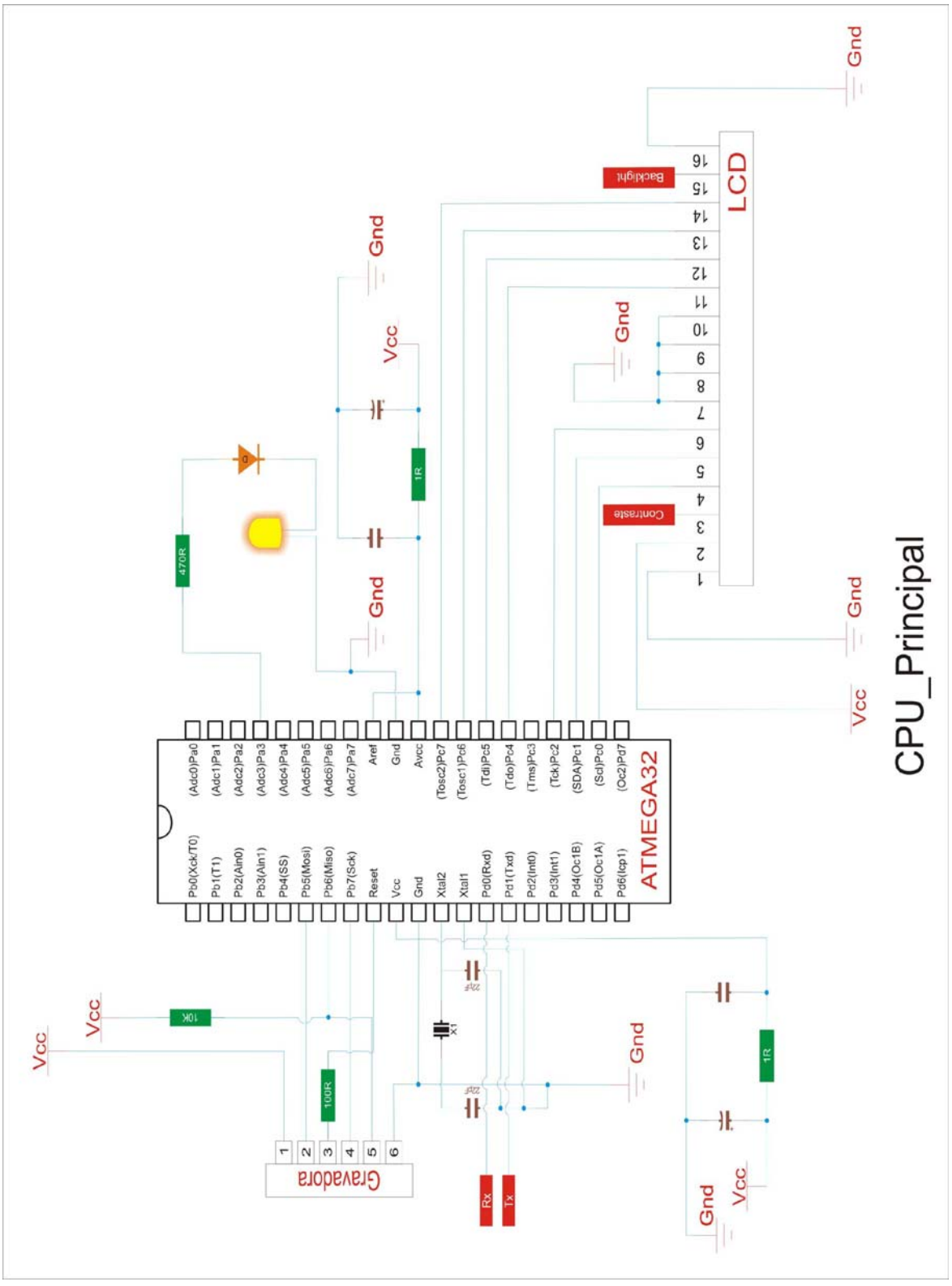
LEÃO, Victor. **RS-485 – Especificação e Utilização.** UFB, Laboratório de Propriedade Ópticas. Disponível em <http://www.national.com/an/AN/AN-1057.pdf> Acesso em 21/02/2009.

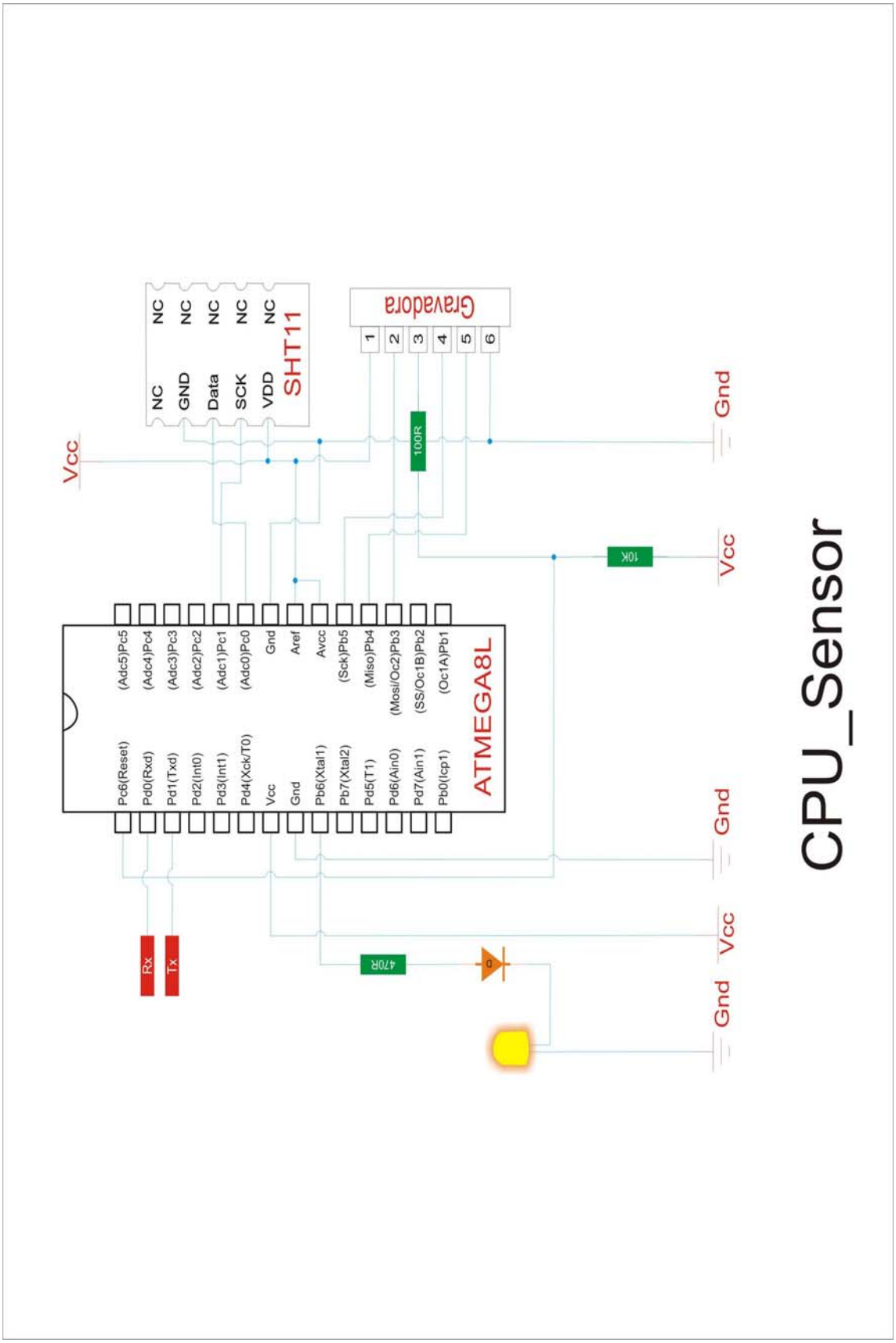
STELMACH, Rafael. **Mau uso do ar condicionado compromete a saúde.** Jornal da Orla. Data: 27 de janeiro de 2008. Disponível em: < http://www.sicon.org.br/saude_no_condominio_integra.asp?codigo=5> Acesso em 15 de jun de 2009.

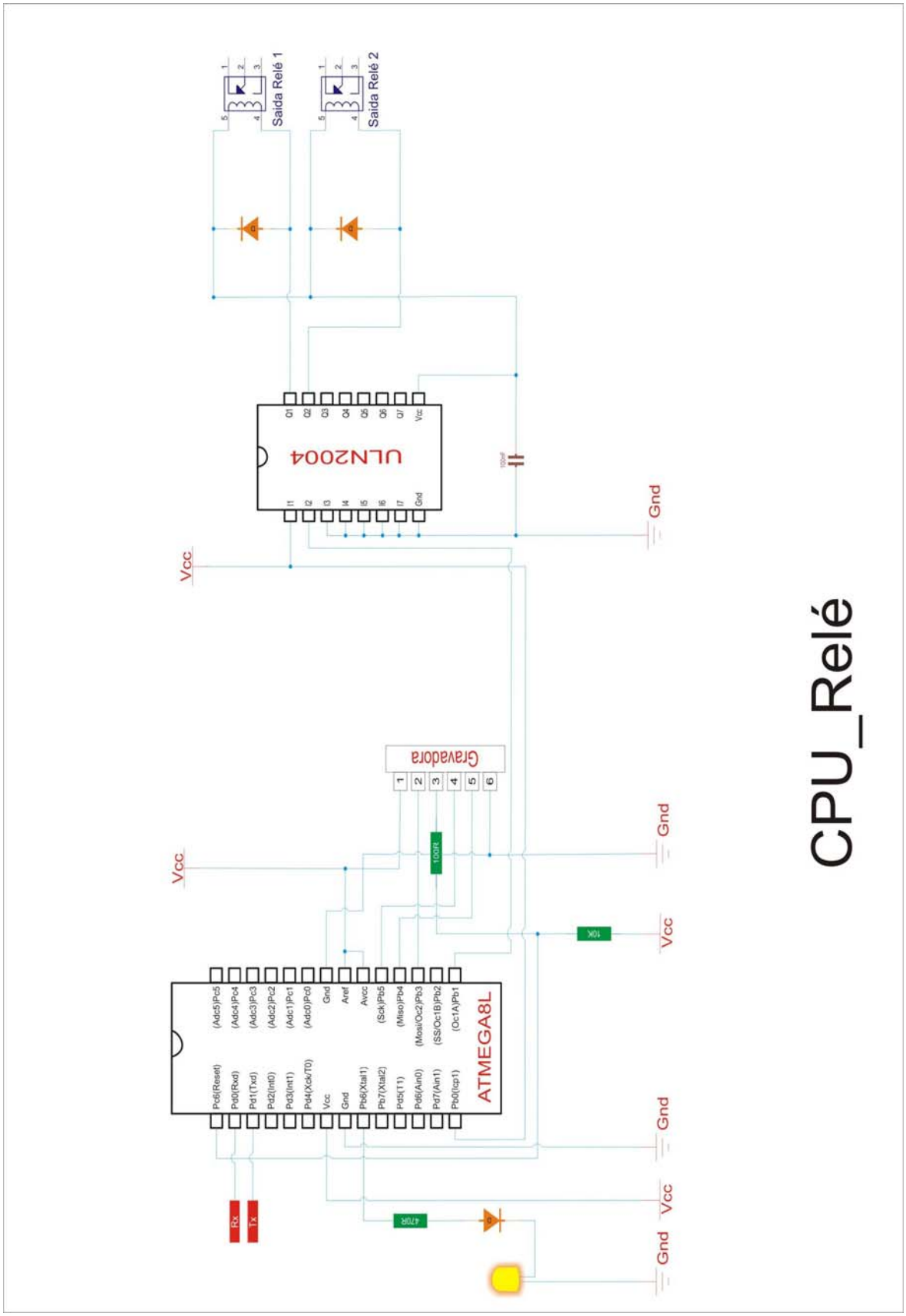
APENDICE

Apêndice A

ESQUEMÁTICOS




CPU_Sensor
ATMEGA8L
Pc6(Reset)
Pd0(Rxd)
Pd1(Txd)
Pd2(Int0)
Pd3(Int1)
Pd4(Xck/T0)
Vcc
Gnd
Pb6(Xtal1)
Pb7(Xtal2)
Pd5(T1)
Pd6(Ain0)
Pd7(Ain1)
Pb0(Icp1)
(Adc5)Pc5
(Adc4)Pc4
(Adc3)Pc3
(Adc2)Pc2
(Adc1)Pc1
(Adc0)Pc0
Gnd
Aref
Avcc
(Sck)Pb5
(Miso)Pb4
(Mosi/Oc2)Pb3
(SS/Oc1B)Pb2
(Oc1A)Pb1
SHT11
NC
NC
NC
NC
NC
NC
GND
Data
SCK
VDD
Gravadora
1
2
3
4
5
6
100R
10K
470R
Rx
Tx
Vcc
Gnd
Vcc
Gnd
Vcc
Gnd


CPU_Relé
Saida Relé 1
Saida Relé 2
ULN2004
Gravadora
ATMEGA8L
Vcc
Gnd
Rx
Tx
100R
10K
470R
10nF

Apêndice B

Programa linguagem C – CPU_Principal

```
/*
******** PROTOCOLO EXCLUSIVO DE COMUNICAÇÃO DA REDE *******
Byte0 = Endereço do CLP de destino
Byte1 = Função (comando): 1=Leitura(solicitação de informação) ou SaídaRele1, 2=Escrita(comando
remoto) ou SaídaRele2
Byte2 = Informação (dados): 0=Temperatura ou Desligar relé, 1=Umidade ou Ligar relé,
2=Status(status do relé), 254=CRC Inválido, 255=Comando inválido
Byte3 = Primeiro (High) Byte do digito verificador CRC16 (Cyclic Redundancy Check)
Byte4 = Segundo (Low) Byte do digito verificador CRC16 (Cyclic Redundancy Check)

********* ENDEREÇO DOS CLP´S *********
1 = CPU_Principal
2 = CPU_Sensor
3 = CPU_AcionaMotor
*/


eeprom char CPU_Principal=1;
eeprom char CPU_Sensor=2;
eeprom char CPU_AcionaRele=3;


//#include <stdlib.h>
#include <mega32.h>

#define             TX_RX PORTD.3

// Alphanumeric LCD Module functions
#asm
   .equ __lcd_port=0x15
#endasm

#include <lcd.h>


//#########   VAIAVEIS PARA COMUNICAÇÃO DA REDE   ##########
char tx_buffer[255];//USADO COMO BUFFER PARA ENVIAR PARA REDE
char FlagBufferNovo=0;//Usado para identificar a chegada de novo Buffer



//######## USART INICIO #######
#define RXB8 1
#define TXB8 0
#define UPE 2
#define OVR 3
#define FE 4
#define UDRE 5
#define RXC 7

#define FRAMING_ERROR (1<<FE)
#define PARITY_ERROR (1<<UPE)
#define DATA_OVERRUN (1<<OVR)
```

```
#define DATA_REGISTER_EMPTY (1<<UDRE)
#define RX_COMPLETE (1<<RXC)

// USART Receiver buffer
#define RX_BUFFER_SIZE 8//tamanho do buffer na rede
char rx_buffer[RX_BUFFER_SIZE];
unsigned char rx_wr_index,rx_rd_index,rx_counter;
// This flag is set on USART Receiver buffer overflow
bit rx_buffer_overflow;

// USART Receiver interrupt service routine
#pragma savereg-
interrupt [USART_RXC] void usart_rx_isr(void)
{
char status,data;
#asm
    push r26
    push r27
    push r30
    push r31
    in   r26,sreg
    push r26
#endasm
status=UCSRA;
data=UDR;
FlagBufferNovo=1;
if ((status & (FRAMING_ERROR | PARITY_ERROR | DATA_OVERRUN))==0)
   {
   rx_buffer[rx_wr_index]=data;
   if (++rx_wr_index == RX_BUFFER_SIZE) rx_wr_index=0;
   if (++rx_counter == RX_BUFFER_SIZE)
      {
      rx_counter=0;
      rx_buffer_overflow=1;
      };
   };
#asm
    pop  r26
    out  sreg,r26
    pop  r31
    pop  r30
    pop  r27
    pop  r26
#endasm
}
#pragma savereg+

#ifndef _DEBUG_TERMINAL_IO_
// Get a character from the USART Receiver buffer
#define _ALTERNATE_GETCHAR_
#pragma used+
char getchar(void)
{
char data;
while (rx_counter==0);
data=rx_buffer[rx_rd_index];
if (++rx_rd_index == RX_BUFFER_SIZE) rx_rd_index=0;
#asm("cli")
--rx_counter;
#asm("sei")
```

```
return data;
}
#pragma used-
#endif

//######## USART FINAL #######



#include <stdio.h>
#include <delay.h>
#include <math.h>



#define ADC_VREF_TYPE 0x40


//***** VARIÁVEIS EM GERAL
char text[20];
char SequenciaComando=1;
char Temperatura;
char Umidade;


//***** FUNÇÕES EM GERAL
void Reiniciar(void);
void crc16(unsigned char *uchMsg, unsigned short usDataLen);//Função para cálculo de CRC
void comunica_rede(unsigned char CLP, unsigned char Funcao, unsigned char Informacao);
void ZeraBuffer(void);



/* Table of CRC values for high-order byte */
const static unsigned char uchCRCHi[] = {
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1,
0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1,
0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40,
0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1,
0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40,
0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40,
0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1,
```

```
0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40
} ;


/* Table of CRC values for low-order byte */
const static char uchCRCLo[] = {
0x00, 0xC0, 0xC1, 0x01, 0xC3, 0x03, 0x02, 0xC2, 0xC6, 0x06,
0x07, 0xC7, 0x05, 0xC5, 0xC4, 0x04, 0xCC, 0x0C, 0x0D, 0xCD,
0x0F, 0xCF, 0xCE, 0x0E, 0x0A, 0xCA, 0xCB, 0x0B, 0xC9, 0x09,
0x08, 0xC8, 0xD8, 0x18, 0x19, 0xD9, 0x1B, 0xDB, 0xDA, 0x1A,
0x1E, 0xDE, 0xDF, 0x1F, 0xDD, 0x1D, 0x1C, 0xDC, 0x14, 0xD4,
0xD5, 0x15, 0xD7, 0x17, 0x16, 0xD6, 0xD2, 0x12, 0x13, 0xD3,
0x11, 0xD1, 0xD0, 0x10, 0xF0, 0x30, 0x31, 0xF1, 0x33, 0xF3,
0xF2, 0x32, 0x36, 0xF6, 0xF7, 0x37, 0xF5, 0x35, 0x34, 0xF4,
0x3C, 0xFC, 0xFD, 0x3D, 0xFF, 0x3F, 0x3E, 0xFE, 0xFA, 0x3A,
0x3B, 0xFB, 0x39, 0xF9, 0xF8, 0x38, 0x28, 0xE8, 0xE9, 0x29,
0xEB, 0x2B, 0x2A, 0xEA, 0xEE, 0x2E, 0x2F, 0xEF, 0x2D, 0xED,
0xEC, 0x2C, 0xE4, 0x24, 0x25, 0xE5, 0x27, 0xE7, 0xE6, 0x26,
0x22, 0xE2, 0xE3, 0x23, 0xE1, 0x21, 0x20, 0xE0, 0xA0, 0x60,
0x61, 0xA1, 0x63, 0xA3, 0xA2, 0x62, 0x66, 0xA6, 0xA7, 0x67,
0xA5, 0x65, 0x64, 0xA4, 0x6C, 0xAC, 0xAD, 0x6D, 0xAF, 0x6F,
0x6E, 0xAE, 0xAA, 0x6A, 0x6B, 0xAB, 0x69, 0xA9, 0xA8, 0x68,
0x78, 0xB8, 0xB9, 0x79, 0xBB, 0x7B, 0x7A, 0xBA, 0xBE, 0x7E,
0x7F, 0xBF, 0x7D, 0xBD, 0xBC, 0x7C, 0xB4, 0x74, 0x75, 0xB5,
0x77, 0xB7, 0xB6, 0x76, 0x72, 0xB2, 0xB3, 0x73, 0xB1, 0x71,
0x70, 0xB0, 0x50, 0x90, 0x91, 0x51, 0x93, 0x53, 0x52, 0x92,
0x96, 0x56, 0x57, 0x97, 0x55, 0x95, 0x94, 0x54, 0x9C, 0x5C,
0x5D, 0x9D, 0x5F, 0x9F, 0x9E, 0x5E, 0x5A, 0x9A, 0x9B, 0x5B,
0x99, 0x59, 0x58, 0x98, 0x88, 0x48, 0x49, 0x89, 0x4B, 0x8B,
0x8A, 0x4A, 0x4E, 0x8E, 0x8F, 0x4F, 0x8D, 0x4D, 0x4C, 0x8C,
0x44, 0x84, 0x85, 0x45, 0x87, 0x47, 0x46, 0x86, 0x82, 0x42,
0x43, 0x83, 0x41, 0x81, 0x80, 0x40
} ;

unsigned char auchCRCHi; /* high CRC byte */
unsigned char auchCRCLo; /* low CRC byte */

// Timer 0 overflow interrupt service routine
interrupt [TIM0_OVF] void timer0_ovf_isr(void)
{
        // Reinitialize Timer 0 value
        TCNT0=0x00;
// Place your code here
//FWdtPulse(1);


}


// External Interrupt 0 service routine
interrupt [EXT_INT0] void ext_int0_isr(void)
{unsigned int x=0;
}

void main(void)
{
// Reset Source checking
if (MCUCSR & 1)
```

```
  {
  // Power-on Reset
  MCUCSR=0;
  // Place your code here
  //Primeiros Bytes para Reg. de controle: 0=UltimoRegHig#, 1=UltimoRegLow,
2=TempoRegDataLog*, 3=NResetsExternos, 4=NPower-OnReset
  //i2c_init();
  //TmpBoot=eeprom_read(4);
  //if (TmpBoot<253){eeprom_write(4,TmpBoot++);}//4=NPower-OnReset

  }
else if (MCUCSR & 2)
  {
  // External Reset
  MCUCSR=0;
  // Place your code here
  //Primeiros Bytes para Reg. de controle: 0=UltimoRegHig#, 1=UltimoRegLow,
2=TempoRegDataLog*, 3=NResetsExternos, 4=NPower-OnReset
  //i2c_init();
  ///TmpBoot=eeprom_read(3);
  //if (TmpBoot<253){eeprom_write(3,TmpBoot++);}//3=NResetsExternos

  }
else if (MCUCSR & 4)
  {
  // Brown-Out Reset
  MCUCSR=0;
  // Place your code here

  }
else
  {
  // Watchdog Reset
  MCUCSR=0;
  // Place your code here

  };


// Input/Output Ports initialization
// Port A initialization
// Func0=In Func1=In Func2=In Func3=Out Func4=In Func5=In Func6=In Func7=In
// State0=T State1=T State2=T State3=0 State4=T State5=T State6=T State7=T
PORTA=0x00;
DDRA=0x08;

// Port B initialization
// Func0=In Func1=In Func2=In Func3=In Func4=In Func5=In Func6=Out Func7=In
// State0=T State1=T State2=T State3=T State4=T State5=P State6=0 State7=P
PORTB=0xA0;
DDRB=0x40;

// Port C initialization
// Func0=In Func1=In Func2=In Func3=Out Func4=In Func5=In Func6=In Func7=In
// State0=T State1=T State2=T State3=0 State4=T State5=T State6=T State7=T
PORTC=0x00;
DDRC=0x08;

// Port D initialization
// Func0=In Func1=Out Func2=In Func3=Out Func4=In Func5=In Func6=In Func7=In
```

```
// State0=T State1=0 State2=P State3=0 State4=P State5=T State6=T State7=T
PORTD=0x14;
DDRD=0x0A;

// Timer/Counter 0 initialization
// Clock source: System Clock
// Clock value: 1000,000 kHz
// Mode: Normal top=FFh
// OC0 output: Disconnected
TCCR0=0x02;
TCNT0=0x30;
OCR0=0x00;

// Timer/Counter 1 initialization
// Clock source: System Clock
// Clock value: Timer 1 Stopped
// Mode: Normal top=FFFFh
// OC1A output: Discon.
// OC1B output: Discon.
// Noise Canceler: Off
// Input Capture on Falling Edge
TCCR1A=0x00;
TCCR1B=0x00;
TCNT1H=0x00;
TCNT1L=0x00;
OCR1AH=0x00;
OCR1AL=0x00;
OCR1BH=0x00;
OCR1BL=0x00;

// Timer/Counter 2 initialization
// Clock source: System Clock
// Clock value: Timer 2 Stopped
// Mode: Normal top=FFh
// OC2 output: Disconnected
ASSR=0x00;
TCCR2=0x00;
TCNT2=0x00;
OCR2=0x00;

// INT0: On
// INT0 Mode: Low level
// INT1: Off
// INT2: Off
GICR|=0x40;
MCUCR=0x00;
MCUCSR=0x00;
GIFR=0x40;


// Timer(s)/Counter(s) Interrupt(s) initialization
TIMSK=0x01;

// Analog Comparator initialization
// Analog Comparator: Off
// Analog Comparator Input Capture by Timer/Counter 1: Off
ACSR=0x80;
SFIOR=0x00;

// ADC initialization
```

```
// ADC Clock frequency: 125,000 kHz
// ADC Voltage Reference: AVCC pin
ADMUX=ADC_VREF_TYPE;
ADCSRA=0x85;
//ADCSRA=0x87;


// LCD module initialization
lcd_init(20);

// Watchdog Timer initialization
// Watchdog Timer Prescaler: OSC/2048
//WDTCR=0x0F;

// Global enable interrupts
//#asm("sei")
//delay_ms(1000);
//#asm("cli")
//LeSensor();


// USART initialization
// Communication Parameters: 8 Data, 1 Stop, No Parity
// USART Receiver: On
// USART Transmitter: On
// USART Mode: Asynchronous
// USART Baud rate: 2400
UCSRA=0x00;
UCSRB=0x98;
UCSRC=0x86;
UBRRH=0x00;
UBRRL=0xCF;


lcd_clear();

lcd_putsf("  SISTEMA DE REDE   ");
//lcd_putsf("12345678901234567890");
lcd_gotoxy(0,1);
lcd_putsf("   VERSAO   1.00    ");
delay_ms(1500);

lcd_gotoxy(0,0);
lcd_putsf(" TRABALHO CONCLUSAO ");

lcd_gotoxy(0,1);
lcd_putsf("    JOAO MARCELO    ");
delay_ms(1000);



#asm("sei")

while(1)
{

        crc16(&rx_buffer[0],3);//efetua a cálculo de CRC
        if ((rx_buffer[3] == auchCRCLo) & (rx_buffer[4] == auchCRCHi) & (FlagBufferNovo==1))
        {
                FlagBufferNovo=0;
```

```
                rx_wr_index=0;
                rx_rd_index=0;
                rx_counter=0;
                rx_buffer_overflow=0;

                if (rx_buffer[0]==CPU_Principal)
                {
                        if (SequenciaComando==1)//Leitura de Temperatura
                        {
                                Temperatura = rx_buffer[2];
                        }

                        if (SequenciaComando==2)//Leitura de Umdade
                        {
                                Umidade = rx_buffer[2];
                        }

                        if (SequenciaComando<=3)//Incrementa a sequencia de comando
                        {
                                SequenciaComando++;
                        }
                        else
                        {
                                SequenciaComando=1;
                        }

                      delay_ms(1000);

                        lcd_clear();
                        sprintf(text,"TEMP.:%u  UMIDADE:%u",Temperatura,Umidade);
                        lcd_puts(text);
                        lcd_gotoxy(0,1);
                        lcd_putsf("    BUFFER RX OK    ");
                }
                ZeraBuffer();

        }
        else
        {
//Byte2 = Informação (dados): 0=Temperatura ou Desligar relé, 1=Umidade ou Ligar relé,
2=Status(status do relé), 254=CRC Inválido, 255=Comando inválido
                if (SequenciaComando==1)//Leitura de temperatura
                {
                        TX_RX = 1;
                        delay_ms(10);
                        comunica_rede(CPU_Sensor, 1, 0);//Leitura Temperatura
                        delay_ms(10);
                        TX_RX = 0;
                        lcd_gotoxy(0,1);
                        lcd_putsf("TX...     CPU_Sensor");

                }
                if (SequenciaComando==2)//Leitura de umidade
                {
                        TX_RX = 1;
                        delay_ms(10);
                        comunica_rede(CPU_Sensor, 1, 1);//Leitura Umidade
                        delay_ms(10);
                        TX_RX = 0;
                        lcd_gotoxy(0,1);
```

```
                lcd_putsf("TX...     CPU_Sensor");
        }

        if (SequenciaComando==3)//Liga/Desliga SaidaRele1 dependendo da umidade
Umidade
        {
                if (Umidade <75)//Liga SaidaRele1
                {
                        TX_RX = 1;
                        delay_ms(10);
                        comunica_rede(CPU_AcionaRele, 1, 1);//Liga SaidaRele1
                        delay_ms(10);
                        TX_RX = 0;
                }
                else
                {
                        if (Umidade >= 90)//Desliga SaidaRele1
                        {
                                TX_RX = 1;
                                delay_ms(10);
                                comunica_rede(CPU_AcionaRele, 1, 0);//Desliga SaidaRele1
                                delay_ms(10);
                                TX_RX = 0;
                        }
                        else
                        {
                if (SequenciaComando<=3)//Incrementa a sequencia de comando
                {
                        SequenciaComando++;
                }
                else
                {
                        SequenciaComando=1;
                }

                        }
                }


                lcd_gotoxy(0,1);
                lcd_putsf("TX... CPU_AcionaRele");
        }

        if (SequenciaComando==4)//Liga/Desliga SaidaRele2
        {
                if (Temperatura > 23)//Liga SaidaRele2
                {
                        TX_RX = 1;
                        delay_ms(10);
                        comunica_rede(CPU_AcionaRele, 2, 1);//Liga SaidaRele2
                        delay_ms(10);
                        TX_RX = 0;
                }
                else
                {
                        if (Temperatura <= 18)//Desliga SaidaRele2
                        {
                                TX_RX = 1;
                                delay_ms(10);
                                comunica_rede(CPU_AcionaRele, 2, 0);//Desliga SaidaRele2
```

```
                                        delay_ms(10);
                                        TX_RX = 0;
                                }
                                else
                                {
                        if (SequenciaComando<=3)//Incrementa a sequencia de comando
                        {
                                SequenciaComando++;
                        }
                        else
                        {
                                SequenciaComando=1;
                        }

                                }
                        }



                        lcd_gotoxy(0,1);
                        lcd_putsf("TX... CPU_AcionaRele");
                }

                //lcd_gotoxy(0,1);
                //lcd_putsf("   COMUNICANDO...   ");
        }
        delay_ms(600);
}


}//MAIN

void Reiniciar(void)
{
    #asm("RJMP __RESET"); // Reset !!!
}



void comunica_rede(unsigned char CLP, unsigned char Funcao, unsigned char Informacao)
{unsigned int Tmp;

//ENVIA DADOS PARA REDE

      tx_buffer[0]=CLP;
      tx_buffer[1]=Funcao;
         tx_buffer[2]=Informacao;

         crc16(&tx_buffer[0],3);//efetua o cálculo de CRC
         tx_buffer[3]=auchCRCLo;//Byte3 = Primeiro (High) Byte do digito verificador CRC16 (Cyclic
Redundancy Check)
         tx_buffer[4]=auchCRCHi;//Byte4 = Segundo (Low) Byte do digito verificador CRC16 (Cyclic
Redundancy Check)
         //tx_buffer[4]=1;//Byte4 = Segundo (Low) Byte do digito verificador CRC16 (Cyclic
Redundancy Check)

         for (Tmp=0; Tmp<= 4; Tmp++)//loop para envio do tx_buffer
         {
                printf("%c",tx_buffer[Tmp]);//envia dados para rede
         }
```

```
}


void crc16(unsigned char *uchMsg, unsigned short usDataLen)//Função para cálculo de CRC
{
      unsigned uIndex ;             /* will index into CRC lookup*/
      auchCRCHi = 0xFF; /* high CRC byte */
      auchCRCLo = 0xFF; /* low CRC byte */
                              /* table */
      while (usDataLen--)           /* pass through message buffer */
            {
            uIndex = auchCRCHi ^ *uchMsg++ ;  /* calculate the CRC */
            auchCRCHi = auchCRCLo ^ uchCRCHi[uIndex] ;
            auchCRCLo = uchCRCLo[uIndex] ;
            }
}

void ZeraBuffer(void)
{
          rx_buffer[0]=0;
          rx_buffer[1]=0;
          rx_buffer[2]=0;
          rx_buffer[3]=0;
          rx_buffer[4]=0;
```

Apêndice C

Programa linguagem C – CPU_Sensor

```
/*
******** PROTOCOLO EXCLUSIVO DE COMUNICAÇÃO DA REDE *******
Byte0 = Endereço do CLP de destino
Byte1 = Função (comando): 1=Leitura(solicitação de informação) ou SaídaRele1, 2=Escrita(comando
remoto) ou SaídaRele2
Byte2 = Informação (dados): 0=Temperatura ou Desligar relé, 1=Umidade ou Ligar relé,
2=Status(status do relé), 254=CRC Inválido, 255=Comando inválido
Byte3 = Primeiro (High) Byte do digito verificador CRC16 (Cyclic Redundancy Check)
Byte4 = Segundo (Low) Byte do digito verificador CRC16 (Cyclic Redundancy Check)

********* ENDEREÇO DOS CLP´S *********
1 = CPU_Principal
2 = CPU_Sensor
3 = CPU_AcionaRele
*/

#include <mega8.h>

#include <delay.h>
//#include <stdio.h>
#include <math.h>

//#include <sensirion.h>


#define SaidaRele1              PORTB.0
#define SaidaRele2              PORTB.1
#define LedIndicador            PORTB.2
#define TX_RX                   PORTB.6

eeprom char CPU_Principal=1;
eeprom char CPU_Sensor=2;
eeprom char CPU_AcionaRele=3;


#define RXB8 1
#define TXB8 0
#define UPE 2
#define OVR 3
#define FE 4
#define UDRE 5
#define RXC 7

#define FRAMING_ERROR (1<<FE)
#define PARITY_ERROR (1<<UPE)
#define DATA_OVERRUN (1<<OVR)
#define DATA_REGISTER_EMPTY (1<<UDRE)
#define RX_COMPLETE (1<<RXC)


//#########   VAIAVEIS PARA COMUNICAÇÃO DA REDE   ##########
unsigned char tx_buffer[255];//USADO COMO BUFFER PARA ENVIAR PARA REDE
char FlagBufferNovo=0;//Usado para identificar a chegada de novo Buffer
```

```
// USART Receiver buffer
#define RX_BUFFER_SIZE 8//tamanho do buffer na rede
char rx_buffer[RX_BUFFER_SIZE];
unsigned char rx_wr_index,rx_rd_index,rx_counter;
// This flag is set on USART Receiver buffer overflow
bit rx_buffer_overflow;

// USART Receiver interrupt service routine
#pragma savereg-
interrupt [USART_RXC] void usart_rx_isr(void)
{
char status,data;
#asm
    push r26
    push r27
    push r30
    push r31
    in   r26,sreg
    push r26
#endasm
status=UCSRA;
data=UDR;
FlagBufferNovo=1;
if ((status & (FRAMING_ERROR | PARITY_ERROR | DATA_OVERRUN))==0)
   {
   rx_buffer[rx_wr_index]=data;
   if (++rx_wr_index == RX_BUFFER_SIZE) rx_wr_index=0;
   if (++rx_counter == RX_BUFFER_SIZE)
      {
      rx_counter=0;
      rx_buffer_overflow=1;
      };
   };
#asm
    pop  r26
    out  sreg,r26
    pop  r31
    pop  r30
    pop  r27
    pop  r26
#endasm

}
#pragma savereg+

#ifndef _DEBUG_TERMINAL_IO_
// Get a character from the USART Receiver buffer
#define _ALTERNATE_GETCHAR_
#pragma used+
char getchar(void)
{
char data;
while (rx_counter==0);
data=rx_buffer[rx_rd_index];
if (++rx_rd_index == RX_BUFFER_SIZE) rx_rd_index=0;
#asm("cli")
--rx_counter;
#asm("sei")
return data;
}
```

```
#pragma used-
#endif

// Standard Input/Output functions
#include <stdio.h>

void Reiniciar(void);
void crc16(unsigned char *uchMsg, unsigned short usDataLen);//Função para cálculo de CRC
void comunica_rede(unsigned char CLP, unsigned char Funcao, unsigned char Informacao);
void ZeraBuffer(void);


//******** FUNÇÕES E VARIÁVEIS DO SENSIRION
//volatile char unidades,dezenas,centenas;
void le_sensirion(void);
unsigned char unidades,dezenas,centenas;


char SHT_WriteByte(unsigned char value);
char SHT_ReadByte(unsigned char ack);
void s_transstart(void);
void s_connectionreset(void);
char s_softreset(void);
char s_measure(unsigned char *p_value, unsigned char *p_checksum, unsigned char mode);
void calc_sth11(float *p_humidity ,float *p_temperature);
//float calc_dewpoint(float h,float t);

char s_write_statusreg(unsigned char *p_value);
//char s_read_statusreg(unsigned char *p_value, unsigned char *p_checksum);

enum {TEMP,HUMI};

typedef union
{ unsigned int i; float f;} value;

sfrb    PINB = 0x16;
sfrb    PORTB = 0x18;
sfrb    DDRB = 0x17;

#define         SHT_DATA_OUT        DDRC.0
#define         SHT_DATA_IN         PINC.0
#define         SHT_SCK                     PORTC.1
//#define               HEAT_SW                     PINB.2                  // Heater On or Off
#define         noACK 0
#define         ACK             1
                  //adr  command  r/w
#define STATUS_REG_W 0x06   //000   0011    0
#define STATUS_REG_R 0x07   //000   0011    1
#define MEASURE_TEMP 0x03   //000   0001    1
#define MEASURE_HUMI 0x05   //000   0010    1
#define RESET       0x1e   //000   1111    0

value humi_val, temp_val;
unsigned char error, checksum, status;



/* Table of CRC values for high-order byte */
const static unsigned char uchCRCHi[] = {
```

```
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1,
0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1,
0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40,
0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1,
0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40,
0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40,
0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1,
0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40
} ;

/* Table of CRC values for low-order byte */
const static char uchCRCLo[] = {
0x00, 0xC0, 0xC1, 0x01, 0xC3, 0x03, 0x02, 0xC2, 0xC6, 0x06,
0x07, 0xC7, 0x05, 0xC5, 0xC4, 0x04, 0xCC, 0x0C, 0x0D, 0xCD,
0x0F, 0xCF, 0xCE, 0x0E, 0x0A, 0xCA, 0xCB, 0x0B, 0xC9, 0x09,
0x08, 0xC8, 0xD8, 0x18, 0x19, 0xD9, 0x1B, 0xDB, 0xDA, 0x1A,
0x1E, 0xDE, 0xDF, 0x1F, 0xDD, 0x1D, 0x1C, 0xDC, 0x14, 0xD4,
0xD5, 0x15, 0xD7, 0x17, 0x16, 0xD6, 0xD2, 0x12, 0x13, 0xD3,
0x11, 0xD1, 0xD0, 0x10, 0xF0, 0x30, 0x31, 0xF1, 0x33, 0xF3,
0xF2, 0x32, 0x36, 0xF6, 0xF7, 0x37, 0xF5, 0x35, 0x34, 0xF4,
0x3C, 0xFC, 0xFD, 0x3D, 0xFF, 0x3F, 0x3E, 0xFE, 0xFA, 0x3A,
0x3B, 0xFB, 0x39, 0xF9, 0xF8, 0x38, 0x28, 0xE8, 0xE9, 0x29,
0xEB, 0x2B, 0x2A, 0xEA, 0xEE, 0x2E, 0x2F, 0xEF, 0x2D, 0xED,
0xEC, 0x2C, 0xE4, 0x24, 0x25, 0xE5, 0x27, 0xE7, 0xE6, 0x26,
0x22, 0xE2, 0xE3, 0x23, 0xE1, 0x21, 0x20, 0xE0, 0xA0, 0x60,
0x61, 0xA1, 0x63, 0xA3, 0xA2, 0x62, 0x66, 0xA6, 0xA7, 0x67,
0xA5, 0x65, 0x64, 0xA4, 0x6C, 0xAC, 0xAD, 0x6D, 0xAF, 0x6F,
0x6E, 0xAE, 0xAA, 0x6A, 0x6B, 0xAB, 0x69, 0xA9, 0xA8, 0x68,
0x78, 0xB8, 0xB9, 0x79, 0xBB, 0x7B, 0x7A, 0xBA, 0xBE, 0x7E,
0x7F, 0xBF, 0x7D, 0xBD, 0xBC, 0x7C, 0xB4, 0x74, 0x75, 0xB5,
0x77, 0xB7, 0xB6, 0x76, 0x72, 0xB2, 0xB3, 0x73, 0xB1, 0x71,
0x70, 0xB0, 0x50, 0x90, 0x91, 0x51, 0x93, 0x53, 0x52, 0x92,
0x96, 0x56, 0x57, 0x97, 0x55, 0x95, 0x94, 0x54, 0x9C, 0x5C,
0x5D, 0x9D, 0x5F, 0x9F, 0x9E, 0x5E, 0x5A, 0x9A, 0x9B, 0x5B,
0x99, 0x59, 0x58, 0x98, 0x88, 0x48, 0x49, 0x89, 0x4B, 0x8B,
0x8A, 0x4A, 0x4E, 0x8E, 0x8F, 0x4F, 0x8D, 0x4D, 0x4C, 0x8C,
0x44, 0x84, 0x85, 0x45, 0x87, 0x47, 0x46, 0x86, 0x82, 0x42,
0x43, 0x83, 0x41, 0x81, 0x80, 0x40
} ;

unsigned char auchCRCHi; /* high CRC byte */
unsigned char auchCRCLo; /* low CRC byte */
```

```
// Timer 0 overflow interrupt service routine
interrupt [TIM0_OVF] void timer0_ovf_isr(void)
{//unsigned char FlagMonitoraTecla=0;

// Reinitialize Timer 0 value
TCNT0=0x30;
// Place your code here

}


// Timer 2 overflow interrupt service routine
interrupt [TIM2_OVF] void timer2_ovf_isr(void)
{
// Reinitialize Timer 2 value
TCNT2=0x30;
// Place your code here

}


void main(void)
{


// Input/Output Ports initialization
// Port B initialization
// Func0=Out Func1=Out Func2=Out Func3=Out Func4=Out Func5=Out Func6=Out Func7=In
// State0=0 State1=0 State2=0 State3=0 State4=0 State5=0 State6=0 State7=P
PORTB=0x80;
DDRB=0x7F;


// Port C initialization
// Func0=In Func1=In Func2=In Func3=In Func4=In Func5=In Func6=Out
// State0=T State1=T State2=T State3=T State4=T State5=T State6=0
PORTC=0x00;
DDRC=0x40;

// Port D initialization
// Func0=In Func1=Out Func2=In Func3=In Func4=In Func5=In Func6=Out Func7=Out
// State0=P State1=0 State2=P State3=P State4=T State5=T State6=0 State7=0
PORTD=0x0D;
DDRD=0xC2;

// Timer/Counter 0 initialization
// Clock source: System Clock
// Clock value: 500,000 kHz
TCCR0=0x02;
TCNT0=0x30;


// Timer/Counter 1 initialization
// Clock source: System Clock
// Clock value: Timer 1 Stopped
// Mode: Normal top=FFFFh
// OC1A output: Discon.
// OC1B output: Discon.
```

```
// Noise Canceler: Off
// Input Capture on Falling Edge
TCCR1A=0x00;
TCCR1B=0x00;
TCNT1H=0x00;
TCNT1L=0x00;
OCR1AH=0x00;
OCR1AL=0x00;
OCR1BH=0x00;
OCR1BL=0x00;

// Timer/Counter 2 initialization
// Clock source: System Clock
// Clock value: Timer 2 Stopped
// Mode: Normal top=FFh
// OC2 output: Disconnected
ASSR=0x00;
TCCR2=0x00;
TCNT2=0x00;
OCR2=0x00;

// External Interrupt(s) initialization
// INT0: Off
// INT1: Off
MCUCR=0x00;

// Timer(s)/Counter(s) Interrupt(s) initialization
//TIMSK=0x41;

// Timer(s)/Counter(s) Interrupt(s) initialization
TIMSK=0x05;//1

// Analog Comparator initialization
// Analog Comparator: Off
// Analog Comparator Input Capture by Timer/Counter 1: Off
// Analog Comparator Output: Off
ACSR=0x80;
SFIOR=0x00;

// USART initialization
// Communication Parameters: 8 Data, 1 Stop, No Parity
// USART Receiver: On
// USART Transmitter: On
// USART Mode: Asynchronous
// USART Baud rate: 2400
UCSRA=0x00;
UCSRB=0x98;
UCSRC=0x86;
UBRRH=0x00;
UBRRL=0x67;



// Watchdog Timer initialization
// Watchdog Timer Prescaler: OSC/2048k
//WDTCR=0x1F;
//WDTCR=0x0F;

// Setup Sensibus Pins
PORTC.1         = 0;      // ClockLow
```

```
DDRC.1 = 1;             // SCK is an output
PORTC.0 = 0;  // Always Zero
                // Toggle DDRB.0 for Data
s_transstart();
delay_ms(200);
if (s_softreset())
{
        LedIndicador=1;
        delay_ms(1500);
        LedIndicador=0;
        delay_ms(200);
        LedIndicador=1;
        delay_ms(1500);
        LedIndicador=0;
        delay_ms(200);
}

s_connectionreset();


// Global enable interrupts
#asm("sei")




while(1)
{
        status = 0b00000000; //Leitura normal
        le_sensirion();
        crc16(&rx_buffer[0],3);//efetua a cálculo de CRC
        if ((rx_buffer[3] == auchCRCLo) & (rx_buffer[4] == auchCRCHi) & (FlagBufferNovo==1))
        {
                FlagBufferNovo=0;
                rx_buffer_overflow=0;
                rx_wr_index=0;
                rx_rd_index=0;
                rx_counter=0;
                if (rx_buffer[0]==CPU_Sensor)
                {
                        LedIndicador=1;

                        if (rx_buffer[1]==1 & rx_buffer[2]==0)//Leitura Temperatura
                        {
                                TX_RX = 1;
                                delay_ms(10);
                                comunica_rede(CPU_Principal, 1, (char)temp_val.f);//CLP, Funcao,
Informacao)
                                delay_ms(10);
                                TX_RX = 0;
                        }

                        if (rx_buffer[1]==1 & rx_buffer[2]==1)//Leitura Umidade
                        {
                                TX_RX = 1;
                                delay_ms(10);
                                comunica_rede(CPU_Principal, 1, (char)humi_val.f);//CLP, Funcao,
Informacao)
                                delay_ms(10);
                                TX_RX = 0;
```

```
                        }
                    }
                    ZeraBuffer();
            }
            LedIndicador=0;
 }
}//MAIN

void Reiniciar(void)
{
     #asm("RJMP __RESET"); // Reset !!!
}

void comunica_rede(unsigned char CLP, unsigned char Funcao, unsigned char Informacao)
{unsigned int Tmp;
       tx_buffer[0]=CLP;
       tx_buffer[1]=Funcao;
           tx_buffer[2]=Informacao;

           crc16(&tx_buffer[0],3);//efetua a cálculo de CRC

           tx_buffer[3]=auchCRCLo;//Byte3 = Primeiro (High) Byte do digito verificador CRC16 (Cyclic
Redundancy Check)
           tx_buffer[4]=auchCRCHi;//Byte4 = Segundo (Low) Byte do digito verificador CRC16 (Cyclic
Redundancy Check)

           for (Tmp=0; Tmp<= 4; Tmp++)//loop para envio do tx_buffer
           {
                    //printf("%c",0);//envia dados para rede
                    printf("%c",tx_buffer[Tmp]);//envia dados para rede
           }
}

void crc16(unsigned char *uchMsg, unsigned short usDataLen)//Função para cálculo de CRC
{
       unsigned uIndex ;             /* will index into CRC lookup*/
       auchCRCHi = 0xFF; /* high CRC byte */
       auchCRCLo = 0xFF; /* low CRC byte */
                                /* table */
       while (usDataLen--)           /* pass through message buffer */
             {
             uIndex = auchCRCHi ^ *uchMsg++ ;  /* calculate the CRC */
             auchCRCHi = auchCRCLo ^ uchCRCHi[uIndex] ;
             auchCRCLo = uchCRCLo[uIndex] ;
             }
}

void ZeraBuffer(void)
{
           rx_buffer[0]=0;
           rx_buffer[1]=0;
           rx_buffer[2]=0;
           rx_buffer[3]=0;
           rx_buffer[4]=0;
}



//-------------------------------------------------------------------------------
// writes a byte on the Sensibus and checks the acknowledge
```

```
//-----------------------------------------------------------------------------
char SHT_WriteByte(unsigned char value)
{
  unsigned char i,error=0;
  for (i=0x80;i>0;i/=2)             //shift bit for masking
          {
          if (i & value)      SHT_DATA_OUT=0;               //masking value with i , write to SENSI-BUS
    else SHT_DATA_OUT=1;
    SHT_SCK=1;                        //clk for SENSI-BUS
    delay_us(50);                                                       //pulswith approx. 5 us
    SHT_SCK=0;
          }
  SHT_DATA_OUT=0;                   //release DATA-line
  SHT_SCK=1;                        //clk #9 for ack
  error=SHT_DATA_IN;                //check ack (DATA will be pulled down by SHT11)
  SHT_SCK=0;
  return error;                     //error=1 in case of no acknowledge
}

//-----------------------------------------------------------------------------
// reads a byte form the Sensibus and gives an acknowledge in case of "ack=1"
//-----------------------------------------------------------------------------
char SHT_ReadByte(unsigned char ack)
{
  unsigned char i,val=0;
  SHT_DATA_OUT=0;                                   //release DATA-line
  for (i=0x80;i>0;i/=2)             //shift bit for masking
    {
    SHT_SCK=1;                                      //clk for SENSI-BUS
    if (SHT_DATA_IN) val=(val | i);          //read bit
    SHT_SCK=0;
          }
  SHT_DATA_OUT=ack;                                 //in case of "ack==1" pull down DATA-Line
  SHT_SCK=1;                        //clk #9 for ack
  delay_us(50);                                             //pulswith approx. 5 us
  SHT_SCK=0;
  SHT_DATA_OUT=0;                                   //release DATA-line
  return val;
}

//-----------------------------------------------------------------------------
// generates a transmission start
//       _____         _________
// DATA:      |_______|
//           ___     ___
// SCK : ___|   |___|   |______
//-----------------------------------------------------------------------------
void s_transstart(void)
{
   SHT_DATA_OUT=0;
   SHT_SCK=0;                   //Initial state
   delay_us(10);
   SHT_SCK=1;
   delay_us(10);
   SHT_DATA_OUT=1;
   delay_us(10);
   SHT_SCK=0;
   delay_us(50);
   SHT_SCK=1;
   delay_us(10);
```

```
  SHT_DATA_OUT=0;
  delay_us(10);
  SHT_SCK=0;
}

//----------------------------------------------------------------------------
// communication reset: DATA-line=1 and at least 9 SCK cycles followed by transstart
//      _____________________________________________________         ________
// DATA:                                                |_______|
//          _   _   _   _   _   _   _   _   _   ___     ___
// SCK : __| |__| |__| |__| |__| |__| |__| |__| |__| |_____|   |___|   |______
//----------------------------------------------------------------------------

void s_connectionreset(void)
{
  unsigned char i;
  SHT_DATA_OUT=0; SHT_SCK=0;     //Initial state
  for(i=0;i<9;i++)                //9 SCK cycles
          {
          SHT_SCK=1;
          delay_us(10);
    SHT_SCK=0;
  }
  s_transstart();                 //transmission start
}



//----------------------------------------------------------------------------
// resets the sensor by a softreset
//----------------------------------------------------------------------------

char s_softreset(void)
{
  unsigned char error=0;
  s_connectionreset();              //reset communication
  error+=SHT_WriteByte(RESET);      //send RESET-command to sensor
  return error;                     //error=1 in case of no response form the sensor
}



//----------------------------------------------------------------------------
// makes a measurement (humidity/temperature) with checksum
//----------------------------------------------------------------------------
char s_measure(unsigned char *p_value, unsigned char *p_checksum, unsigned char mode)
{
  unsigned error=0;
  unsigned int i;

  s_transstart();                   //transmission start
  switch(mode){                       //send command to sensor
    case TEMP   : error+=SHT_WriteByte(MEASURE_TEMP); break;
    case HUMI   : error+=SHT_WriteByte(MEASURE_HUMI); break;
    default     : break;
  }
  for (i=0;i<65535;i++) if(SHT_DATA_IN==0) break; //wait until sensor has finished the measurement
  if(SHT_DATA_IN) error+=1;                // or timeout (~2 sec.) is reached
```

```
  *(p_value+1)  =SHT_ReadByte(ACK);    //read the first byte (MSB)
  *(p_value)  =SHT_ReadByte(ACK);    //read the second byte (LSB)
  *p_checksum =SHT_ReadByte(noACK);  //read checksum
  return error;
}

//-----------------------------------------------------------------------------------------
// calculates temperature [°C] and humidity [%RH]
// input :  humi [Ticks] (12 bit)
//          temp [Ticks] (14 bit)
// output:  humi [%RH]
//          temp [°C]
//-----------------------------------------------------------------------------------------

void calc_sth11(float *p_humidity ,float *p_temperature)
{

  //float rh=*p_humidity;             // rh:      Humidity [Ticks] 12 Bit
  //float t=*p_temperature;           // t:       Temperature [Ticks] 14 Bit
  float rh_lin;                     // rh_lin:  Humidity linear
  float rh_true;                    // rh_true: Temperature compensated humidity
  float t_C;                        // t_C   :  Temperature [°C]

  t_C=*p_temperature*0.01 - 40;                 //calc. temperature from ticks to [°C]

/*
const float C1=-4.0;              // for 12 Bit
const float C2=+0.0405;           // for 12 Bit
const float C3=-0.0000028;        // for 12 Bit
const float T1=+0.01;             // for 14 Bit @ 5V
const float T2=+0.00008;          // for 14 Bit @ 5V
  */


  //rh_lin=C3*(*p_humidity)*(*p_humidity) + C2*(*p_humidity) + C1;     //calc. humidity from ticks to
[%RH]
  rh_lin=(-0.0000028)*(*p_humidity)*(*p_humidity) + 0.0405*(*p_humidity) + (-4.0);     //calc. humidity
from ticks to [%RH]
  rh_true=(t_C-25)*(0.01+0.00008*(*p_humidity))+rh_lin;   //calc. temperature compensated humidity
[%RH]
  if(rh_true>100)rh_true=100;       //cut if the value is outside of
  if(rh_true<0.1)rh_true=0.1;       //the physical possible range

  *p_temperature=t_C;               //return temperature [°C]
  *p_humidity=rh_true;              //return humidity[%RH]
}
/*
//--------------------------------------------------------------------
// calculates dew point
// input:   humidity [%RH], temperature [°C]
// output:  dew point [°C]
//--------------------------------------------------------------------

float calc_dewpoint(float h,float t)
{
  float logEx,dew_point;
  logEx=0.66077+7.5*t/(237.3+t)+(log10(h)-2);
  dew_point = (logEx - 0.66077)*237.3/(0.66077+7.5-logEx);
  return dew_point;
}
```

```
*/

/*
//---------------------------------------------------------------------------
// reads the status register with checksum (8-bit)
//---------------------------------------------------------------------------
char s_read_statusreg(unsigned char *p_value, unsigned char *p_checksum)
{
  unsigned char error=0;
  s_transstart();                  //transmission start
  error=SHT_WriteByte(STATUS_REG_R); //send command to sensor
  *p_value=SHT_ReadByte(ACK);        //read status register (8-bit)
  *p_checksum=SHT_ReadByte(noACK);   //read checksum (8-bit)
  return error;                    //error=1 in case of no response form the sensor
}
*/

//---------------------------------------------------------------------------
// writes the status register with checksum (8-bit)
//---------------------------------------------------------------------------
char s_write_statusreg(unsigned char *p_value)
{
  unsigned char error=0;
  s_transstart();                  //transmission start
  error+=SHT_WriteByte(STATUS_REG_W);//send command to sensor
  error+=SHT_WriteByte(*p_value);    //send value of status register
  return error;                    //error>=1 in case of no response form the sensor
}




void le_sensirion(void)
{
          s_transstart();
          error=0;
          s_write_statusreg(&status);
          error+=s_measure((unsigned char*) &humi_val.i,&checksum,HUMI);
       error+=s_measure((unsigned char*) &temp_val.i,&checksum,TEMP);
          if(error!=0) s_connectionreset();
          else{
                    humi_val.f=(float)humi_val.i;                  //converts integer to float
                    temp_val.f=(float)temp_val.i;                  //converts integer to float
                    calc_sth11(&humi_val.f,&temp_val.f);           //calculate humidity, temperature
                    //dew_point=calc_dewpoint(humi_val.f,temp_val.f); //calculate dew point
          } //else
}
```

Apêndice D

Programa linguagem C – CPU_AcionaRele

```
/*
******** PROTOCOLO EXCLUSIVO DE COMUNICAÇÃO DA REDE *******
Byte0 = Endereço do CLP de destino
Byte1 = Função (comando): 1=Leitura(solicitação de informação) ou SaídaRele1, 2=Escrita(comando
remoto) ou SaídaRele2
Byte2 = Informação (dados): 0=Temperatura ou Desligar relé, 1=Umidade ou Ligar relé,
2=Status(status do relé), 254=CRC Inválido, 255=Comando inválido
Byte3 = Primeiro (High) Byte do digito verificador CRC16 (Cyclic Redundancy Check)
Byte4 = Segundo (Low) Byte do digito verificador CRC16 (Cyclic Redundancy Check)

********* ENDEREÇO DOS CLP´S *********
1 = CPU_Principal
2 = CPU_Sensor
3 = CPU_AcionaRele
*/

#include <mega8.h>

#include <delay.h>

#define SaidaRele1              PORTB.0
#define SaidaRele2              PORTB.1
#define LedIndicador            PORTB.2
#define TX_RX                   PORTB.6

eeprom char CPU_Principal=1;
eeprom char CPU_Sensor=2;
eeprom char CPU_AcionaRele=3;


#define RXB8 1
#define TXB8 0
#define UPE 2
#define OVR 3
#define FE 4
#define UDRE 5
#define RXC 7

#define FRAMING_ERROR (1<<FE)
#define PARITY_ERROR (1<<UPE)
#define DATA_OVERRUN (1<<OVR)
#define DATA_REGISTER_EMPTY (1<<UDRE)
#define RX_COMPLETE (1<<RXC)


//#########   VAIAVEIS PARA COMUNICAÇÃO DA REDE   ##########
char tx_buffer[255];//USADO COMO BUFFER PARA ENVIAR PARA REDE
char FlagBufferNovo=0;//Usado para identificar a chegada de novo Buffer

// USART Receiver buffer
#define RX_BUFFER_SIZE 8//tamanho do buffer na rede
char rx_buffer[RX_BUFFER_SIZE];
unsigned char rx_wr_index,rx_rd_index,rx_counter;
```

```
// This flag is set on USART Receiver buffer overflow
bit rx_buffer_overflow;

// USART Receiver interrupt service routine
#pragma savereg-
interrupt [USART_RXC] void usart_rx_isr(void)
{
char status,data;
#asm
    push r26
    push r27
    push r30
    push r31
    in   r26,sreg
    push r26
#endasm
status=UCSRA;
data=UDR;
FlagBufferNovo=1;
if ((status & (FRAMING_ERROR | PARITY_ERROR | DATA_OVERRUN))==0)
  {
  rx_buffer[rx_wr_index]=data;
  if (++rx_wr_index == RX_BUFFER_SIZE) rx_wr_index=0;
  if (++rx_counter == RX_BUFFER_SIZE)
    {
    rx_counter=0;
    rx_buffer_overflow=1;
    };
  };
#asm
    pop  r26
    out  sreg,r26
    pop  r31
    pop  r30
    pop  r27
    pop  r26
#endasm
}
#pragma savereg+

#ifndef _DEBUG_TERMINAL_IO_
// Get a character from the USART Receiver buffer
#define _ALTERNATE_GETCHAR_
#pragma used+
char getchar(void)
{
char data;
while (rx_counter==0);
data=rx_buffer[rx_rd_index];
if (++rx_rd_index == RX_BUFFER_SIZE) rx_rd_index=0;
#asm("cli")
--rx_counter;
#asm("sei")
return data;
}
#pragma used-
#endif

// Standard Input/Output functions
#include <stdio.h>
```

```
void Reiniciar(void);
void crc16(unsigned char *uchMsg, unsigned short usDataLen);//Função para cálculo de CRC
void comunica_rede(unsigned char CLP, unsigned char Funcao, unsigned char Informacao);
void ZeraBuffer(void);

/* Table of CRC values for high-order byte */
const static unsigned char uchCRCHi[] = {
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1,
0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1,
0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40,
0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1,
0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40,
0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40,
0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1,
0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40
} ;

/* Table of CRC values for low-order byte */
const static char uchCRCLo[] = {
0x00, 0xC0, 0xC1, 0x01, 0xC3, 0x03, 0x02, 0xC2, 0xC6, 0x06,
0x07, 0xC7, 0x05, 0xC5, 0xC4, 0x04, 0xCC, 0x0C, 0x0D, 0xCD,
0x0F, 0xCF, 0xCE, 0x0E, 0x0A, 0xCA, 0xCB, 0x0B, 0xC9, 0x09,
0x08, 0xC8, 0xD8, 0x18, 0x19, 0xD9, 0x1B, 0xDB, 0xDA, 0x1A,
0x1E, 0xDE, 0xDF, 0x1F, 0xDD, 0x1D, 0x1C, 0xDC, 0x14, 0xD4,
0xD5, 0x15, 0xD7, 0x17, 0x16, 0xD6, 0xD2, 0x12, 0x13, 0xD3,
0x11, 0xD1, 0xD0, 0x10, 0xF0, 0x30, 0x31, 0xF1, 0x33, 0xF3,
0xF2, 0x32, 0x36, 0xF6, 0xF7, 0x37, 0xF5, 0x35, 0x34, 0xF4,
0x3C, 0xFC, 0xFD, 0x3D, 0xFF, 0x3F, 0x3E, 0xFE, 0xFA, 0x3A,
0x3B, 0xFB, 0x39, 0xF9, 0xF8, 0x38, 0x28, 0xE8, 0xE9, 0x29,
0xEB, 0x2B, 0x2A, 0xEA, 0xEE, 0x2E, 0x2F, 0xEF, 0x2D, 0xED,
0xEC, 0x2C, 0xE4, 0x24, 0x25, 0xE5, 0x27, 0xE7, 0xE6, 0x26,
0x22, 0xE2, 0xE3, 0x23, 0xE1, 0x21, 0x20, 0xE0, 0xA0, 0x60,
0x61, 0xA1, 0x63, 0xA3, 0xA2, 0x62, 0x66, 0xA6, 0xA7, 0x67,
0xA5, 0x65, 0x64, 0xA4, 0x6C, 0xAC, 0xAD, 0x6D, 0xAF, 0x6F,
0x6E, 0xAE, 0xAA, 0x6A, 0x6B, 0xAB, 0x69, 0xA9, 0xA8, 0x68,
0x78, 0xB8, 0xB9, 0x79, 0xBB, 0x7B, 0x7A, 0xBA, 0xBE, 0x7E,
0x7F, 0xBF, 0x7D, 0xBD, 0xBC, 0x7C, 0xB4, 0x74, 0x75, 0xB5,
0x77, 0xB7, 0xB6, 0x76, 0x72, 0xB2, 0xB3, 0x73, 0xB1, 0x71,
0x70, 0xB0, 0x50, 0x90, 0x91, 0x51, 0x93, 0x53, 0x52, 0x92,
0x96, 0x56, 0x57, 0x97, 0x55, 0x95, 0x94, 0x54, 0x9C, 0x5C,
0x5D, 0x9D, 0x5F, 0x9F, 0x9E, 0x5E, 0x5A, 0x9A, 0x9B, 0x5B,
0x99, 0x59, 0x58, 0x98, 0x88, 0x48, 0x49, 0x89, 0x4B, 0x8B,
```

```
0x8A, 0x4A, 0x4E, 0x8E, 0x8F, 0x4F, 0x8D, 0x4D, 0x4C, 0x8C,
0x44, 0x84, 0x85, 0x45, 0x87, 0x47, 0x46, 0x86, 0x82, 0x42,
0x43, 0x83, 0x41, 0x81, 0x80, 0x40
} ;

unsigned char auchCRCHi; /* high CRC byte */
unsigned char auchCRCLo; /* low CRC byte */


// Timer 0 overflow interrupt service routine
interrupt [TIM0_OVF] void timer0_ovf_isr(void)
{//unsigned char FlagMonitoraTecla=0;

// Reinitialize Timer 0 value
TCNT0=0x30;
// Place your code here

}


// Timer 2 overflow interrupt service routine
interrupt [TIM2_OVF] void timer2_ovf_isr(void)
{
// Reinitialize Timer 2 value
TCNT2=0x30;
// Place your code here

}


void main(void)
{
// Input/Output Ports initialization
// Port B initialization
// Func0=Out Func1=Out Func2=Out Func3=Out Func4=Out Func5=Out Func6=Out Func7=In
// State0=0 State1=0 State2=0 State3=0 State4=0 State5=0 State6=0 State7=P
PORTB=0x80;
DDRB=0x7F;

// Port C initialization
// Func0=In Func1=In Func2=In Func3=In Func4=In Func5=In Func6=In
// State0=T State1=T State2=T State3=T State4=T State5=T State6=T
PORTC=0x00;
DDRC=0x00;

// Port D initialization
// Func0=In Func1=Out Func2=In Func3=In Func4=In Func5=In Func6=Out Func7=Out
// State0=P State1=0 State2=P State3=P State4=T State5=T State6=0 State7=0
PORTD=0x0D;
DDRD=0xC2;

// Timer/Counter 0 initialization
// Clock source: System Clock
// Clock value: 500,000 kHz
TCCR0=0x02;
TCNT0=0x30;


// Timer/Counter 1 initialization
// Clock source: System Clock
```

```
// Clock value: Timer 1 Stopped
// Mode: Normal top=FFFFh
// OC1A output: Discon.
// OC1B output: Discon.
// Noise Canceler: Off
// Input Capture on Falling Edge
TCCR1A=0x00;
TCCR1B=0x00;
TCNT1H=0x00;
TCNT1L=0x00;
OCR1AH=0x00;
OCR1AL=0x00;
OCR1BH=0x00;
OCR1BL=0x00;

// Timer/Counter 2 initialization
// Clock source: System Clock
// Clock value: 3,906 kHz
// Mode: Normal top=FFh
// OC2 output: Disconnected
ASSR=0x00;
TCCR2=0x07;
TCNT2=0x30;
OCR2=0x00;

// External Interrupt(s) initialization
// INT0: Off
// INT1: Off
MCUCR=0x00;

// Timer(s)/Counter(s) Interrupt(s) initialization
TIMSK=0x41;

// Analog Comparator initialization
// Analog Comparator: Off
// Analog Comparator Input Capture by Timer/Counter 1: Off
// Analog Comparator Output: Off
ACSR=0x80;
SFIOR=0x00;

// USART initialization
// Communication Parameters: 8 Data, 1 Stop, No Parity
// USART Receiver: On
// USART Transmitter: On
// USART Mode: Asynchronous
// USART Baud rate: 2400
UCSRA=0x00;
UCSRB=0x98;
UCSRC=0x86;
UBRRH=0x00;
UBRRL=0x67;

// Watchdog Timer initialization
// Watchdog Timer Prescaler: OSC/2048k
//WDTCR=0x1F;
//WDTCR=0x0F;

// Global enable interrupts
#asm("sei")
//Carregador=1;
```

```
while(1)
{
	crc16(&rx_buffer[0],3);//efetua a cálculo de CRC
	if ((rx_buffer[3] == auchCRCLo) & (rx_buffer[4] == auchCRCHi) & (FlagBufferNovo==1))
	{
		FlagBufferNovo=0;
		rx_buffer_overflow=0;
		rx_wr_index=0;
		rx_rd_index=0;
		rx_counter=0;
		if (rx_buffer[0]==CPU_AcionaRele)
		{
			LedIndicador=1;

//Byte2 = Informação (dados): 0=Temperatura ou Desligar relé, 1=Umidade ou Ligar relé,
2=Status(status do relé), 254=CRC Inválido, 255=Comando inválido
			if (rx_buffer[1]==1 & rx_buffer[2]==0)//Desliga SaidaRele1
			{
				TX_RX = 1;
				delay_ms(10);
				comunica_rede(CPU_Principal, 2, 0);//Retorna o mesmo comando
para CPU_Principal entender que o comando foi aceito
				delay_ms(10);
				TX_RX = 0;
				SaidaRele1=0;
			}

			if (rx_buffer[1]==1 & rx_buffer[2]==1)//Liga SaidaRele1
			{
				TX_RX = 1;
				delay_ms(10);
				comunica_rede(CPU_Principal, 2, 1);//Retorna o mesmo comando
para CPU_Principal entender que o comando foi aceito
				delay_ms(10);
				TX_RX = 0;
				SaidaRele1=1;
			}

			if (rx_buffer[1]==2 & rx_buffer[2]==0)//Desliga SaidaRele2
			{
				TX_RX = 1;
				delay_ms(10);
				comunica_rede(CPU_Principal, 2, 0);//Retorna o mesmo comando
para CPU_Principal entender que o comando foi aceito
				delay_ms(10);
				TX_RX = 0;
				SaidaRele2=0;
			}

			if (rx_buffer[1]==2 & rx_buffer[2]==1)//Liga SaidaRele2
			{
				TX_RX = 1;
				delay_ms(10);
				comunica_rede(CPU_Principal, 2, 1);//Retorna o mesmo comando
para CPU_Principal entender que o comando foi aceito
				delay_ms(10);
				TX_RX = 0;
				SaidaRele2=1;
			}
		}
```

```
                ZeraBuffer();
        }


        if (LedIndicador==1)//delay para visualização do LedIndicador
        {
                //delay_ms(50);
                LedIndicador=0;
        }
 }
}//MAIN

void Reiniciar(void)
{
    #asm("RJMP __RESET"); // Reset !!!
}

void comunica_rede(unsigned char CLP, unsigned char Funcao, unsigned char Informacao)
{unsigned int Tmp;

     tx_buffer[0]=CLP;
     tx_buffer[1]=Funcao;
        tx_buffer[2]=Informacao;

        crc16(&tx_buffer[0],3);//efetua a cálculo de CRC

        tx_buffer[3]=auchCRCLo;//Byte3 = Primeiro (High) Byte do digito verificador CRC16 (Cyclic
Redundancy Check)
        tx_buffer[4]=auchCRCHi;//Byte4 = Segundo (Low) Byte do digito verificador CRC16 (Cyclic
Redundancy Check)

        for (Tmp=0; Tmp<= 4; Tmp++)//loop para envio do tx_buffer
        {
                //printf("%c",0);//envia dados para rede
                printf("%c",tx_buffer[Tmp]);//envia dados para rede
        }
}

void crc16(unsigned char *uchMsg, unsigned short usDataLen)//Função para cálculo de CRC
{
    unsigned uIndex ;           /* will index into CRC lookup*/
    auchCRCHi = 0xFF; /* high CRC byte */
    auchCRCLo = 0xFF; /* low CRC byte */
                        /* table */
    while (usDataLen--)         /* pass through message buffer */
        {
        uIndex = auchCRCHi ^ *uchMsg++ ;  /* calculate the CRC */
        auchCRCHi = auchCRCLo ^ uchCRCHi[uIndex] ;
        auchCRCLo = uchCRCLo[uIndex] ;
        }
}

void ZeraBuffer(void)
{
        rx_buffer[0]=0;
        rx_buffer[1]=0;
        rx_buffer[2]=0;
        rx_buffer[3]=0;
        rx_buffer[4]=0;
}
```