CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE SANTA CATARINA
PÓS-GRADUAÇÃO "LATU SENSU" EM DESENVOLVIMENTO DE PRODUTOS ELETRÔNICOS

JEAN CARLOS ESSER

DATA LOGGER – COLETOR DE DADOS

FLORIANÓPOLIS
2008

JEAN CARLOS ESSER

DATA LOGGER – COLETOR DE DADOS

Apresentação de monografia ao Centro Federal de Educação Tecnológica de Santa Catarina como condição prévia para a conclusão do Curso de Pós-Graduação “Lato Sensu” em Desenvolvimento de Produtos Eletrônicos.

Prof. Orientador: Muriel Bittencourt de Liz, Dr.

FLORIANÓPOLIS

2008

AGRADECIMENTOS

Aos mestres e colegas do curso de Pós-Graduação “Lato Sensu” em Desenvolvimento de Produtos Eletrônicos, ao meu professor orientador Prof°. Muriel, Anderson e Waldir meus sócios e amigos que me apoiaram neste projeto.

DEDICATÓRIA

A minha querida esposa que me apóia e sabe me compreender nos momentos mais difíceis de nossas vidas.

SUMÁRIO

1 INTRODUÇÃO

Este projeto surgiu da necessidade da empresa Tecnocontrol de monitorar um novo equipamento desenvolvido, e que precisa ter algumas grandezas físicas analisadas, tais como pressão, tensão elétrica, temperatura e umidade interna, abertura do painel, por um longo período de tempo, para poder conhecer o rendimento, o funcionamento e suas limitações.

Estas informações recolhidas em espaços de tempo pré-determinados serão analisadas e mostrarão o funcionamento do equipamento montado em campo e, com isso, será possível melhorar a eficiência dos equipamentos, tanto do ponto de vista de produção, quanto do ponto de vista da eficiência energética.

Os dados coletados são posteriormente lidos por um software capaz de gerar gráficos e relatórios e, com isso, demonstrar como o equipamento monitorado se comportou em um determinado espaço de tempo.

Este trabalho vem demonstrar de forma prática a construção deste coletor de dados, tornar seu custo menor sendo um produto tão confiável quanto qualquer coletor de dados encontrado no mercado.

Além disso, serão apresentados os meios de transmissão dos dados coletados remotamente por meio de um modem GPRS ou de um rádio modem ligado ao canal de comunicação serial do coletor de dados.

2 DESCRIÇÃO DO EQUIPAMENTO

2.1 CONCEPÇÃO DO SISTEMA DE COLETOR DE DADOS

O equipamento terá a função de coletar dados em campo e armazená-los para posterior leitura e verificação, para conhecer o real funcionamento do equipamento que terá suas grandezas físicas medidas.

O “cérebro” do equipamento que será responsável por controlar a leitura, a armazenagem e a comunicação de todos os blocos que compõem o coletor de dados é um microcontrolador PIC18F2520 da Microchip.

Para compreender o funcionamento e a construção deste coletor, sua estrutura será apresentada em forma de blocos e cada um dos blocos será estudado em detalhes no decorrer deste trabalho. Abaixo teremos uma breve descrição de cada um dos blocos que compõem o coletor de dados.

A Fig. 1 mostra o diagrama de blocos do coletor de dados.

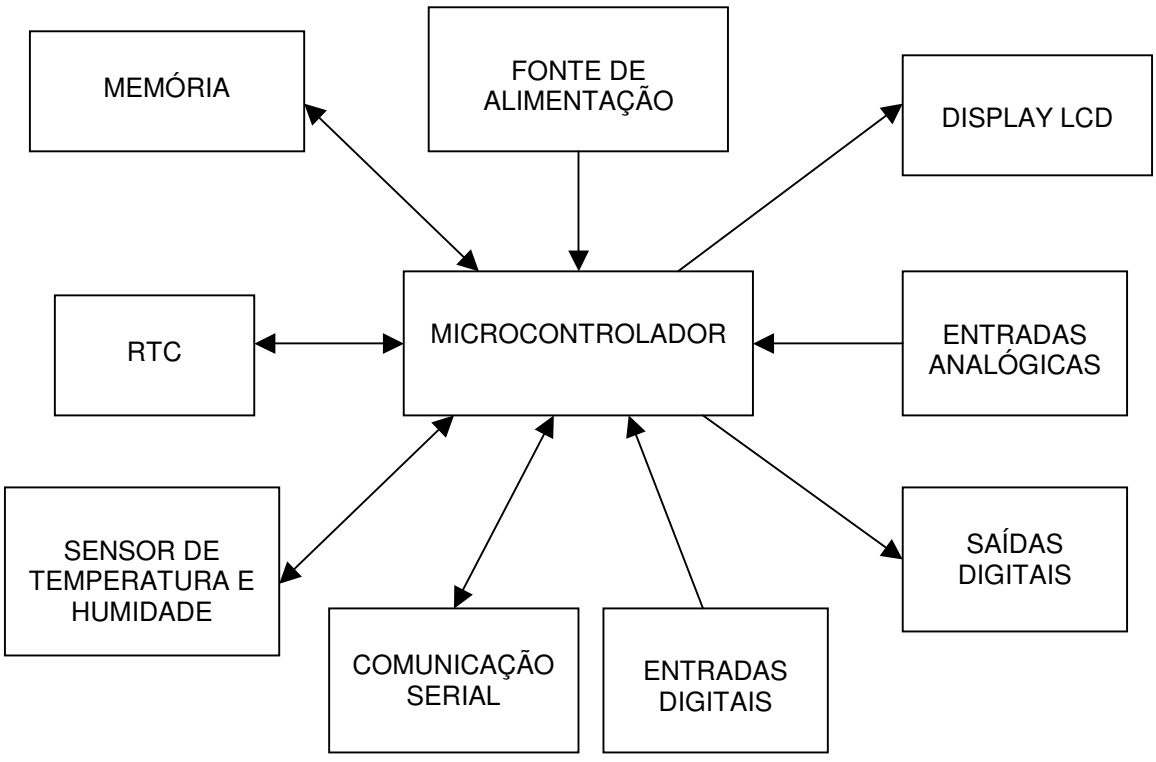


Fig. 1 – Diagrama de blocos do coletor de dados.

As entradas digitais têm a função de ler dados, por exemplo, ligado ou desligado, porta aberta ou fechada, válvula aberta ou fechada, isto é, qualquer sinal em forma digital 0 ou 1.

As entradas analógicas tem a função de ler sinais de tensão de 0 a 10V ou de corrente de 4 a 20mA, que podem representar grandezas físicas como tensão, pressão, vazão, nível, temperatura, umidade, etc.

Além de coletar os dados, será possível, através de duas saídas a relé, controlar algum equipamento. Esta função foi implementada para criar uma rotina que, caso ocorra alguma falha, permita programar o coletor de dados de maneira que, caso ocorra um aumento súbito em alguma variável que esteja sendo monitorada, o controlador possa atuar a fim de proteger o equipamento monitorado.

O sensor SHT11 da Sensirion é responsável pela leitura de temperatura e umidade do ambiente onde estará instalado o coletor de dados, assim as entradas analógicas ficam livres para monitorar outras grandezas.

O coletor possui ainda um relógio de tempo real (*Real-Time Clock – RTC*), o DS1307 da Maxim. Este fornece ao microcontrolador a data e a hora exata em que a medição de uma determinada grandeza foi realizada.

Os dados coletados serão armazenados em uma memória EEPROM, modelo 24AA1025 da Microchip, com capacidade de 128 kbytes.

Caso haja necessidade de monitorar alguma informação no local e o usuário que lerá essas informações não dispuser de um computador para descarregar os dados, foi previsto um display de cristal líquido para que essa monitoração local seja feita. Caso não haja necessidade da instalação do display, os pinos ficam livres para serem usados em outra função, como por exemplo, uma expansão através de uma comunicação com outra placa, com outras funcionalidades que não foram previstas na placa do coletor de dados.

A comunicação e troca de dados do coletor com um computador serão feitas via padrão EIA-232 e, para isso, foi utilizado o circuito integrado MAX 232 da Maxim, que converterá o sinal TTL gerado pelo canal serial do microcontrolador em um nível padrão de tensão para o computador. Estes dados poderão trafegar via cabo serial, modem GPRS e também via rádio modem, possibilitando com estes dois últimos meios a coleta de dados remotamente.

Com as funções descritas acima, serão cobertas todas as necessidades de monitoramento de sinais e grandezas físicas, bem como sua análise futura, criando assim um banco de dados sobre o funcionamento dos equipamentos em que o coletor de dados foi instalado.

2.2 ENTRADAS DIGITAIS

Quando houver a necessidade de saber se um determinado equipamento está ligado ou desligado, a única forma de medir isso é verificar se ele está ou não energizado. E no presente caso, esta é a informação de interesse.

O coletor de dados possui quatro entradas digitais isoladas opticamente. Como os sinais deverão ser lidos pelo microcontrolador (ele também é o responsável por controlar todo o funcionamento do resto do circuito) não são compatíveis com o nível de tensão lidos, é necessário utilizar componentes que façam esta interface e também protejam as entradas do microcontrolador. Para evitar problemas desta ordem, as entradas digitais são isoladas do resto do circuito a fim de evitar que qualquer surto de tensão danifique e comprometa todo o sistema.

Na Fig. 2 temos o circuito elétrico das entradas digitais que usam para isolar o sinal de entrada dos pinos do microcontrolador um circuito integrado optoacoplador, o 4N25. A saída do 4N25, através do pino 4, caso sua entrada seja acionada, envia para o microcontrolador um nível alto (5V).

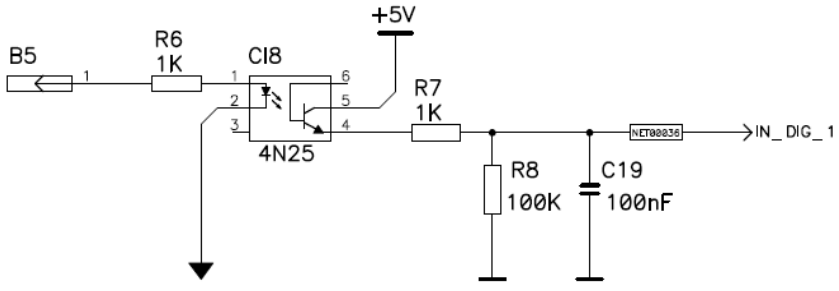


Fig. 2 – Circuito elétrico da entrada digital.

O resistor R7 tem a função de limitar a corrente do sinal e proteger a entrada do microcontrolador. Já o resistor de R8 tem a função de quando a entrada do 4N25 não estiver acionada, manter o sinal para o microcontrolador em nível baixo (0V). O

capacitor de 100nF é utilizado como filtro. Os quatro sinais são ligados em entradas do microcontrolador e o nível destes sinais é TTL.

Os pinos do microcontrolador usados para receber os sinais digitais são os pinos RA4, RA5, RC0 E RC1, identificados respectivamente como IN_DIG_1, IN_DIG_2, IN_DIG_3 e IN_DIG_4.

2.3 ENTRADAS ANALÓGICAS

Vivemos em um mundo analógico, todas as grandezas físicas tem seu sinal variável no tempo e como o coletor de dados trabalha em um meio digital é necessário que se convertam esses sinais.

O microcontrolador que será usado já possui internamente este conversor, o que facilita muito o desenvolvimento do circuito e a parte de controle do sistema.

Os sinais analógicos a serem medidos serão sinais padrão de tensão de 0 a 10V e de corrente de 4 a 20mA, que na verdade serão convertidos para sinais de tensão de 0 a 5V que é a tensão máxima que o microcontrolador poderá ler em seu canal analógico.

O coletor de dados possui quatro entradas analógicas que estão preparadas para ler tensão de 0 a 10V ou corrente de 4 a 20mA.

Na Fig. 3 é possível ver o circuito elétrico de uma das entradas analógicas.

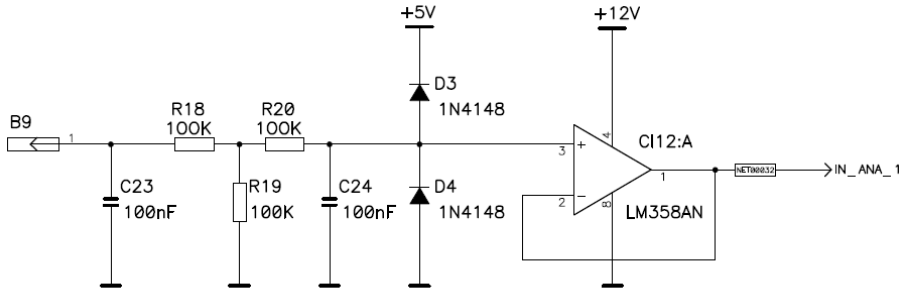


Fig. 3 - Circuito elétrico da entrada analógica.

Se a entrada for configurada para leitura de um sinal analógico de tensão de 0 a 10V, os resistores R18 e R19 se apresentarão como um divisor de tensão e, com isso, a máxima tensão medida será de 5V. Mas ser for necessário ler um sinal de corrente de 4 a 20mA, basta substituir R18 por um *jumper* e substituir R19 por um

resistor de 250 Ω com precisão de 1%, o sinal lido será uma tensão que variará de 1 a 5V, conforme a corrente de entrada.

O resistor R20 irá limitar a corrente de entrada do sinal, e os capacitores C23 e C24 tem a função de filtrar qualquer ruído indesejado que estiver presente ocorrendo na entrada do sinal analógico. Os diodos D3 e D4 tem a função de limitar a tensão de entra e proteger a entrada do amplificador operacional, desviando qualquer sinal superior a 5V e inferior a GND.

O amplificador operacional está montado na configuração de *buffer*, ou seja, com ganho unitário e sua função é de garantir um perfeito acoplamento das impedâncias do circuito de entrada do sinal analógico e o sinal de saída do amplificador operacional que será ligado diretamente ao canal analógico do microcontrolador.

Os pinos do microcontrolador usados para receber os sinais analógicos são os pinos RA0, RA1, RA2 E RA3, identificadas respectivamente como IN_ ANA_1, IN_ ANA_2, IN_ ANA_3 e IN_ ANA_4.

2.4 SAÍDAS A RELÉ

O coletor de dados tem a função de armazenar informações sobre o funcionamento de um determinado equipamento para posterior análise. Mas o que pode acontecer se as grandezas físicas medidas ultrapassarem algum limite que possa ocasionar dano ao equipamento que está sendo monitorado? Certamente um prejuízo muito grande. Pensando nisso, o coletor de dados aqui apresentado possui duas saídas a relé, onde no programa principal do microcontrolador pode ser criada uma rotina de segurança, que ao detectar alguma anomalia previamente programada, desligue o equipamento, protegendo-o contra danos maiores. Ou, como ele pode se comunicar remotamente, o que será visto mais adiante, alguma providência pode ser tomada a distância.

O relé utilizado possui um contato reversível, com isso é possível montar várias configurações de lógicas para proteção ou acionamento.

A Fig. 4 mostra o circuito utilizado para acionamento do relé.

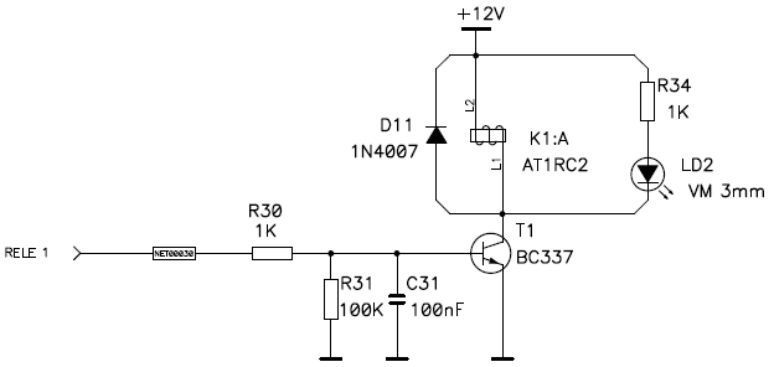


Fig. 4 - Circuito elétrico de acionamento do relé.

O funcionamento do circuito é bastante simples, o microcontrolador controla através de um transistor o acionamento do relé. O resistor R30 limita a corrente do sinal de acionamento, o resistor R31 garante nível baixo enquanto o microcontrolador mantém o sinal de saída desligado e o C31 evita que o transistor atue caso haja algum ruído na base do transistor T1. O diodo D11 é montado em paralelo com a bobina do relé e tem a função de proteger o circuito contra a sobretensão da bobina do relé. Em paralelo com a bobina também é acionado um LED que indica se o relé está ou não acionado.

Os pinos do microcontrolador usados para acionar os relés são os pinos RB6 E RB7, identificados respectivamente como RELE_1 e RELE_2.

2.5 MEDIÇÃO DE TEMPERATURA E UMIDADE

Um dos principais problemas encontrados em equipamentos que apresentam defeitos de funcionamento é a temperatura elevada no ambiente em que estes equipamentos estão colocados. O aumento de temperatura pode gerar condensação, e com isso aumentar a umidade dentro destes equipamentos. Pensando nessa situação o coletor de dados possui uma interface para medição exclusiva de temperatura e de umidade.

2.5.1 SENSOR SHT11

Para essa finalidade foi escolhido o sensor SHT11 da Sensirion, onde estão presentes no mesmo encapsulamento os dois sensores. Este componente possui, num único encapsulamento, um sensor de umidade relativa e um sensor de temperatura. Os sensores estão acoplados a um conversor A/D e a leitura dos valores pode ser feita através de um canal de comunicação serial dedicado. Isso resulta em uma ótima qualidade de sinal, uma resposta rápida e oferece uma grande imunidade às perturbações externas. A Fig. 5 mostra o sensor SHT11.

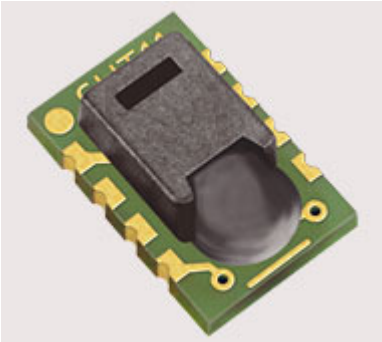


Fig. 5 – Sensor SHT11.

A faixa de leitura da temperatura é de –40 a 123,8°C com uma resolução de 0,01°C e a saída do sinal convertido pode ser de 12 ou 14 bits. Já a umidade relativa pode variar de 0 a 100% com uma resolução de 0,03 e a saída do sinal convertido pode ser de 8 ou 12 bits. A vantagem em se utilizar uma resolução menor do conversor A/D é a de diminuir o tempo entre a solicitação da leitura do sinal e a resposta do sensor que pode ser 11/55/210ms para 8/12/14 bits, e este tempo ainda pode variar 15% dependendo da freqüência do cristal interno.

2.5.2 ALIMENTAÇÃO

O SHT11 requer uma tensão de alimentação de 2,4 a 5V. Após alimentar o componente é necessário que se aguarde 11ms para que ele inicialize. Neste tempo não devemos enviar nenhum comando ao componente. Os pinos de alimentação são o 1 para GND e o 4 para VCC.

2.5.3 COMUNICAÇÃO SERIAL (2-WIRE)

A comunicação do SHT11 consiste em uma comunicação serial de somente duas linhas e é otimizado para um baixo consumo de energia.

A função do pino 3, o SCK, é sincronizar a comunicação entre um microcontrolador e o SHT11. Uma vez que é uma interface totalmente estática, não existe requisito de freqüência mínima de *clock*.

O pino 2 tem a função de ser o canal de comunicação de dados (*data channel*) que é bi-direcional. Ele envia sinais em nível baixo, e por isso é necessário colocar um resistor de 10kΩ para possibilitar nível alto nesse pino.

2.5.4 PROTOCOLO DE COMUNICAÇÃO

Para iniciar a comunicação entre o SHT11 e o microcontrolador é necessário um pulso de *start* que consiste em colocar o pino 2 (*data*) em nível zero por dois pulsos de *clock.* Após isso podemos enviar o comando para leitura da temperatura ou da umidade.

Na Fig. 6 temos um exemplo de como é o procedimento para a leitura de umidade.

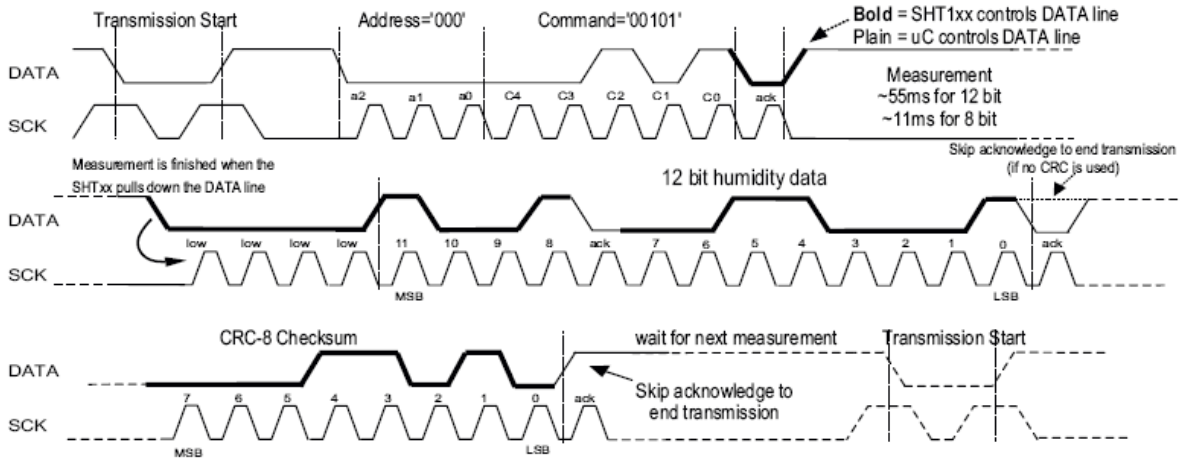


Fig. 6 – Procedimento para leitura de umidade no SHT11.

A Fig. 7 mostra de forma resumida os comandos de solicitação de leitura e recebimento do pacote com a resposta.

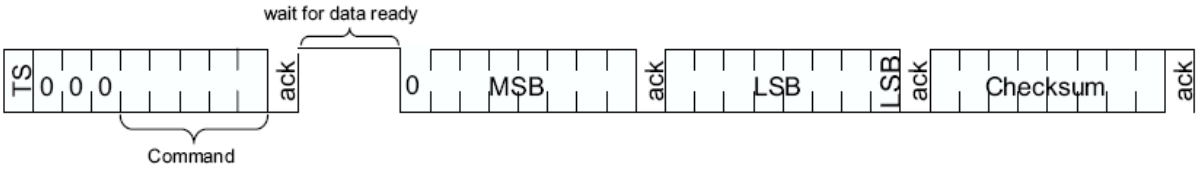


Fig. 7 – Solicitação e resposta de leitura.

Por padrão a leitura dos valores de temperatura e de umidade são de 14 e 12 bits, respectivamente. Mas podem ser alterados para 12 e 8 bits, respectivamente. Com isso é possível ter uma redução no tempo de leitura, que depende do número de bits que se deseja na leitura dos dados. Para ler este valor basta enviar um pulso de *start* e na seqüência o comando **00000110**, em seguida será recebido o valor do registrador e o *checksum*. Para alterar a resolução do conversor A/D para 8 bits na leitura do valor de umidade e 12 bits no valor da temperatura, basta enviar um pulso de *start* e na seqüência o comando **00000111 ack 00000001 ack**. A Fig. 8 mostra as seqüências de leitura e escrita no registrador.

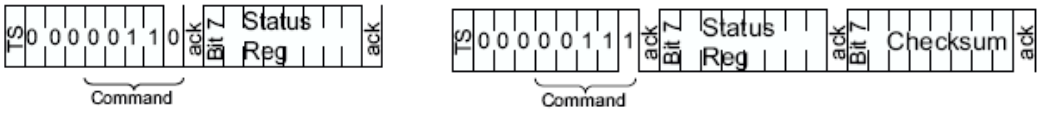


Fig. 8 – Comandos para leitura e escrita no registrador.

2.5.5 CIRCUITO ELÉTRICO

Foram usados dois pinos do microcontrolador para a interface com o sensor SHT11. O pino 2 do sensor, destinado a trafegar os dados, está ligado no pino RC4 do microcontrolador e o pino 3 do sensor destinado ao *clock,* está ligado ao pino RC2 do microcontrolador. O resistor R5 serve para manter o sinal da linha sempre em nível alto e o capacitor C11 de 100nF é o capacitor de desacoplamento do componente. Os demais pinos do sensor não são utilizados.

O circuito da Fig. 9 mostra a forma de ligação do sensor SHT11.

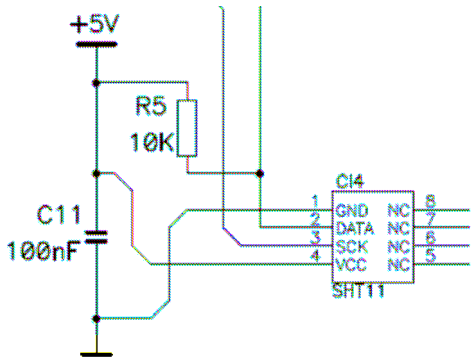


Fig. 9 – Circuito elétrico de ligação do SHT11.

2.6 RELÓGIO DE TEMPO REAL (*REAL-TIME CLOCK – RTC*)

Um relógio de tempo real (RTC ou *real-time clock*, em inglês) é um relógio computacional (geralmente sob a forma de um circuito integrado) que mantém o controle do tempo/calendário. Embora o termo freqüentemente refira-se a dispositivos em computadores pessoais, servidores e sistemas embarcados, os RTCs estão presentes na quase totalidade dos dispositivos eletrônicos que precisam manter um controle preciso do tempo.

Em inglês, a expressão "*real-time clock*" é utilizada para evitar confusão com um outro "*clock*" também utilizado em computadores. Este último é apenas um padrão de freqüência que controla os ciclos da eletrônica digital, e não é feito para contagem do tempo/calendário.

Embora o controle do tempo possa ser feito sem um RTC, usar um traz benefícios:

- Baixo consumo de energia;
- Libera o sistema principal para tarefas mais críticas;
- Costuma ser mais preciso do que outros métodos.

Os RTCs freqüentemente possuem uma fonte de energia alternativa, de forma que podem continuar a contagem do tempo enquanto a fonte de energia principal está desligada ou indisponível. Esta fonte alternativa é, normalmente, uma bateria.

A maioria dos RTCs usa um cristal de quartzo, mas alguns usam a freqüência da rede de alimentação elétrica . Em muitos casos, a freqüência do oscilador é de 32,768 kHz.

2.6.1 O DS1307

É de fundamental importância que todas as informações que serão coletadas e armazenadas na memória do coletor de dados indiquem a data e o horário em que foram realizadas as leituras.

Para esta função o coletor de dados possui um RTC (*Real Time Clock*), e os dados são transferidos via comunicação serial, usando o protocolo I^2C.

O modelo adotado, o DS1307 da *Dallas Semiconductor*, fornece informações dos segundos, minutos, horas, dia, mês e ano, no formato BCD e apresenta baixo consumo de energia. Possui capacidade de reconhecer os meses com menos de 31 dias e inclusive anos bissextos.

Possui ainda, um circuito sensor interno que detecta falha na alimentação principal e, automaticamente, passa a ser alimentado pela bateria de 3,5 V do tipo íon-lítio de 35 mAh. Essa bateria garante o funcionamento do RTC por até 10 anos na ausência da alimentação principal.

Seu funcionamento exige a conexão de um cristal de quartzo com freqüência de 32,768 kHz (XT2) e sua tensão de operação pode variar de 4,5 V a 5,5 V. A Fig. 10 mostra a forma de ligação do RTC.

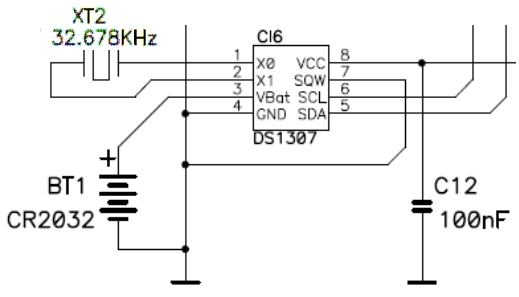


Fig. 10 - Circuito elétrico do DS1307.

O DS1307 possui o mesmo encapsulamento do microcontrolador, ou seja, tipo PDIP, porém com 8 pinos, sendo que a descrição de cada pino, juntamente com o significado da denominação utilizada, pode ser obtida na folha de dados do componente no Anexo I.

2.7 ARMAZENAMENTO DE DADOS

A memória é a capacidade de reter, recuperar, armazenar e evocar informações disponíveis, seja internamente, no cérebro (memória humana), seja externamente, em dispositivos artificiais (memória artificial).

Diante desta constatação podemos afirmar que uma das partes mais importantes do coletor de dados é a armazenagem dos dados lidos em campo. A memória que irá guardar os dados deve ser grande o suficiente para poder armazenar uma quantidade razoável de dados, viabilizando um diagnóstico mais apurado do equipamento que se quer monitorar.

Memórias podem ser do tipo volátil ou não volátil. Uma memória não volátil é a que retém a informação armazenada quando a energia elétrica é desligada. Memória volátil é aquela que perde a informação armazenada quando a energia elétrica desaparece (interrupção de alimentação elétrica ou desligamento da chave ON/OFF do equipamento). Devido ao fato do coletor de dados ser um equipamento que deverá ler e armazenar dados é usada uma memória não volátil.

Por ser compatível com o padrão de comunicação disponível no microcontrolador e possuir uma capacidade grande de armazenagem, foi escolhida a memória 24AA1025 da Microchip que é capaz de armazenar 128K bytes de dados, suficiente para armazenar informações por 40 dias.

A memória 24AA1025 se comunica pelo padrão I^2C, e em função do RTC também se comunicar por este barramento, temos uma padronização na comunicação destes dispositivos, o que facilita muito a programação do microcontrolador.

A ligação elétrica da memória 24AA1025 pode ser vista na Fig. 11.

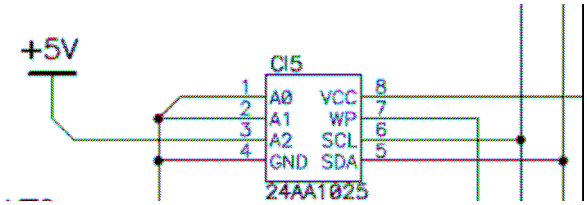


Fig. 11 – Circuito elétrico da memória 24AA1025.

2.8 COMUNICAÇÃO SERIAL

Um canal de comunicação é um caminho pelo qual a informação pode trafegar. Ela pode ser definida por uma linha física (fio) que conecta dispositivos de comunicação.

Em comunicação digital, a informação é representada por bits de dados individuais, que podem ser agrupados em mensagens de vários bits. Um byte (conjunto de 8 bits) é um exemplo de uma unidade de mensagem que pode trafegar através de um canal digital de comunicações. Uma coleção de bytes pode ser agrupada em um “frame” ou outra unidade de mensagem de maior nível. Esses múltiplos níveis de agrupamento facilitam o reconhecimento de mensagens e interconexões de dados complexos.

A maioria das mensagens digitais são mais longas que alguns poucos bits. Por não ser prático nem econômico transferir todos os bits de uma mensagem simultaneamente, a mensagem é dividida em partes menores e transmitida seqüencialmente. A transmissão bit-serial converte a mensagem em um bit por vez através de um canal. Cada bit representa uma parte da mensagem. Os bits individuais são então rearranjados no destino para compor a mensagem original. Em geral por um canal irá passar apenas um bit por vez. A transmissão bit-serial é normalmente chamada de transmissão serial, e é o método de comunicação escolhido por diversos periféricos.

O padrão utilizado é o RS-232, comumente usado nas portas seriais dos computadores pessoais, (também conhecido por EIA RS-232C ou V.24), que é um padrão para troca serial de dados binários entre um DTE (terminal de dados, de *Data Terminal Equipment*) e um DCE (comunicador de dados, de *Data Communication Equipment*).

No protocolo de comunicação RS-232, caracteres são enviados um a um como um conjunto de bits. A codificação mais comumente usada é o "*start-stop* assíncrono" que usa um bit de início, seguido por sete ou oito bits de dados, possivelmente um bit de paridade, e um ou dois bits de parada sendo, então, necessários 10 bits para enviar um único caractere. Tal fato acarreta a necessidade em dividir por um fator de dez a taxa de transmissão para obter a velocidade de transmissão. O padrão define os níveis elétricos correspondentes aos níveis lógicos um e zero, a velocidade de transmissão padrão e os tipos de conectores.

O RS-232 é recomendado para conexões curtas (quinze metros ou menos). Os sinais variam de 3 a 15 volts positivos ou negativos, sendo que valores próximos de zero não são sinais válidos. O nível lógico um é definido por ser voltagem negativa, a condição de sinal é chamada marca e tem significado funcional de OFF (desligado). O nível lógico zero é positivo, a condição de sinal é espaço, e a função é ON (ligado).

Como o microcontrolador possui um canal de comunicação serial, adotaremos este padrão para troca de dados entre o coletor de dados e o computador. O canal de comunicação serial do microcontrolador usa padrão TTL e para acoplar o sinal do microcontrolador ao computador utilizaremos o componente MAX232 (Maxim). Ele inclui um circuito de "*charge pump*" capaz de gerar tensões de +10V e –10V a partir de uma fonte de alimentação simples de +5V, bastando para isso alguns capacitores externos. Este componente também tem 2 *receivers* e 2 *drivers* no mesmo encapsulamento. Nos casos onde serão implementados somente as linhas de transmissão e de recepção de dados, não seria necessário dois circuitos integrados e fontes de alimentação extras.

O coletor de dados utiliza somente os pinos de RX, TX e GND para a comunicação, o que torna a programação do microcontrolador mais simples, sem a necessidade de controle do fluxo de dados. A velocidade de comunicação será de 9600bps, 8 bits de dados, 1 bit de parada e nenhuma paridade.

A Fig. 12 mostra a ligação elétrica do componente MAX 232.

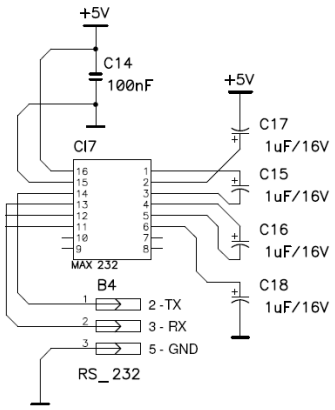


Fig. 12 – Circuito elétrico do MAX232.

2.8.1 MEIOS DE TRANSMISSÃO E COLETA DOS DADOS

Por se tratar de um equipamento que possui comunicação serial, o meio de transmissão dos dados e coleta pelo computador pode ser por cabo serial, modem GPRS e Rádio Modem. Este meio vai depender da necessidade de coleta de dados on-line, das distâncias envolvidas e da disponibilidade de sinal para esta troca remota de dados.

2.8.2 COMUNICAÇÃO VIA GPRS

O GPRS (*General Packet Radio Service*) é um novo serviço de valor agregado não baseado em voz que permite o envio e recepção de informações através de uma rede telefônica móvel. Ele suplementa as tecnologias atuais de CSD (*Circuit Switched Data*) e SMS (*Short Message Service*).

Taxas de transferência teóricas de até 171,2 kbps são possíveis com GPRS usando todos os oito *timeslots* ao mesmo tempo. Isso é uma taxa de transferência três vezes mais rápida do que as possíveis nas redes de telecomunicações fixas e dez vezes mais que os atuais serviços de CSD nas redes GSM.

O GPRS facilita conexões instantâneas, pois a informação pode ser enviada ou recebida imediatamente conforme a necessidade do usuário. Não há necessidade de conexões *dial-up* através de *modems*. Algumas vezes, diz-se que os

usuários de GPRS estão “sempre conectados”. Disponibilidade imediata é uma das vantagens de GPRS quando comparado com CSD. Alta disponibilidade imediata é uma característica muito importante para aplicações críticas como autorização remota de lançamento em cartões de crédito, quando é inaceitável que o cliente seja mantido em estado de espera por mais de 30 segundos além do necessário.

O GPRS facilita muitas novas aplicações não disponíveis através das redes GSM, dadas as limitações na taxa de transferência dos CSDs (9,6 kbps) e do tamanho da mensagem no SMS (160 caracteres). As aplicações com este meio de troca de dados vão desde navegação na rede mundial de computadores até transferência de arquivos para automação de residências - a habilidade de acessar e controlar remotamente os equipamentos e recursos disponíveis em uma casa.

Para usar GPRS para transmitir os dados de nossa aplicação é necessário:

- Um telefone móvel ou terminal que suporte GPRS;
- Uma assinatura em uma rede de telefonia móvel que suporte GPRS;
- Ter o uso de GPRS habilitado. Acesso automático ao GPRS pode ser permitido por algumas operadoras; outras poderão requerer uma opção específica de adesão;
- Conhecimento de como enviar e receber informações através do GPRS usando seu aparelho telefônico, incluindo configurações de hardware e software, o que cria a necessidade de um serviço de atendimento ao cliente;
- Um destino para enviar ou um local de onde receber informações através do GPRS.

Enquanto que com SMS esse destino ou origem era, freqüentemente, outro telefone móvel, com GPRS é mais provável que se pareça com um endereço Internet, já que GPRS foi projetado para tornar o acesso à internet totalmente disponível aos usuários móveis desde o início. Desde a disponibilidade do serviço, os usuários do GPRS podem acessar qualquer página da rede mundial de computadores ou outras aplicações internet - fornecendo uma massa crítica inicial de uso.

Em alguns casos a coleta de dados instantânea pode ser necessária, e a utilização de um modem GPRS pode ser viável e muito prática, permitindo com isso o monitoramento *on-line* das informações.

Uma das desvantagens em usar este meio de troca de dados é o custo mensal cobrado pela operadora de telefonia móvel. Mas este custo não inviabiliza sua utilização.

A Fig. 13 mostra um modem GPRS da Motorola.

Fig. 13 – Modem GPRS.

2.8.3 COMUNICAÇÃO VIA RÁDIO MODEM

É uma opção cara do ponto de vista da instalação, mas é um ótimo investimento em longo prazo.

Uma das vantagens de se usar este tipo de meio de comunicação é de que seu custo de operação é praticamente zero, diferente do GPRS que é pago, uma vez que o sistema de rádio é propriedade do cliente.

Já a maior desvantagem de se usar a comunicação com rádio é que a comunicação é serial ponto-a-ponto, ou seja, em se tratando de vários pontos de leitura, a comunicação é feita em forma de varredura, um ponto por vez. Se ocorrer um defeito em um rádio em campo, apenas aquela estação ficará sem leitura; se o defeito for no equipamento de rádio da central, todas as estações ligadas a este equipamento ficam sem comunicação.

Hoje em dia são necessárias licenças para operar com rádio modem, mas existe um sistema com potência até 1W que é livre destas licenças, chamada de *Spread Spectrum.*

A tecnologia *Spread Spectrum* usa uma técnica de codificação para transmissão digital onde, ao invés de transmitir o sinal continuamente sobre uma banda de freqüência estreita, várias partes são transmitidas separadamente através de um amplo espectro de freqüência.

Neste caso, o alcance fica limitado pela potência, mas se o sistema a ser monitorado ficar a uma distância com visada entre os dois pontos a serem

conectados, o fabricante garante uma distância de visada de até 90km, neste caso rádios com esta tecnologia são interessantes.

A Fig. 14 mostra um rádio modem com tecnologia *Spread Spectrum* da empresa Freewave.

Fig. 14 – Rádio *Spread Spectrum* da Freewave.

2.9 FONTE DE ALIMENTAÇÃO

Todo dispositivo eletroeletrônico necessita de uma fonte de alimentação para seu funcionamento.

A fonte de alimentação do coletor de dados tem por base uma fonte clássica com regulação linear, composta por capacitores de filtro, capacitores de desacoplamento da fonte e dois circuitos integrados reguladores de tensão com proteção incorporada contra sobrecorrente e sobretemperatura para cada uma das tensões necessárias ao funcionamento dos componentes da placa do coletor de dados.

Conforme o circuito elétrico da Fig. 15, o diodo D1 tem a função de proteger a entrada da fonte contra a entrada de tensão inversa. O LED LD1 é utilizado para indicar visualmente que o circuito de alimentação está funcionando perfeitamente.

Esta fonte alimenta com segurança os relés e demais componentes que são utilizados na placa do coletor de dados.

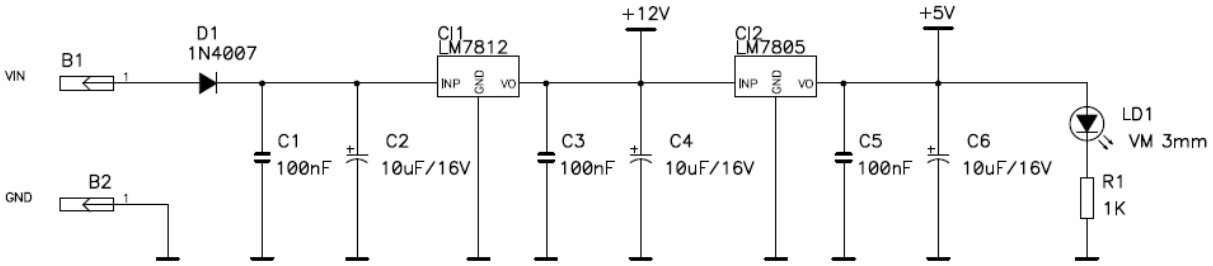


Fig. 15 – Circuito elétrico da fonte de alimentação.

2.10 INTERFACE HOMEM-MÁQUINA

Apesar de ser um coletor de dados, onde os dados serão transferidos para um computador para posterior análise, será utilizada uma comunicação do aparelho com o usuário através de um display de cristal líquido.

Isso se faz necessário caso se deseje monitorar alguma variável instantaneamente. Facilitando o acompanhamento de algumas grandezas físicas a fim de saber se o equipamento está dentro da normalidade estabelecida para seu funcionamento.

O mostrador utilizado foi o LCD (*Liquid Crystal Display*). Um display de cristal líquido alfanumérico de 2 linhas por 16 colunas, tendo por base o módulo controlador HD44780, da Hitachi, mostrado na Fig. 16.

Fig. 16 – Display de cristal líquido (LCD).

Esse tipo de LCD possui 16 pinos e pode ser conectado ao microcontrolador através de um barramento de dados de 4 ou 8 bits, sendo que no presente projeto o

mesmo foi configurado para operar no modo de 4 bits. Sua tensão de alimentação pode variar de 4,5 a 5,5V. Por meio do LCD é possível obter algumas informações do aparelho.

2.10.1 DISPLAY DE CRISTAL LÍQUIDO LCD

O display de LCD é largamente utilizado em diversos aparelhos eletrônicos com a finalidade de mostrar resultados preliminares ou informações que auxiliem no manejo do aparelho.

Para colocá-lo em funcionamento, primeiro é necessário configurá-lo, ou seja, programar o display para transferir os dados para ele (8 ou 4 bits), quantas linhas vamos utilizar, se a mensagem deve ficar fixa ou rolar, se a escrita será da esquerda para direita ou da direita para esquerda, ou seja, todas essas configurações são necessárias antes de escrever qualquer mensagem. A folha de dados do display LCD traz essas informações mais detalhadas.

A pinagem do display LCD 16x2 segue abaixo:

- Pinos de dados: D7 -...- D0 (8bits) - os pinos de dados são usados para enviar as palavras de configurações e os dados (caracteres);
- Pinos de controle: EN (6), RS (4), R/W (5) - o pino EN informa ao display de LCD quando o dado está pronto para ser lido. O pino RS é usado para diferenciar se a palavra que foi enviada ao LCD é de configuração ou caractere;
- Pinos de alimentação: Vcc (2) e GND (1);
- Pino de controle de contraste: VO (3) - este pino permite alterar o contraste do display;
- Pinos de iluminação do fundo - *backlight*: A (16), K (15) - nem todos os displays possuem iluminação de fundo.

A Fig. 17 mostra a ligação elétrica do display de cristal líquido.

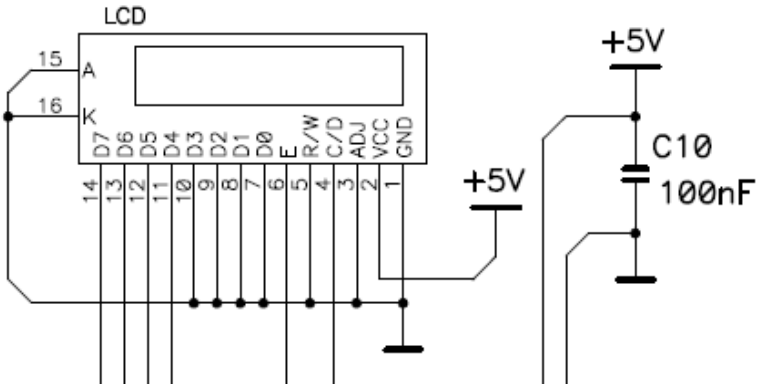


Fig. 17 – Circuito elétrico do display de cristal líquido (LCD).

2.11 PLACA DE CIRCUITO IMPRESSO

A placa de circuito impresso é a responsável por agrupar todos os componentes e fazer as ligações entre eles.

Para desenhar o circuito elétrico foi utilizado o software PCAD 2004 Schematic e para o desenho da placa de circuito Impresso foi utilizado o software PCAD 2004 PCB ambos da empresa Altium.

Por se tratar de um circuito que deve ter um tamanho reduzido, a placa de circuito impresso foi projetada em face dupla. Para melhorar o desempenho e a estabilidade do funcionamento dos circuitos, foi feita na parte inferior da placa uma malha de terra que cobre toda a área não utilizada da placa de circuito impresso.

O esquema elétrico e o desenho da placa de circuito impresso podem ser vistos nos Anexos 8 e 9, respectivamente.

2.12 MICROCONTROLADOR

Há uma grande variedade de microcontroladores comercialmente disponíveis na atualidade. Não é feita aqui qualquer comparação entre os diversos tipos de um mesmo fabricante, e nem mesmo entre os vários fabricantes. Antes, contudo, é focado naquele considerado adequado para o presente projeto, levando em conta os seguintes aspectos:

- Baixo custo;

- Baixo consumo de energia;
- Fácil aquisição;
- Possibilidade de utilização de uma linguagem de programação de alto nível e fácil implementação.

Optou-se pelo microcontrolador PIC18F2520, fabricado pela empresa americana *Microchip Technology*, que possui representação no Brasil. Abaixo segue algumas das principais características deste microcontrolador.

2.12.1 CARACTERÍSTICAS DA CPU E MEMÓRIAS

A CPU e as memórias possuem as seguintes características:

- CPU RISC com arquitetura Harvard;
- Conjunto de apenas 77 instruções;
- Instruções executadas em um único ciclo de *clock*, exceto para instrução de desvio;
- Otimizado para programação em linguagem de alto nível;
- Velocidade de operação de até 40 MHz (ou 100 ns por instrução);
- Memória de programa tipo *Flash* de 32K com palavras de 14 bits e suporta 100.000 ciclos de escrever/apagar;
- Suporta programação *in-circuit;*
- Suporta *debug;*
- Memória de dados tipo RAM de 1536 *bytes*;
- Memória de dados tipo EEPROM de 256 *bytes* e suporta 1.000.000 de ciclos de escrever/apagar;
- Retêm dados por período superior a 100 anos.

2.12.2 PERIFÉRICOS EMBUTIDOS

- 25 pinos de entrada ou saída;
- Alta corrente entre dreno/fonte 25 mA/25 mA;
- Três interrupções externas programáveis;
- Quatro entradas para interrupções para mudança de estado;

- Dois módulos de Capture/Compare/PWM (CCP);
- Módulo MSSP suportando SPI™ ou I²C™;
- Módulo USART;
- Dez entradas analógicas de 10-bit;
- Dois canais de comparadores analógicos;
- Um *timer* de 8 bit;
- Dois *timers* de 16 bit.

2.12.3 CARACTERÍSTICAS ESPECIAIS

Adicionalmente, o microcontrolador possui as seguintes características:

- Código de proteção programável;
- Modo de economia de energia;
- Baixo consumo de energia;
- Larga faixa de tensão de operação (2,0 V a 5,5V);
- Várias opções de osciladores;
- *Reset* programável por *Brown-out;*
- W*atchdog timer.*

A Fig. 18 mostra a pinagem do PIC18F2520.

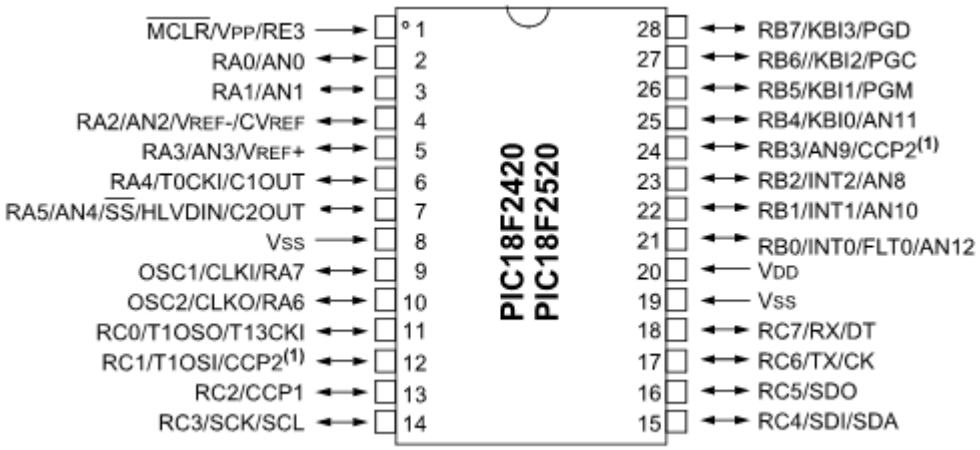


Fig. 18 – Pinagem do PIC18F2520.

O PIC18F2520 é fornecido em diversos tipos de encapsulamentos, sendo que no presente projeto foi escolhido o encapsulamento PDIP (*Plastic Dual In-line Package*) de 28 pinos, devido à maior facilidade de montagem, tanto em uma matriz de contato, para o desenvolvimento do protótipo, quanto em placa de circuito impresso, para a versão final.

2.13 SOFTWARE DE PROGRAMAÇÃO

O MPLAB é um programa que tem a função de gerenciador, para o desenvolvimento de projetos com a família PIC de microcontroladores. É distribuído gratuitamente pela Microchip, fabricante dos PICs.

O MPLAB integra num único ambiente o editor de programa fonte, o compilador, o simulador e quando conectado às ferramentas da Microchip também integra o gravador do PIC e o emulador.

O Programa fonte, ou simplesmente fonte do programa é uma seqüência em texto, escrita numa linguagem de programação que será convertida em códigos de máquina para ser gravado no PIC. Todo o código desenvolvido para o coletor de dados será em linguagem de alto nível, a linguagem C.

O Compilador é o programa que converte o código fonte em códigos de máquina. O Simulador é o programa que simula o funcionamento da CPU (PIC), conforme o programa fonte que está sendo desenvolvido.

O Projeto no MPLAB é um conjunto de arquivos e informações que informa ao ambiente integrado qual o PIC que está sendo usado, qual a freqüência de *clock*, qual a linguagem de programação usada, qual o layout das janelas, etc.

É importante lembrar que o MPLAB se integra ao ambiente Windows, permitindo cópia de arquivos, de textos de um aplicativo para outro de uma forma bem simplificada.

Utilizaremos neste projeto a versão MPLAB IDE 8.0 e para compilar os comandos da linguagem C utilizaremos o compilador *PCWH Compiler* da CCS versão 4.04.

A Fig. 19 mostra um Diagrama de Blocos Simplificado do *firmware* para o Microcontrolador.

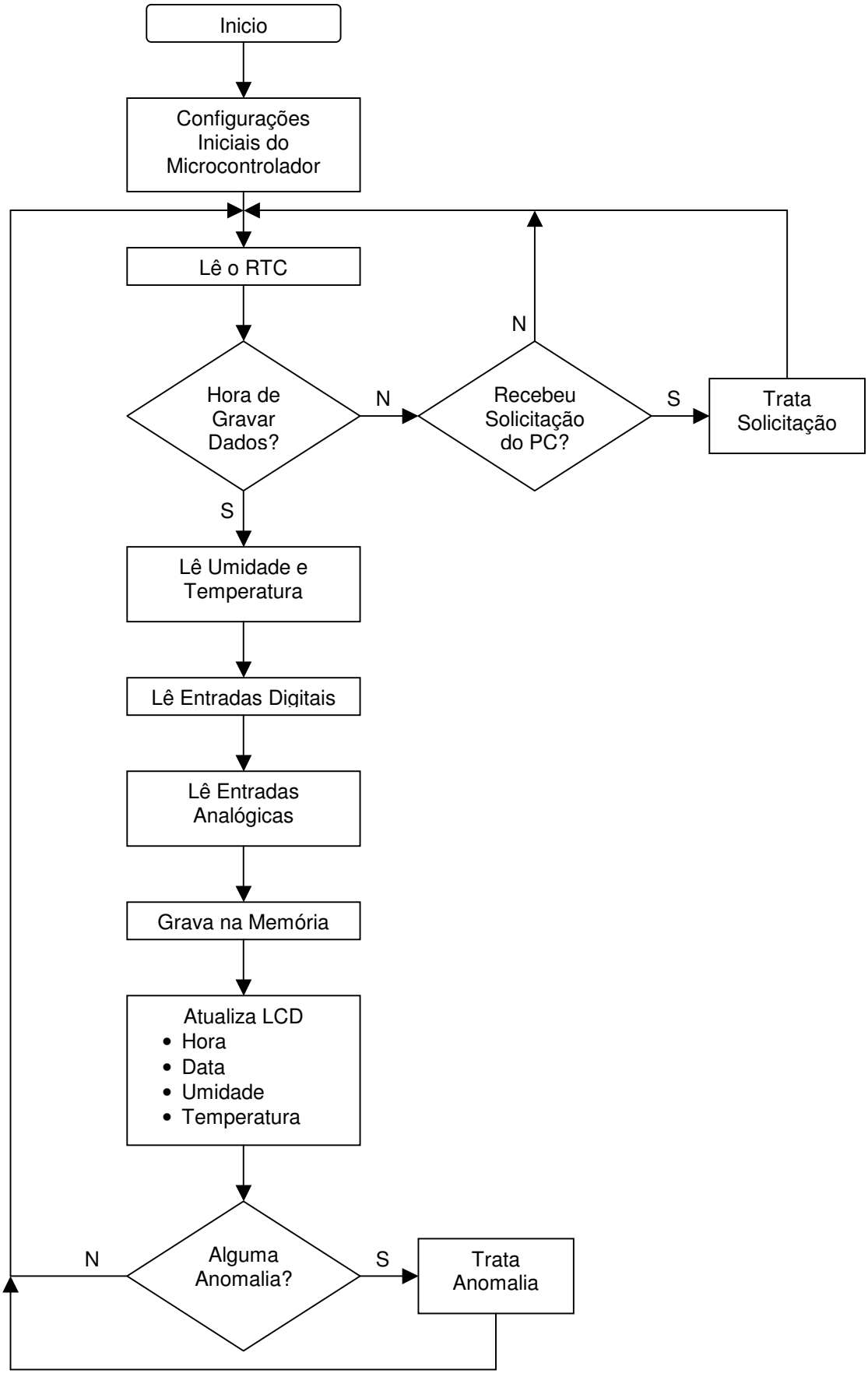


Fig. 19 – Diagrama de blocos do f*irmware* do microcontrolador.

3 APLICAÇÃO PRÁTICA

A necessidade de desenvolver um coletor de dados onde fosse possível monitorar várias grandezas físicas e estados lógicos foi apresentada pela empresa Tecnocontrol Tecnologia e Sistemas Ltda, para que fosse possível monitorar ar o funcionamento de um equipamento.

Este equipamento é usado para pressurizar água tratada e aumentar a área coberta pela distribuição pelas empresas de saneamento, e é chamado de *Booster®*.

A Fig. 20 mostra um *Booster®* da empresa Tecnocontrol.

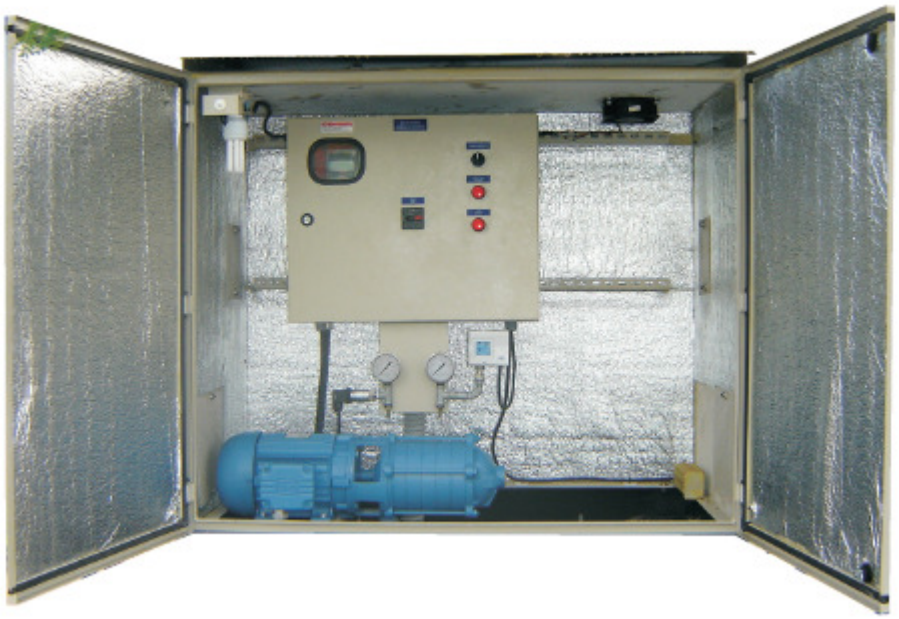


Fig. 20 – Foto de um *Booster®* da empresa Tecnocontrol.

Este pressurizador de água é composto por um motor elétrico, uma bomba hidráulica e um painel que controla através da pressão de saída de água a rotação do motor, que por sua vez aumenta e diminui a rotação da bomba, mantendo constante a pressão de saída e assim abastecendo todas as casas ligadas ao longo da rede de distribuição. Também são controladas umidade e temperatura dentro do gabinete.

Por se tratar de um produto novo do qual ainda não havia um conhecimento sobre seu real funcionamento, foi necessário desenvolver um equipamento que

monitorasse algumas variáveis essenciais e mostrasse o real funcionamento deste equipamento, o *Booster®.*

A proposta de criar um coletor de dados surgiu, pois os coletores encontrados no mercado além de caros, não tinham todas as funcionalidades que seriam necessárias para monitorar o funcionamento do *Booster®.*

Nesta etapa o coletor de dados monitora sinais digitais que representam se há ou não água na entrada da bomba, se a temperatura ultrapassou um valor pré-determinado no termostato de segurança, se houve abertura da porta no período de monitoramento e se houve alguma falha no painel elétrico. Já os sinais analógicos monitoram a umidade relativa e temperatura dentro do gabinete.

Neste momento estes dados são suficientes para conhecer o funcionamento deste pressurizador e saber se ele está operando dentro dos limites estabelecidos ou se é necessário alguma alteração de projeto ou até mesmo de construção do gabinete.

Devido a fase de adaptação tanto do pressurizador quanto do coletor de dados, ainda não foi desenvolvido um software para computador capaz de receber os dados e tratar de forma autônoma estas informações, gerando gráficos e mostrando de forma mais intuitiva os dados coletados ao operador.

Atualmente os dados são mostrados através do Bloco de Notas do Windows® e importados para o Excel®, o que torna um pouco trabalhosa a análise dos dados. A Fig. 21 mostra o estado de duas entradas digitais e a Fig. 22 mostra os valores lidos por quatro entradas analógicas.

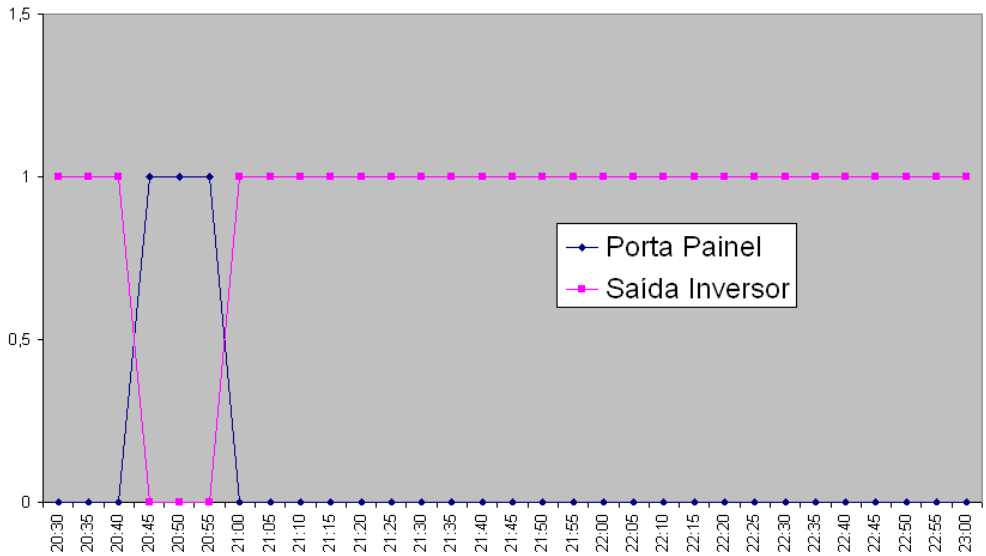


Fig. 21 – Gráfico com o estado de duas entradas digitais.

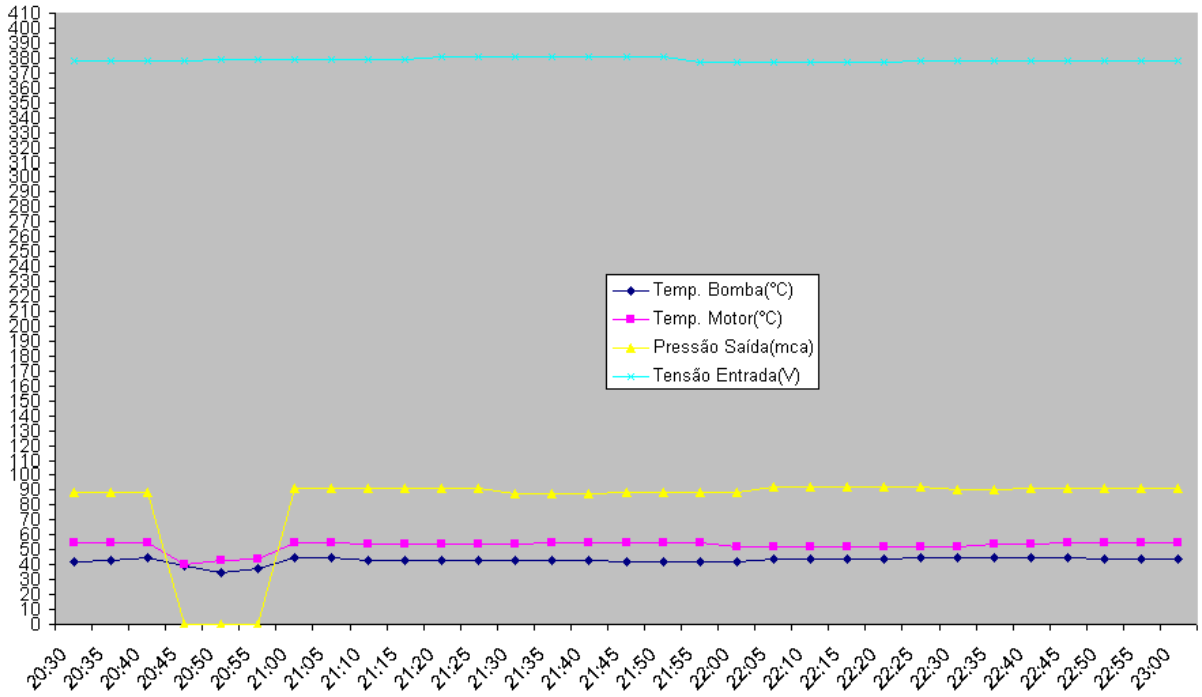


Fig. 22 – Gráfico com o estado de quatro portas analógicas.

4 CONCLUSÃO

Ao fazer a análise e projetar um equipamento tudo parece um tanto quanto simples, mas desenvolver um equipamento que procura trazer uma inovação e ser um referencial nem sempre é tarefa fácil.

Foi o caso do coletor de dados, que concebido inicialmente com o propósito de ser um equipamento barato e eficiente, conseguiu atingir este objetivo.

Foi possível mostrar ao longo deste trabalho as funcionalidades e facilidades para desenvolver um coletor de dados de baixo custo, pois o coletor de dados tem um custo de R$ 500,00, enquanto os disponíveis no mercado custam acima de R$ 1.500,00. É claro que se produzido em escala comercial a tendência é ficar ainda mais barato. Do ponto de vista financeiro o coletor de dados alcançou o objetivo.

O desenvolvimento da placa e sua montagem foram simples e com isso facilitaram o desenvolvimento do restante do projeto.

A programação do microcontrolador para ler os dados digitais e analógicos, a função criada para escrever no display LCD, a comunicação com o RTC, com a memória de dados e o canal de comunicação serial foram muito simples. Já a função para comunicação com o sensor de temperatura e umidade e o microcontrolador, por ser um protocolo proprietário, geraram mais de trabalho.

Outro ponto importante foi à criação do protocolo de comunicação do microcontrolador com o computador, que particularmente dispendeu um trabalho considerável, pois foi necessário tratar o dado recebido e enviado pelo microcontrolador a fim de garantir a integridade da informação.

A função de escrita e leitura da memória EEPROM também é de difícil implementação, afinal foi necessário fazer o controle e escrever em bancos diferentes alguns dados, conforme a memória estava sendo preenchida.

Mas, no geral, todo o hardware se comportou como o esperado e proporcionou um funcionamento aceitável para o coletor de dados.

O microcontrolador utilizado também se mostrou bastante eficiente tanto do ponto de vista de velocidade quanto de confiabilidade, não apresentando nenhum problema até o momento.

É claro que, com o aperfeiçoamento do coletor de dados e das necessidades futuras de ampliação, serão necessárias algumas melhorias e até mesmo a utilização de um microcontrolador com maior capacidade de processamento.

Em relação ao projeto final do coletor de dados, é possível afirmar que ele alcançou a todos os objetivos esperados, tanto do ponto de vista de funcionamento quanto do ponto de vista financeiro.

Quanto ao funcionamento foi possível monitorar os principais dados do pressurizador e saber como estava se comportando. Com isso foi possível mudar algumas coisa e melhorá-lo e, assim, possibilitar um melhor funcionamento.

Um ponto importante a ressaltar foi à utilização de uma função do inversor de freqüência utilizado pelo pressurizador que se chama DORMIR. Esta função desliga o motor e com isso há uma economia de energia elétrica quando o consumo é muito baixo ou até mesmo inexistente, que é o que ocorre durante as madrugadas.

Foi possível também confrontar informações com as empresas que compram os produtos, pois em alguns casos havia um aquecimento excessivo na bomba, o que ocasionavam problemas de funcionamento. Ao comparar alguns dados coletados foi possível comprovar a falta de água na entrada da bomba e, quando isso acontecia, havia um aquecimento além dos valores determinados de funcionamento. A empresa acabou melhorando a chegada da água até o pressurizador e com isso foi possível diminuir os problemas causados por falta de água na entrada da bomba.

Essas mudanças só foram possíveis depois que foram coletados e analisados os dados de funcionamento do pressurizador e com isso foi possível melhorar a eficiência e o consumo de energia elétrica dos equipamentos instalados.

Houve ainda uma situação onde foi possível comunicar remotamente o coletor de dados com um computador e assim possibilitando a troca instantânea de dados através de um Rádio Modem da Freewave. Neste caso, o pressurizador estava instalado próximo a empresa Tecnocontrol e isso facilitou essa comunicação, que por sinal se mostrou uma forma eficaz de coleta de dados e também demonstra o que foi proposto sobre as possibilidades de comunicação do coletor de dados com o computador.

Algumas melhorias previstas para serem adaptadas ao coletor são a mudança de uma plataforma de 8 bits para uma plataforma de 32 bits, o que agilizaria o processamento e também melhoraria o desempenho do coletor de

dados, afinal a tendência é que este equipamento possa ser utilizado para outras finalidades e coletar dados cada vez maiores e mais precisos e seria possível a implementação de novos periféricos.

Outro fator importante que deve ser melhorado é o hardware para gravação dos dados. Como a tendência é de coletar cada vez mais dados e com valores cada vez mais precisos, a memória utilizada hoje se tornará muito pequena. Uma alternativa para isso é a implementação de cartões de memória do tipo SDCard ou até mesmo uma entrada de comunicação USB para a conexão com um PenDrive. Isso tornaria muito mais flexível à troca e armazenagem dos dados.

Um sistema de energia auxiliar esta sendo estudado para que na falta de energia elétrica, exista a gravação de pelo menos uma leitura completa dos dados para poder comprovar e saber que de fato o problema foi a falta de energia da concessionária e não um defeito no equipamento instalado.

Para comunicação e troca de dados instantâneos, uma prática comum nos dias de hoje e a utilização da comunicação via GPRS. Este também é um ponto que está sendo analisado para ser implementado em versões futuras do coletor de dados e com isso melhorar seu desempenho.

Quanto ao software que deverá ser usado para comunicar o coletor de dados com o computador, que não foi abordado neste trabalho, será feito através de uma interface amigável e que mostre de forma fácil os dados e gráficos gerados a partir das informações obtidas em campo e com isso possibilite o acompanhamento do funcionamento dos equipamentos monitorados pelo coletor de dados. Essa é uma tarefa que deverá ser implementada em breve.

Ao final deste trabalho é possível mostrar as vantagens e desvantagens em desenvolver um equipamento para coleta de dados remotamente, afinal é um projeto simples a primeira vista, mas no decorrer do projeto as dificuldades vão aparecendo e devem ser vencidas, pois se trata de um equipamento que vem aliar tecnologia e melhorar o funcionamento de outros equipamentos.

Por fim este projeto demonstrou em várias esferas todos os conhecimentos adquiridos durante o curso de pós-graduação *Lato Sensu* em Desenvolvimento de Produtos Eletrônicos.

5 BIBLIOGRAFIA

PEREIRA, F. **Microcontroladores PIC:** Técnicas Avançadas. 1.ed. São Paulo: Érica, 2002. 358 p.

PEREIRA, F. **Microcontroladores PIC:** Programação em C. 2.ed. São Paulo: Érica, 2003. 358 p.

PEREIRA, F. **Microcontroladores MSP430:** Teoria e Prática. 1.ed. São Paulo: Érica, 2005. 414 p.

PEREIRA, F. **Tecnologia ARM:** Microcontrolador de 32 Bits. 1.ed. São Paulo: Érica, 2007. 448 p.

SOUZA, D. J. **Desbravando o PIC:** Ampliado e Revisado para PIC16F628A. 6.ed. São Paulo: Érica, 2003. 268 p.

SOUZA, D. J.; LAVINIA N. C. **Conectando o PIC:** Recursos Avançados. 1.ed. São Paulo: Érica, 2003. 380 p.

MALVINO, A. P. **ELETRÔNICA :** Volume 1. 1.ed. São Paulo: McGraw-Hill, 1987. 520 p.

SCHILDT, H **C Completo e Total**. São Paulo: McGraw-Hill, 1990.

CIPELLI, A. M. V.; SANDRINI W. J.; MARKUS O. Teoria e Desenvolvimento de Projetos de Circuitos Eletrônicos. 23.ed. São Paulo: Érica, 2007. 464 p.

Sites:

http://www.microchip.com - Fabricante de Microcontroladores PIC, da memória EEPROM 24AA1025 e diversos outros semicondutores

http://www.maxim-ic.com - Fabricante do MAX232, do RTC DS1307 e diversos outros semicondutores

http://www.sensirion.com - Fabricante do Sensor de Umidade e de Temperatura SHT11

http://www.fairchildsemi.com - Fabricante dos Reguladores de Tensão, do LM358 e diversos outros semicondutores

http://pt.wikipedia.org/wiki/Coletor_de_dados - conceito e definições sobre coletores de dados

http://www.rogercom.com - site com vários artigos e projetos na área de eletrônica

http://www.camiresearch.com/Data_Com_Basics/RS232_standard.html - site com especificações do padrão RS232.

http://www.informattechnology.com.br - Representante e Distribuidor do Modem GPRS da Motorola no Brasil

http://www.teleco.com.br - Site com informações sobre Telecomunicações

http://www.freewave.com - Fabricante de Rádio Modem

http://www.tecnocontrol.ind.br - Fabricante do *Booster®.*

ANEXOS

Anexo 1 – Dados principais do RTC DS1307

DALLAS SEMICONDUCTOR

DS1307
64 x 8 Serial Real-Time Clock

www.maxim-ic.com

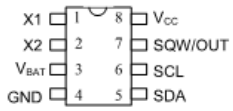
FEATURES

- Real-time clock (RTC) counts seconds, minutes, hours, date of the month, month, day of the week, and year with leap-year compensation valid up to 2100
- 56-byte, battery-backed, nonvolatile (NV) RAM for data storage
- Two-wire serial interface
- Programmable squarewave output signal
- Automatic power-fail detect and switch circuitry
- Consumes less than 500nA in battery backup mode with oscillator running
- Optional industrial temperature range: -40°C to +85°C
- Available in 8-pin DIP or SOIC
- Underwriters Laboratory (UL) recognized

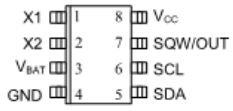
ORDERING INFORMATION

DS1307	8-Pin DIP (300-mil)
DS1307Z	8-Pin SOIC (150-mil)
DS1307N	8-Pin DIP (Industrial)
DS1307ZN	8-Pin SOIC (Industrial)

PIN ASSIGNMENT



DS1307 8-Pin DIP (300-mil)



DS1307 8-Pin SOIC (150-mil)

PIN DESCRIPTION

V_{CC}	- Primary Power Supply
X1, X2	- 32.768kHz Crystal Connection
V_{BAT}	- +3V Battery Input
GND	- Ground
SDA	- Serial Data
SCL	- Serial Clock
SQW/OUT	- Square Wave/Output Driver

DESCRIPTION

The DS1307 Serial Real-Time Clock is a low-power, full binary-coded decimal (BCD) clock/calendar plus 56 bytes of NV SRAM. Address and data are transferred serially via a 2-wire, bi-directional bus. The clock/calendar provides seconds, minutes, hours, day, date, month, and year information. The end of the month date is automatically adjusted for months with fewer than 31 days, including corrections for leap year. The clock operates in either the 24-hour or 12-hour format with AM/PM indicator. The DS1307 has a built-in power sense circuit that detects power failures and automatically switches to the battery supply.

Anexo 2 – Dados principais da memória 24AA1025

MICROCHIP

24AA1025/24LC1025/24FC1025

1024K $I^2C^{™}$ CMOS Serial EEPROM

Device Selection Table:

Part Number	Vcc Range	Max Clock Frequency	Temp Ranges
24AA1025	1.8-5.5V	400 kHz†	I
24LC1025	2.5-5.5V	400 kHz	I
24FC1025	2.5-5.5V	1 MHz	I

†100 kHz for Vcc < 2.5V.

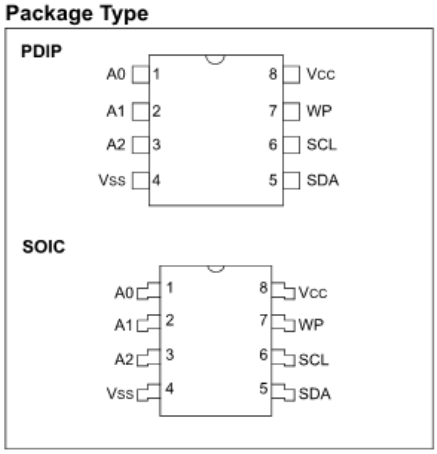
Features:

- Low-power CMOS technology:
  - Maximum write current 5 mA at 5.5V
  - Maximum read current 500 µA at 5.5V
  - Standby current 100 nA, typical at 5.5V
- 2-wire serial interface bus, $I^2C^{™}$ compatible
- Cascadable for up to four devices
- Self-timed erase/write cycle
- 128-byte Page Write mode available
- 5 ms max. write cycle time
- Hardware write-protect for entire array
- Output slope control to eliminate ground bounce
- Schmitt Trigger inputs for noise suppression
- 1,000,000 erase/write cycles
- Electrostatic discharge protection > 4000V
- Data retention > 200 years
- 8-pin PDIP, SOIC packages
- Temperature ranges:
  - Industrial (I): -40°C to +85°C

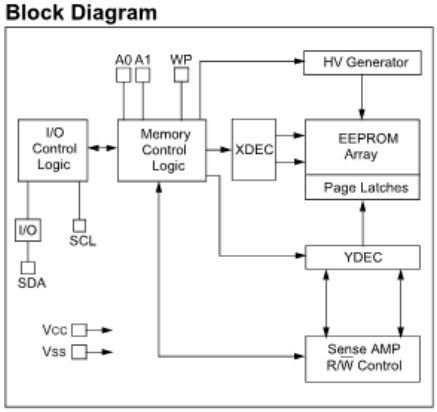
Description:

The Microchip Technology Inc. 24AA1025/24LC1025/24FC1025 (24XX1025*) is a 128K x 8 (1024K bit) Serial Electrically Erasable PROM, capable of operation across a broad voltage range (1.8V to 5.5V). It has been developed for advanced, low power applications such as personal communications or data acquisition. This device has both byte write and page write capability of up to 128 bytes of data. This device is capable of both random and sequential reads. Reads may be sequential within address boundaries 0000h to FFFFh and 10000h to 1FFFFh. Functional address lines allow up to four devices on the same data bus. This allows for up to 4 Mbits total system EEPROM memory. This device is available in the standard 8-pin plastic DIP and SOIC packages.

Package Type



Block Diagram



*24XX1025 is used in this document as a generic part number for the 24AA1025/24LC1025/24FC1025 devices.

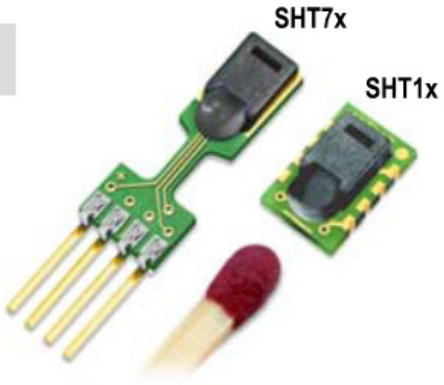
Anexo 3 – Dados principais do sensor SHT11

SENSIRION
THE SENSOR COMPANY

SHT1x / SHT7x

Humidity & Temperature Sensor

Evaluation Kit Available



- _ Relative humidity and temperature sensors
- _ Dew point
- _ Fully calibrated, digital output
- _ Excellent long-term stability
- _ No external components required
- _ Ultra low power consumption
- _ Surface mountable or 4-pin fully interchangeable
- _ Small size
- Automatic power down

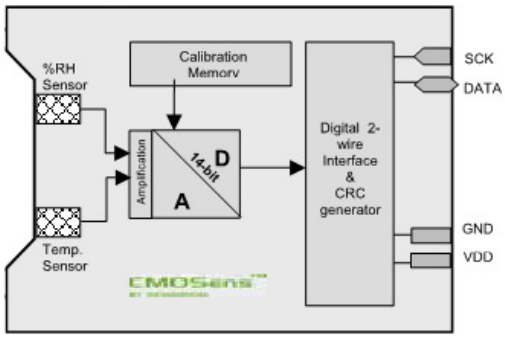
SHT1x / SHT7x Product Summary

The SHTxx is a single chip relative humidity and temperature multi sensor module comprising a calibrated digital output. Application of industrial CMOS processes with patented micro-machining (CMOSens® technology) ensures highest reliability and excellent long term stability. The device includes a capacitive polymer sensing element for relative humidity and a bandgap temperature sensor. Both are seamlessly coupled to a 14bit analog to digital converter and a serial interface circuit on the same chip. This results in superior signal quality, a fast response time and insensitivity to external disturbances (EMC) at a very competitive price.
Each SHTxx is individually calibrated in a precision humidity chamber with a chilled mirror hygrometer as reference. The calibration coefficients are programmed into the OTP memory. These coefficients are used internally during measurements to calibrate the signals from the sensors.
The 2-wire serial interface and internal voltage regulation allows easy and fast system integration. Its tiny size and low power consumption makes it the ultimate choice for even the most demanding applications.
The device is supplied in either a surface-mountable LCC (Leadless Chip Carrier) or as a pluggable 4-pin single-in-line type package. Customer specific packaging options may be available on request.

Applications

- _ HVAC
- _ Automotive
- _ Consumer Goods
- _ Weather Stations
- _ (De-) Humidifiers
- _ Test & Measurement
- _ Data Logging
- _ Automation
- _ White Goods
- _ Medical

Block Diagram



Ordering Information

Part Number	Humidity accuracy	Temperature accuracy	Package
SHT11	±3.5%RH	±0.5°C @25°C	SMT (LCC)
SHT15	±2.0%RH	±0.4°C	SMT (LCC)
SHT71	±3.5%RH	±0.5°C @25°C	4-pin single-in-line
SHT75	±2.0%RH	±0.4°C	4-pin single-in-line

Anexo 4 – Dados principais do conversor TTL/RS232 MAX232

19-4323; Rev 11; 2/03

MAXIM

+5V-Powered, Multichannel RS-232 Drivers/Receivers

General Description

The MAX220–MAX249 family of line drivers/receivers is intended for all EIA/TIA-232E and V.28/V.24 communications interfaces, particularly applications where ±12V is not available.

These parts are especially useful in battery-powered systems, since their low-power shutdown mode reduces power dissipation to less than 5µW. The MAX225, MAX233, MAX235, and MAX245/MAX246/MAX247 use no external components and are recommended for applications where printed circuit board space is critical.

Applications

Portable Computers

Low-Power Modems

Interface Translation

Battery-Powered RS-232 Systems

Multidrop RS-232 Networks

Features

Superior to Bipolar

- **Operate from Single +5V Power Supply (+5V and +12V—MAX231/MAX239)**
- **Low-Power Receive Mode in Shutdown (MAX223/MAX242)**
- **Meet All EIA/TIA-232E and V.28 Specifications**
- **Multiple Drivers and Receivers**
- **3-State Driver and Receiver Outputs**
- **Open-Line Detection (MAX243)**

Ordering Information

PART	TEMP RANGE	PIN-PACKAGE
MAX220CPE	0°C to +70°C	16 Plastic DIP
MAX220CSE	0°C to +70°C	16 Narrow SO
MAX220CWE	0°C to +70°C	16 Wide SO
MAX220C/D	0°C to +70°C	Dice*
MAX220EPE	-40°C to +85°C	16 Plastic DIP
MAX220ESE	-40°C to +85°C	16 Narrow SO
MAX220EWE	-40°C to +85°C	16 Wide SO
MAX220EJE	-40°C to +85°C	16 CERDIP
MAX220MJE	-55°C to +125°C	16 CERDIP

Ordering Information continued at end of data sheet.

**Contact factory for dice specifications.*

Selection Table

Part Number	Power Supply (V)	No. of RS-232 Drivers/Rx	No. of Ext. Caps	Nominal Cap. Value (µF)	SHDN & Three-State	Rx Active in SHDN	Data Rate (kbps)	Features
MAX220	+5	2/2	4	0.1	No	—	120	Ultra-low-power, industry-standard pinout
MAX222	+5	2/2	4	0.1	Yes	—	200	Low-power shutdown
MAX223 (MAX213)	+5	4/5	4	1.0 (0.1)	Yes	✔	120	MAX241 and receivers active in shutdown
MAX225	+5	5/5	0	—	Yes	✔	120	Available in SO
MAX230 (MAX200)	+5	5/0	4	1.0 (0.1)	Yes	—	120	5 drivers with shutdown
MAX231 (MAX201)	+5 and +7.5 to +13.2	2/2	2	1.0 (0.1)	No	—	120	Standard +5/+12V or battery supplies; same functions as MAX232
MAX232 (MAX202)	+5	2/2	4	1.0 (0.1)	No	—	120 (64)	Industry standard
MAX232A	+5	2/2	4	0.1	No	—	200	Higher slew rate, small caps
MAX233 (MAX203)	+5	2/2	0	—	No	—	120	No external caps
MAX233A	+5	2/2	0	—	No	—	200	No external caps, high slew rate
MAX234 (MAX204)	+5	4/0	4	1.0 (0.1)	No	—	120	Replaces 1488
MAX235 (MAX205)	+5	5/5	0	—	Yes	—	120	No external caps
MAX236 (MAX206)	+5	4/3	4	1.0 (0.1)	Yes	—	120	Shutdown, three state
MAX237 (MAX207)	+5	5/3	4	1.0 (0.1)	No	—	120	Complements IBM PC serial port
MAX238 (MAX208)	+5	4/4	4	1.0 (0.1)	No	—	120	Replaces 1488 and 1489
MAX239 (MAX209)	+5 and +7.5 to +13.2	3/5	2	1.0 (0.1)	No	—	120	Standard +5/+12V or battery supplies; single-package solution for IBM PC serial port
MAX240	+5	5/5	4	1.0	Yes	—	120	DIP or flatpack package
MAX241 (MAX211)	+5	4/5	4	1.0 (0.1)	Yes	—	120	Complete IBM PC serial port
MAX242	+5	2/2	4	0.1	Yes	✔	200	Separate shutdown and enable
MAX243	+5	2/2	4	0.1	No	—	200	Open-line detection simplifies cabling
MAX244	+5	8/10	4	1.0	No	—	120	High slew rate
MAX245	+5	8/10	0	—	Yes	✔	120	High slew rate, int. caps, two shutdown modes
MAX246	+5	8/10	0	—	Yes	✔	120	High slew rate, int. caps, three shutdown modes
MAX247	+5	8/9	0	—	Yes	✔	120	High slew rate, int. caps, nine operating modes
MAX248	+5	8/8	4	1.0	Yes	✔	120	High slew rate, selective half-chip enables
MAX249	+5	6/10	4	1.0	Yes	✔	120	Available in quad flatpack package

For pricing, delivery, and ordering information, please contact Maxim/Dallas Direct! at 1-888-629-4642, or visit Maxim's website at www.maxim-ic.com.

Anexo 5 – Dados principais do Microcontrolador PIC18F2520

MICROCHIP **PIC18F2420/2520/4420/4520**

28/40/44-Pin Enhanced Flash Microcontrollers with 10-Bit A/D and nanoWatt Technology

Power Managed Modes:

- Run: CPU on, peripherals on
- Idle: CPU off, peripherals on
- Sleep: CPU off, peripherals off
- Idle mode currents down to 5.8 µA typical
- Sleep mode current down to 0.1 µA typical
- Timer1 Oscillator: 1.8 µA, 32 kHz, 2V
- Watchdog Timer: 2.1 µA
- Two-Speed Oscillator Start-up

Peripheral Highlights:

- High-current sink/source 25 mA/25 mA
- Three programmable external interrupts
- Four input change interrupts
- Up to 2 Capture/Compare/PWM (CCP) modules, one with Auto-Shutdown (28-pin devices)
- Enhanced Capture/Compare/PWM (ECCP) module (40/44-pin devices only):
  - One, two or four PWM outputs
  - Selectable polarity
  - Programmable dead time
  - Auto-Shutdown and Auto-Restart
- Master Synchronous Serial Port (MSSP) module supporting 3-wire SPI™ (all 4 modes) and I^2C™ Master and Slave Modes
- Enhanced Addressable USART module:
  - Supports RS-485, RS-232 and LIN 1.2
  - RS-232 operation using internal oscillator block (no external crystal required)
  - Auto-Wake-up on Start bit
  - Auto-Baud Detect
- 10-bit, up to 13-channel Analog-to-Digital Converter module (A/D):
  - Auto-acquisition capability
  - Conversion available during Sleep
- Dual analog comparators with input multiplexing)

Flexible Oscillator Structure:

- Four Crystal modes, up to 40 MHz
- 4X Phase Lock Loop (available for crystal and internal oscillators)
- Two External RC modes, up to 4 MHz
- Two External Clock modes, up to 40 MHz
- Internal oscillator block:
  - 8 user selectable frequencies, from 31 kHz to 8 MHz
  - Provides a complete range of clock speeds from 31 kHz to 32 MHz when used with PLL
  - User tunable to compensate for frequency drift
- Secondary oscillator using Timer1 @ 32 kHz
- Fail-Safe Clock Monitor:
  - Allows for safe shutdown if peripheral clock stops

Special Microcontroller Features:

- C compiler optimized architecture:
  - Optional extended instruction set designed to optimize re-entrant code
- 100,000 erase/write cycle Enhanced Flash program memory typical
- 1,000,000 erase/write cycle Data EEPROM memory typical
- Flash/Data EEPROM Retention: 100 years typical
- Self-programmable under software control
- Priority levels for interrupts
- 8 x 8 Single-Cycle Hardware Multiplier
- Extended Watchdog Timer (WDT):
  - Programmable period from 4 ms to 131s
- Single-supply 5V In-Circuit Serial Programming™ (ICSP™) via two pins
- In-Circuit Debug (ICD) via two pins
- Wide operating voltage range: 2.0V to 5.5V
- Programmable 16-level High/Low-Voltage Detection (HLVD) module:
  - Supports interrupt on High/Low-Voltage Detection
- Programmable Brown-out Reset (BOR
  - With software enable option

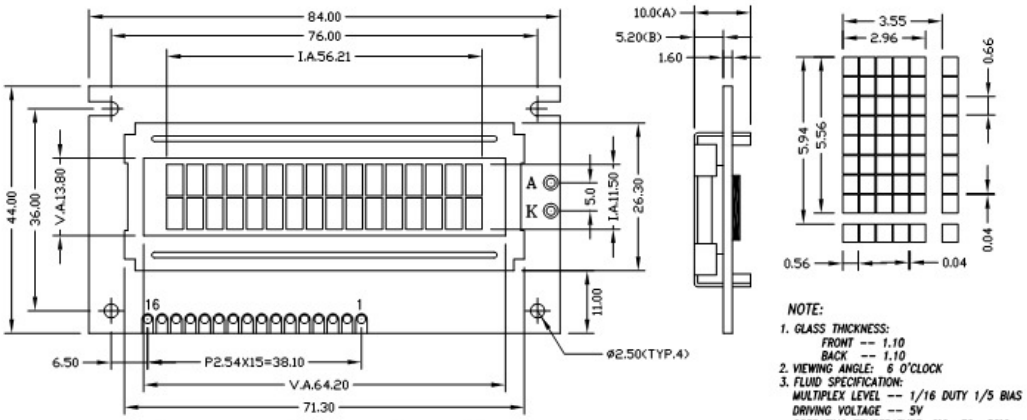
Anexo 6 – Dados principais do Display de Cristal Líquido

LCD GROUP

ITM-1602BYTBL

INTECH LCD (SHEN ZHEN) CO.,LTD			
VER	A0	FILE NO	IT-III-ED-004
REV.NO	01	SHEET	OF
DATE	24.FEB.2000		

① EXTERNAL DIMENSION



	A	B
LED B/L	13.40mm	9.0mm
WITHOUT B/L	10.00mm	5.2mm

NOTE:
1. GLASS THICKNESS:
   FRONT -- 1.10
   BACK -- 1.10
2. VIEWING ANGLE: 6 O'CLOCK
3. FLUID SPECIFICATION:
   MULTIPLEX LEVEL -- 1/16 DUTY 1/5 BIAS
   DRIVING VOLTAGE -- 5V
   OPERATING TEMPERATURE: 0°C TO 50°C
   STORAGE TEMPERATURE: -20°C TO 70°C
4. CONNECTOR: ZEBRA
5. STN YELLOW-GREEN MODE
6. TRANSFLECTIVE

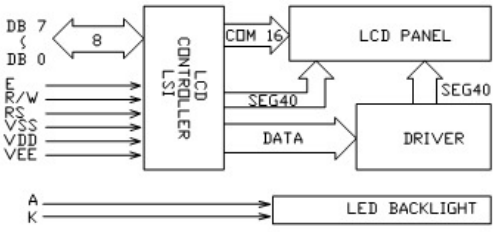
② MECHANICAL DATA

ITEM	SPECFICATION	UNIT
Module Size (W X H X T)	84.0 X 44.0 X 10.0	mm
Viewing Area (W X H)	64.2X 13.8	mm
Internal Area (W X H)	56.21 X 11.50	mm
Dot Pitch (W X H)	0.70 X 0.60	mm
Dot Size (W X H)	0.66 X 0.56	mm
Character Pith (WXH)	5.94 X 3.55	mm
Character Size (W X H)	5.56 X 2.96	mm
Number of Character	16 Character X 2 Line	Char

③ PIN CONFIGURATION

ITEM	SYMBOL	LEVEL	DESCRIPTION
1	A	H	LED BACKLIGHT +
2	K	L	LED BACKLIGHT -
3	VSS	L	Power GND
4	VDD	H	Power Positive
5	VEE	L	Voltage For LCD Panel Drive
6	RS	H/L	Register Select
7	R/W	H/L	Read/Write Select
8	E	H/L	Read/Write Enable
9	DB0	H/L	
10	DB1	H/L	
11	DB2	H/L	
12	DB3	H/L	Data Bus
13	DB4	H/L	
14	DB5	H/L	
15	DB6	H/L	
16	DB7	H/L	

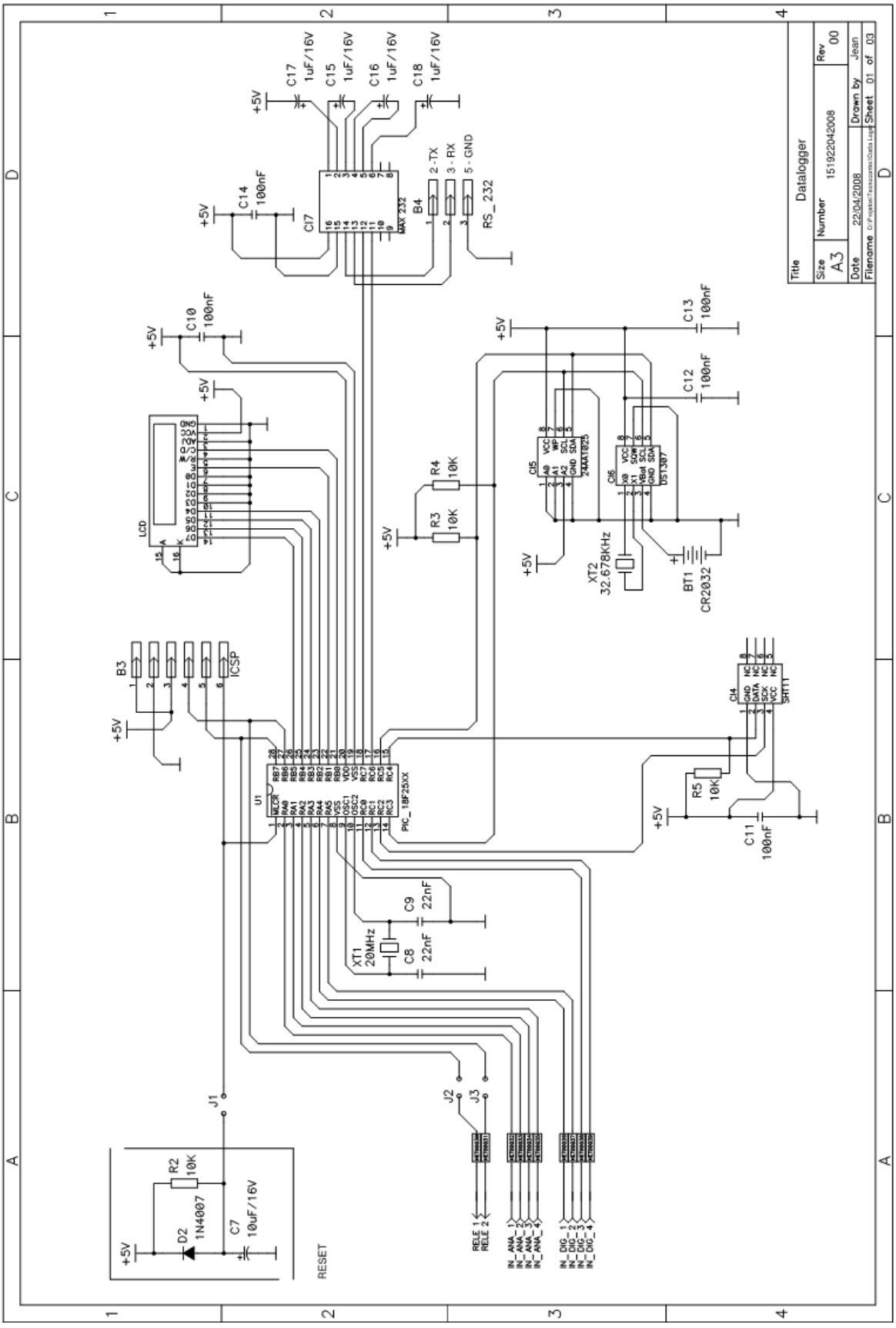
④ BLOCK DIAGRAM

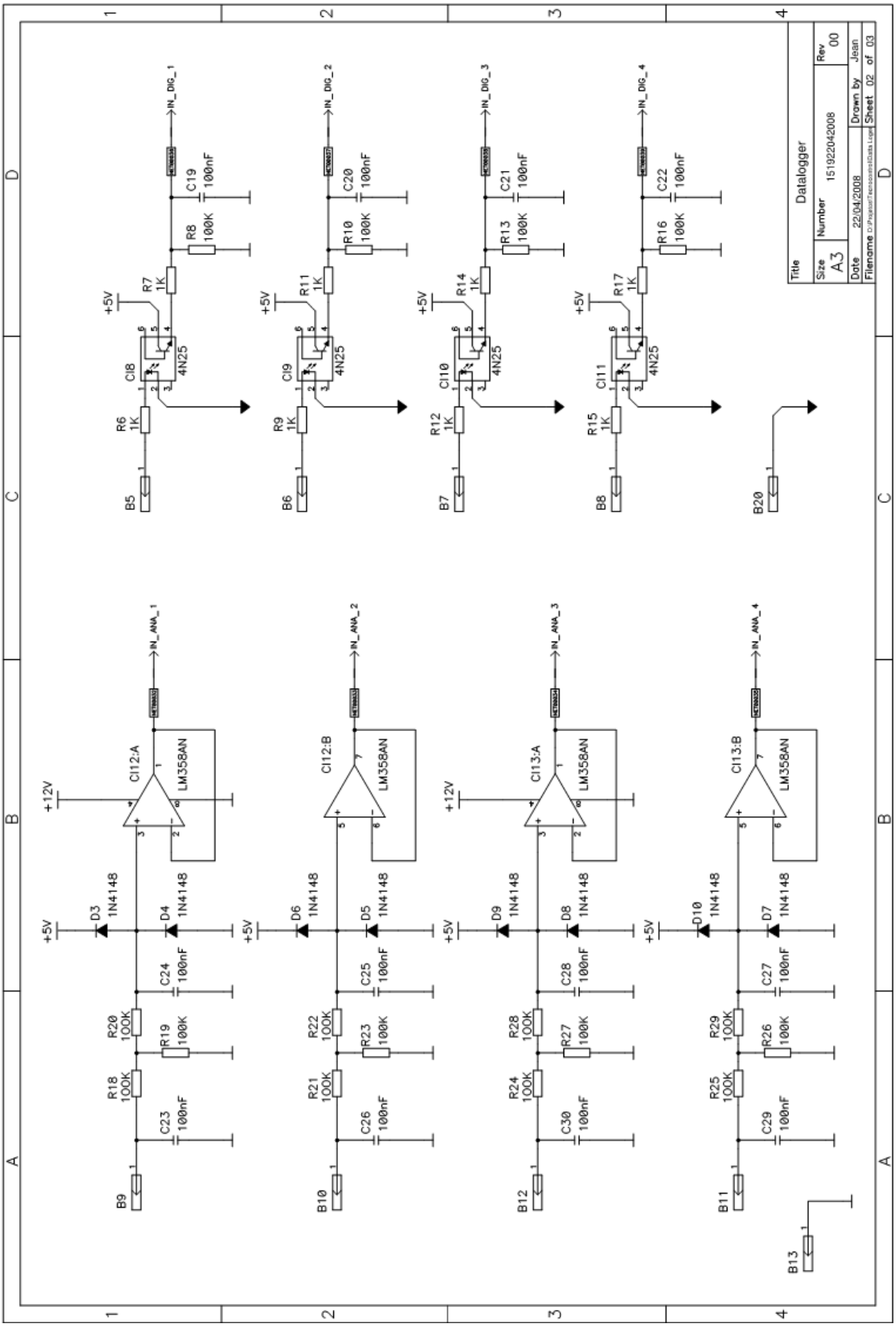


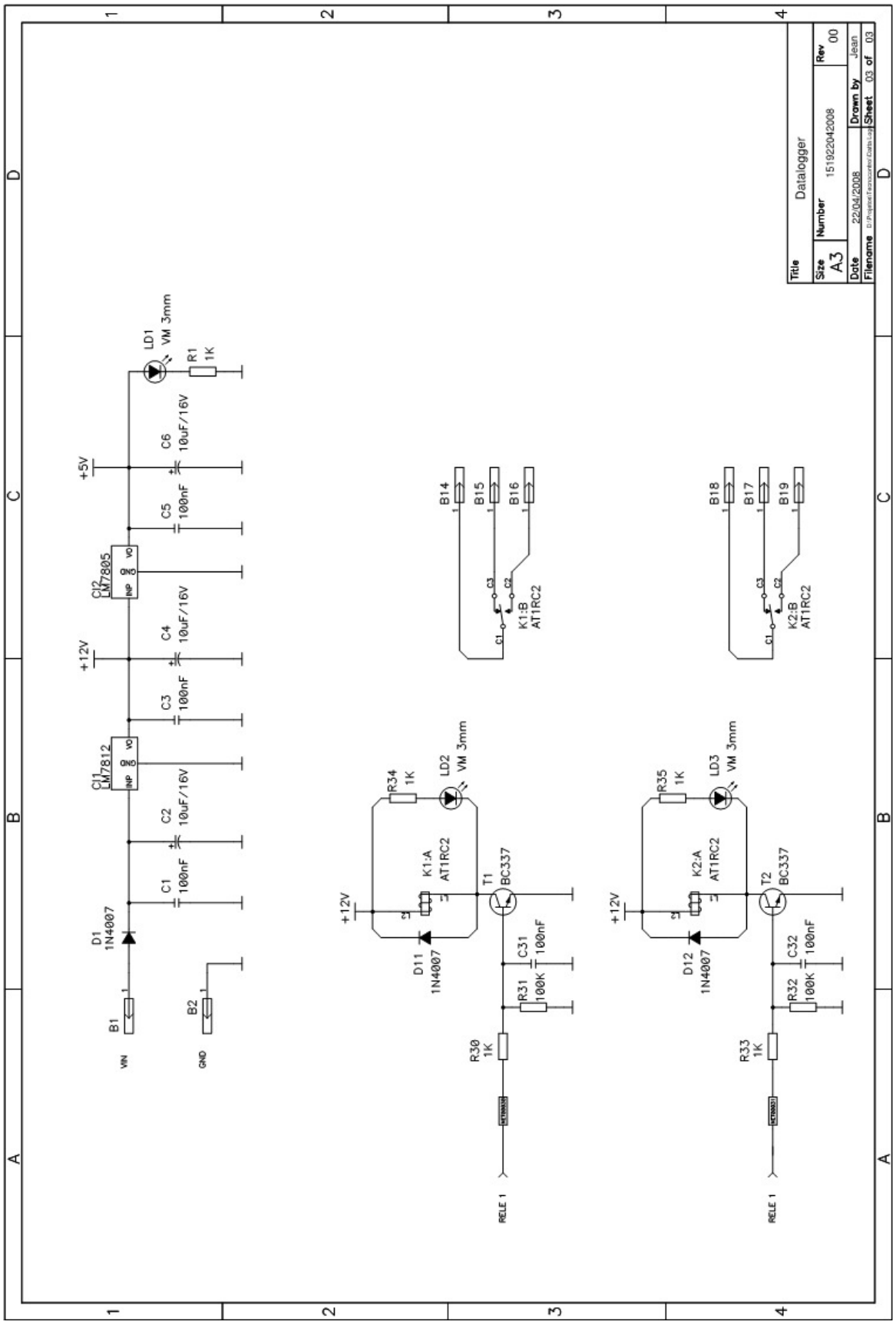
⑤ ABSOLUTE MAXIMUM RATINGS(25°C)

PARAMETER	SYMBOL	MIN.	MAX.	UNIT
Supply Voltage Logic	V_{DD}	-0.3	+7.0	V
Supply Voltage Driver	V_{LCD}	V_{DD}-15.0	V_{DD}+0.3	V
Input Voltage	V_{IN}	-0.3	V_{DD}+0.3	V

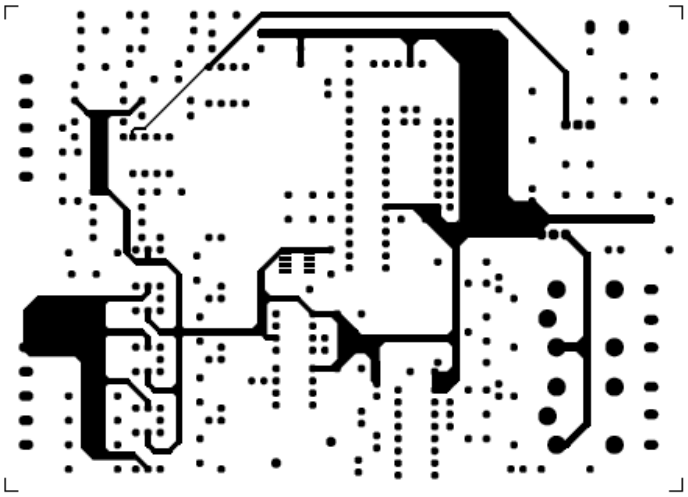
Anexo 7 – Desenho do Esquema Elétrico do Coletor de Dados

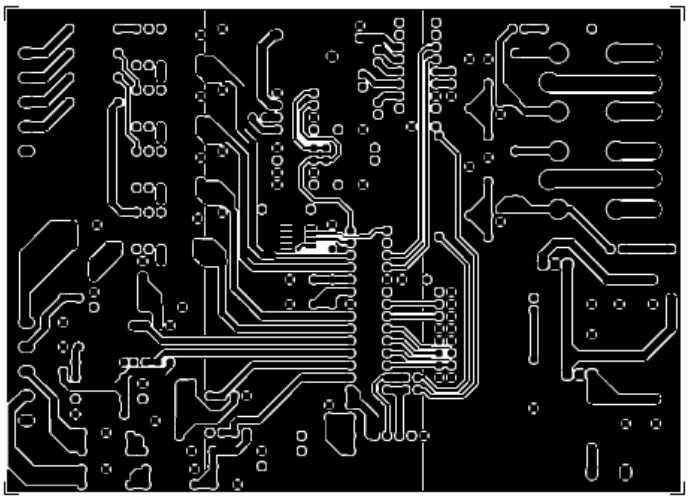


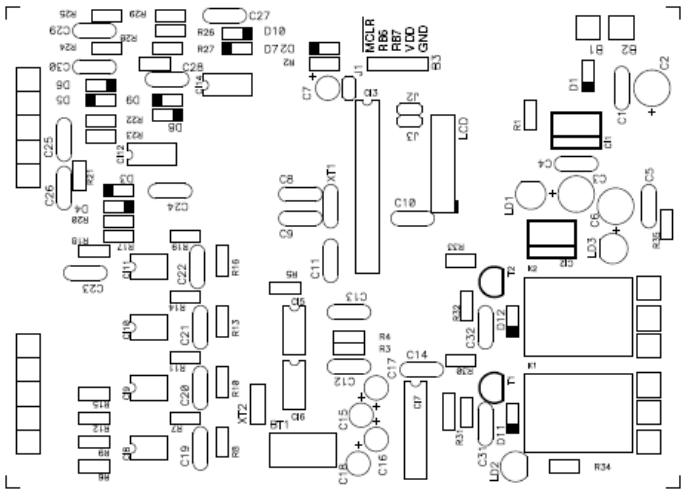

Title Datalogger
Size A3
Number 1519220042008
Rev 00
Date 22/04/2008
Drawn by Jean
Filename D:\Projetos\Tecnosinis\Data Logger\
Sheet 02 of 03
IN_DIG_1
IN_DIG_2
IN_DIG_3
IN_DIG_4
IN_ANA_1
IN_ANA_2
IN_ANA_3
IN_ANA_4
+5V
+12V
4N25
LM358AN
1N4148


Title Datalogger
Size A3
Number 151922042008
Rev 00
Date 22/04/2008
Drawn by Jean
Filename D:\Projetos\Transceiver\Data Log\
Sheet 03 of 03
LD1 VM 3mm
R1 1K
+5V
C6 10uF/16V
C5 100nF
CI2 LM7805
+12V
C4 10uF/16V
C3 100nF
CI1 LM7812
C2 10uF/16V
C1 100nF
D1 1N4007
B1 VIN
B2 GND
B14
B15
B16
K1:B AT1RC2
B18
B17
B19
K2:B AT1RC2
R34 1K
LD2 VM 3mm
K1:A AT1RC2
T1 BC337
D11 1N4007
C31 100nF
R31 100K
R30 1K
RELE 1
R35 1K
LD3 VM 3mm
K2:A AT1RC2
T2 BC337
D12 1N4007
C32 100nF
R32 100K
R33 1K
RELE 1

Anexo 8 – Desenho da Placa de Circuito Impresso







Anexo 9 – Código Fonte do Microcontrolador

```
/*=============================================================================
        DEFINIÇÃO DAS VARIÁVEIS INTERNAS DO PIC
=============================================================================*/
#include  <18f2525.h>          // microcontrolador utilizado
#device adc=8
/*=============================================================================
                Configurações para gravação
=============================================================================*/
#FUSES NOWDT
#FUSES WDT128
#FUSES XT
#FUSES PROTECT
#FUSES BROWNOUT
#FUSES BORV20
#FUSES PUT
#FUSES NOSTVREN
#FUSES NODEBUG
#FUSES NOLVP
#FUSES NOWRT
#FUSES NOWRTD
#FUSES NOWRTB
#FUSES NOWRTC
#FUSES NOCPD
#FUSES NOCPB
#FUSES NOEBTR
#FUSES NOEBTRB
/*=============================================================================
        Definições das prioridades na interrupção
=============================================================================*/
#priority timer1,rda
/*=============================================================================
        Definições para uso de Rotinas de Delay
=============================================================================*/
#use       delay(clock=4000000,RESTART_WDT)
#use       rs232(baud= 9600,xmit=pin_c6,rcv=pin_c7, parity=n , bits=8)
/*=============================================================================
                Constantes internas
=============================================================================*/
//A definição de constantes facilita a programação e a manutenção.
#define    t_filtro 500                                    // tamanho do filtro
#define    tempo    1500                                   // tempo em mS
#define delta_timer1 (65536 - 64535)
#define nop delay_cycles(1)
/*=============================================================================
        Definição e inicialização das variáveis
 =============================================================================*/
```

```
//Neste bloco estão definidas as variáveis globais do programa.
char        dados_rx[150];
float       conversao;
long int    endereco;
long int    conta_bloco;
long int    tempo_gravar;
long int    var1;
long int    var2;
int tempo_grava1;
int tempo_grava2;
int tempo_grava3;
int tempo_grava4;
int         ContaBytes=0;
int         timeout_rx=0;
int         dia;
int         mes;
int         ano;
int         hora;
int          minuto;
int         segundo;
int         var_temp1;
int         var_temp2;
int         val_ad_1=0;
int         val_ad_2=0;
int         val_ad_3=0;
int         val_ad_4=0;
int         dado;
int         end_low;
int         end_hi;
int         ctrl_escr;
int         ctrl_le;
int         estado;
int         atualiza=3;
int         conta_tela=0;
int         gravar;
int         banco_0=0;
int         banco_1=0;
int         estado_r1;
int         estado_r2;
int         var3;
int         var4;
BYTE sec;
BYTE min;
BYTE hrs;
BYTE day;
BYTE month;
BYTE yr;
BYTE dow;
/*=========================================================================
                        Bibliotecas
=========================================================================*/
```

```
#include  <D:\Bibliotecas\lcd_4vias.c>
#include  <D:\Bibliotecas\ds1307.c>
#include  <D:\Bibliotecas\conversores.c>
#include  <D:\Bibliotecas\24AA1025.c>
/*================================================================================
              Definição e inicialização das flags
 ================================================================================*/
short int   flag_trata_rx=False,flag_ajusta_hora=false,flag_atualiza=false,
            flag_grava=false,flag_buffer_vazio=true,flag_envia_dados=false;
/*================================================================================
               Definição e inicialização dos port's
================================================================================*/
#use        fast_io(a)
#use        fast_io(b)
#use        fast_io(c)
#byte       porta = 0xf80
#byte       portb = 0xf81
#byte       portc = 0xf82
/*================================================================================
                          ENTRADAS
================================================================================*/
//entradas analógicas
#bit        in_ana_1 =          porta.0
#bit        in_ana_2 =          porta.1
#bit        in_ana_3 =          porta.2
#bit        in_ana_4 =          porta.3
//entrdas digitais
#bit        in_dig_1  =         porta.4
#bit        in_dig_2  =         porta.5
#bit        in_dig_3  =         portc.0
#bit        in_dig_4  =         portc.1
/*================================================================================
                          SAÍDAS
================================================================================*/
#bit rele1 =            portb.6
#bit rele2 =            portb.7
/*================================================================================
               FLAGS DE IDENTIFICAÇÃO DO ESTADO DOS PINOS
================================================================================*/
#bit        bit_0 = estado.0     //Estado do Relé 1
#bit        bit_1 = estado.1     //Estado do Relé 2
#bit        bit_2 = estado.2     //Estado da entrada digital 1
#bit        bit_3 = estado.3     //Estado da entrada digital 2
#bit        bit_4 = estado.4     //Estado da entrada digital 3
#bit        bit_5 = estado.5     //Estado da entrada digital 4
#bit        bit_6 = estado.6     //Sem função definida
#bit        bit_7 = estado.7     //Sem função definida
/*================================================================================
            TELAS DE VISUALIZAÇÃO DOS VALROES E ESTADOS DAS ENTRADAS E SAÍDAS
 ================================================================================*/
```

```
void telas()
{
bit_6=false;
bit_7=false;
switch (conta_tela)
        {
        case 0 :  set_adc_channel(0);
                  delay_us(50);
                  conversao=read_adc();
                  conversao_ad_tensao();
                  envia_byte_lcd(0,0xC0);                    // posiciona o cursor na linha 0, coluna 0
                  printf (escreve_lcd, "ENT. AN1-> %2.2fV     ",conversao);    // imprime menssagem no display
                  break;
        case 1 :  set_adc_channel(1);
                  delay_us(50);
                  conversao=read_adc();
                  conversao_ad_temp();
                  envia_byte_lcd(0,0xC0);        // posiciona o cursor na linha 0, coluna 0
                  printf (escreve_lcd, "ENT. AN2-> %2.2fC     ",conversao);    // imprime menssagem no display
                  break;
        case 2 :  set_adc_channel(2);
                  delay_us(50);
                  conversao=read_adc();
                  conversao_ad_press1();
                  envia_byte_lcd(0,0xC0);                          // posiciona o cursor na linha 0, coluna 0
                  printf (escreve_lcd, "ENT. AN3-> %2.2fmca     ",conversao);// imprime menssagem no display
                  break;
        case 3 :  set_adc_channel(3);
                  delay_us(50);
                  conversao=read_adc();
                  conversao_ad_press2();
                  envia_byte_lcd(0,0xC0);                          // posiciona o cursor na linha 0, coluna 0
                  printf (escreve_lcd, "ENT. AN4-> %2.2fmca     ",conversao);// imprime menssagem no display
                  break;
        case 4 :  if (in_dig_1)
                  {
                  envia_byte_lcd(0,0xC0);                          // posiciona o cursor na linha 0, coluna 0
                  printf (escreve_lcd, "ENT. DIG1-> ON      ");        // imprime menssagem no display
                  }
                  else
                  {
                  envia_byte_lcd(0,0xC0);                          // posiciona o cursor na linha 0, coluna 0
                  printf (escreve_lcd, "ENT. DIG1-> OFF      ");       // imprime menssagem no display
                  }
                  break;
        case 5 :  if (in_dig_2)
                  {
                  envia_byte_lcd(0,0xC0);                          // posiciona o cursor na linha 0, coluna 0
                  printf (escreve_lcd, "ENT. DIG2-> ON      ");        // imprime menssagem no display
                  }
                  else
```

```
            {
            envia_byte_lcd(0,0xC0);                         // posiciona o cursor na linha 0, coluna 0
            printf (escreve_lcd, "ENT. DIG2-> OFF     ");   // imprime menssagem no display
            }
            break;
case 6 :    if (in_dig_3)
            {
            envia_byte_lcd(0,0xC0);                         // posiciona o cursor na linha 0, coluna 0
            printf (escreve_lcd, "ENT. DIG3-> ON      ");   // imprime menssagem no display
            }
            else
            {
            envia_byte_lcd(0,0xC0);                         // posiciona o cursor na linha 0, coluna 0
            printf (escreve_lcd, "ENT. DIG3-> OFF     ");   // imprime menssagem no display
            }
            break;
case 7 :    if (in_dig_4)
            {
            envia_byte_lcd(0,0xC0);                         // posiciona o cursor na linha 0, coluna 0
            printf (escreve_lcd, "ENT. DIG4-> ON      ");   // imprime menssagem no display
            }
            else
            {
            envia_byte_lcd(0,0xC0);                         // posiciona o cursor na linha 0, coluna 0
            printf (escreve_lcd, "ENT. DIG4-> OFF     ");   // imprime menssagem no display
            }
            break;
case 8 :    if (rele1)
            {
            envia_byte_lcd(0,0xC0);                         // posiciona o cursor na linha 0, coluna 0
            printf (escreve_lcd, "RELE1-> ON          ");   // imprime menssagem no display
            }
            else
            {
            envia_byte_lcd(0,0xC0);                         // posiciona o cursor na linha 0, coluna 0
            printf (escreve_lcd, "RELE1-> OFF         ");   // imprime menssagem no display
            }
            break;
case 9 :    if (rele2)
            {
            envia_byte_lcd(0,0xC0);                         // posiciona o cursor na linha 0, coluna 0
            printf (escreve_lcd, "RELE2-> ON          ");   // imprime menssagem no display
            }
            else
            {
            envia_byte_lcd(0,0xC0);                         // posiciona o cursor na linha 0, coluna 0
            printf (escreve_lcd, "RELE2-> OFF         ");   // imprime menssagem no display
            }
            break;
case 10:    break;
}//fim do switch case
```

```
}
/*=============================================================================
                    Ajusta Relógio
=============================================================================*/
void acerta_hora()
{
ds1307_get_date(day,month,yr,dow);
ds1307_get_time(hrs,min,sec);
if          (flag_buffer_vazio)  //Verifica se pode mostrar hora e data ou
            {
            //DATA
            envia_byte_lcd(0,0x81);
            printf(escreve_lcd,"\%02d/\%02d/\%02d ",day,month,yr);
            //HORA
            envia_byte_lcd(0,0x8A);
            printf(escreve_lcd,"\%02d:\%02d:\%02d", hrs,min,sec);
            }//fim do iff de buffer vazio
}
/*=============================================================================
                    Configurações do Pic
=============================================================================*/
void main()
{
// configura microcontrolador
setup_timer_1 (t1_internal | t1_div_by_1);
setup_adc_ports(AN0_TO_AN3|VSS_VDD);
setup_adc(ADC_CLOCK_DIV_8);
// configura os tris
set_tris_a(0b11111111);       // configuração da direção dos pinos de I/O
set_tris_b(0b00000000);
set_tris_c(0b10000011);
// inicializa os ports
porta=0x00;                                     // limpa porta
portb=0x00;                                     // limpa portb
portc=0x00;                                     // limpa portc
/*=============================================================================
                    Configurações Iniciais
 =============================================================================*/
set_timer1(delta_timer1);               // carraga timer1
enable_interrupts(INT_TIMER1);          //Habilita a interrupção de TIMER1
enable_interrupts(INT_RDA);
enable_interrupts(GLOBAL);              //Habilitação global das interrupções
// inicialização do lcd
inicializa_lcd();                                       //configura lcd
envia_byte_lcd(0,0x85);                         // posiciona o cursor na linha 0, coluna 0
printf (escreve_lcd, "DATALOGGER");  // imprime menssagem no display
printf("**Tecnocontrol Tecnologia e Sistemas Ltda**\r\n");
printf("**          DATALOGGER          **\r\n");
printf("**          Versao 1.0          **\r\n");
printf("**           %s           **\r\n", __DATE__);
```

```
gravar=read_eeprom(0);        //Lê o valor gravado na posição 0 da eeprom interna
banco_0=read_eeprom(1); //Lê o valor gravado na posição 1 da eeprom interna
banco_1=read_eeprom(2); //Lê o valor gravado na posição 2 da eeprom interna
end_low=read_eeprom(3);
end_hi=read_eeprom(4);
endereco=make16 (end_hi,end_low);
end_low=read_eeprom(5);
end_hi=read_eeprom(6);
conta_bloco=make16 (end_hi,end_low);
if          (gravar==0xff)
            {
            gravar=0;
            banco_0=1;
            banco_1=0;
            endereco=0;
            conta_bloco=0;
            }
disable_interrupts(GLOBAL);
write_eeprom(0,gravar);
write_eeprom(1,banco_0);
write_eeprom(2,banco_1);
enable_interrupts(GLOBAL);
//Verifica o último estado dos relés
estado_r1=read_eeprom(7);
if          (estado_r1==1) rele1=true;
else        rele1=false;

estado_r2=read_eeprom(8);
if          (estado_r2==1) rele2=true;
else        rele2=false;

tempo_grava1=read_eeprom(9);
tempo_grava2=read_eeprom(10);
tempo_grava3=read_eeprom(11);
tempo_grava4=read_eeprom(12);

if          (tempo_grava1==0xff)
            {
            tempo_gravar=60;
            }
else
            {
            var1=(tempo_grava1-0x30);
            var2=(tempo_grava2-0x30);
            var3=(tempo_grava3-0x30);
            var4=(tempo_grava4-0x30);
            var1=var1*1000;
            var2=var2*100;
            var3=var3*10;
            tempo_gravar=(var1+var2+var3+var4);
            }
```

```
/*=============================================================================
                    Rotina Principal
 =============================================================================*/
loop:
        for(;;)//while(TRUE)                                    // rotina principal
        {
        //Rotina que verifica se é para gravar pacotes na eeprom externa
        if      (flag_grava)
                {
                flag_grava=false;
                if      (gravar)
                        {
                        //Verifica se já não encheu os bancos
                        if      ((!banco_0)&&(!banco_1))
                                {
                                //Menssagem de Estouro dos Bancos
                                flag_buffer_vazio=false;        //Não libera para mostrar data e hora
                                limpa_lcd();                    //Limpa LCD
                                envia_byte_lcd(0,0x83);         // posiciona o cursor na linha 0, coluna 0
                                printf (escreve_lcd, "MEMORIA CHEIA     "); // imprime menssagem no display
                                }
                        else
                                {
                                conta_bloco++;      //incrementa a contagem de qtos blocos já foram gravados
                                endereco++;         //incrementa endereço para armazenar dados
                                //Rotina que verifica se é para escrever no BANCO 0 ou no BANCO 1 e também se
                                //não estão cheios
                                if      (conta_bloco >= 6552)
                                        {
                                        if      (banco_0)
                                                {
                                                conta_bloco=0;
                                                endereco=0;
                                                banco_0=0;
                                                banco_1=1;
                                                disable_interrupts(GLOBAL);
                                                write_eeprom(1,banco_0);
                                                write_eeprom(2,banco_1);
                                                enable_interrupts(GLOBAL);
                                                }
                                        if      (banco_1)
                                                {
                                                conta_bloco=0;
                                                endereco=0;
                                                banco_0=0;
                                                banco_1=0;
                                                disable_interrupts(GLOBAL);
                                                write_eeprom(1,banco_0);
                                                write_eeprom(2,banco_1);
                                                enable_interrupts(GLOBAL);
                                                }
```

```
            }
if          (banco_0)
            {
            ctrl_escr=ctrl_escr_low;
            }
if          (banco_1)
            {
            ctrl_escr=ctrl_escr_hi;
            }
//Grava indicador de inicio do bloco
dado=0xfe;
write_eeprom_ext(); //Envia dado para ser gravado na posição de memória
                    //passada pelo endereco
//Grava dia
endereco++;
dado=day;
write_eeprom_ext(); //Envia dado para ser gravado na posição de memória
                    //passada pelo endereco
//Grava mes
endereco++;
dado=month;
write_eeprom_ext(); //Envia dado para ser gravado na posição de memória
                    //passada pelo endereco
//Grava hora
endereco++;
dado=hrs;
write_eeprom_ext(); //Envia dado para ser gravado na posição de memória
                    //passada pelo endereco
//Grava minuto
endereco++;
dado=min;
write_eeprom_ext(); //Envia dado para ser gravado na posição de memória
                    //passada pelo endereco
//Grava Entrada Analógica 1
endereco++;
set_adc_channel(0);
delay_us(50);
dado=read_adc();
write_eeprom_ext(); //Envia dado para ser gravado na posição de memória
                    //passada pelo endereco
//Grava Entrada Analógica 2
endereco++;
set_adc_channel(1);
delay_us(50);
dado=read_adc();
write_eeprom_ext(); //Envia dado para ser gravado na posição de memória
                    //passada pelo endereco
//Grava Entrada Analógica 3
endereco++;
set_adc_channel(2);
delay_us(50);
```

```
                        dado=read_adc();
                        write_eeprom_ext(); //Envia dado para ser gravado na posição de memória
                                            // passada pelo endereco
                        //Grava Entrada Analógica 4
                        endereco++;
                        set_adc_channel(3);
                        delay_us(50);
                        dado=read_adc();
                        write_eeprom_ext(); //Envia dado para ser gravado na posição de memória
                                            //passada pelo endereco
                        //Grava Byte com estados das entradas digitais e dos relés
                        if      (in_dig_1) bit_2=true; else bit_2=false;
                        if      (in_dig_2) bit_3=true; else bit_3=false;
                        if      (in_dig_3) bit_4=true; else bit_4=false;
                        if      (in_dig_4) bit_5=true; else bit_5=false;
                        if      (rele1) bit_0=true;   else bit_0=false;
                        if      (rele2)    bit_1=true;          else bit_1=false;
                        endereco++;
                        dado=estado;
                        write_eeprom_ext(); //Envia dado para ser gravado na posição de memória
                                            //passada pelo endereco
                        //Grava endereço na EEPROM interna
                        end_low = make8 (endereco,0);//Quebra enderco em HI e LOW
                        end_hi = make8 (endereco,1);
                        disable_interrupts(GLOBAL);
                        write_eeprom(3,end_low);
                        write_eeprom(4,end_hi);
                        enable_interrupts(GLOBAL);
                        //Grava bloco na EEPROM interna
                        end_low = make8 (conta_bloco,0);//Quebra enderco em HI e LOW
                        end_hi = make8 (conta_bloco,1);
                        disable_interrupts(GLOBAL);
                        write_eeprom(5,end_low);
                        write_eeprom(6,end_hi);
                        enable_interrupts(GLOBAL);
                        }//Fim do else de bancos cheios
                printf("\nGravou no Endereco: %lu\r\n",conta_bloco);      //Mensagem confirmando
                                                                        //a solicitação externa
                }//fim do if(gravar)
        }//fim do if (flag_grava)
//Rotina que envia dados armazenados na EEPROM externa pela serial
if      (flag_envia_dados)
        {
        flag_envia_dados=false;
        if      (banco_0)
                {
                //Enviar dados armazenados no BANCO 0
                ctrl_le = ctrl_le_low;
                ctrl_escr = ctrl_escr_low;
                for     (endereco=0;endereco <= 65520;endereco++)    //envia dados até encontrar o $
                                                                     //q finaliza a informação.
```

```
                    {
                    read_eeprom_ext();
                    if      (dado == 0xfe)
                            {
                            printf ("\r\n");
                            }
                    if      (dado == 0xff)
                            {
                            break;
                            }
                    printf ("%u ",dado);
                    }
            }//Fim do if (banco_0)
        if  (banco_1)
            {
            //Enviar dados armazenados no BANCO 0
            ctrl_le = ctrl_le_hi;
            ctrl_escr = ctrl_escr_hi;
            for     (endereco=0;endereco <= 65520;endereco++)    //envia dados até encontrar o $
                                                                 //q finaliza a informação.
                    {
                    read_eeprom_ext();
                    if      (dado == '$')
                            {
                            printf ("\r\n");
                            }
                    if      (dado == 0xff)
                            {
                            break;
                            }
                    printf ("%u ",dado);
                    }
            }//Fim do if (banco_0)
        }
//Rotina que acerta o relógio
if      (flag_ajusta_hora)
        {
        flag_ajusta_hora=false;
        acerta_hora();
        }
//Rotina que atualiza as telas mostrando os valores medidos a cada 3 segundos
if      (flag_atualiza)
        {
        flag_atualiza=false;
        conta_tela++;
        if      (conta_tela==10)
                {
                conta_tela=0;
                }
        telas();
        }
```

```
// Rotina que trata a recepção de dados pela serial
if          (flag_trata_rx)
            {
            flag_trata_rx=False;            //Evita chamasdas recursivas.
            ContaBytes=0;
            //Rotina que programa o RTC
            if ((dados_rx[0]=='P')&&(dados_rx[1]=='R')&&(dados_rx[2]=='O')&&(dados_rx[3]=='G'))
                        {
                        var_temp1=(dados_rx[4] - 0x30);
                        var_temp2=(dados_rx[5] - 0x30);
                        dia=(var_temp1 * 10) + var_temp2;

                        var_temp1=(dados_rx[7] - 0x30);
                        var_temp2=(dados_rx[8] - 0x30);
                        mes=(var_temp1 * 10) + var_temp2;

                        var_temp1=(dados_rx[10] - 0x30);
                        var_temp2=(dados_rx[11] - 0x30);
                        ano=(var_temp1 * 10) + var_temp2;

                        var_temp1=(dados_rx[13] - 0x30);
                        var_temp2=(dados_rx[14] - 0x30);
                        hora=(var_temp1 * 10) + var_temp2;

                        var_temp1=(dados_rx[16] - 0x30);
                        var_temp2=(dados_rx[17] - 0x30);
                        minuto=(var_temp1 * 10) + var_temp2;

                        var_temp1=(dados_rx[19] - 0x30);
                        var_temp2=(dados_rx[20] - 0x30);
                        segundo=(var_temp1 * 10) + var_temp2;

                        ds1307_set_date_time(dia,mes,ano,2,hora,minuto,segundo);

                        printf("Programação do RTC OK!\r\n");  //Mensagem confirmando a solicitação externa

                        }//Fim do if recebe dados
            //Rotina que lê as entradas analógicas e envia pela serial
            if ((dados_rx[0]=='$')&&(dados_rx[1]=='0')&&(dados_rx[2]=='1'))
                        {
                        val_ad_1=0;
                        val_ad_2=0;
                        val_ad_3=0;
                        val_ad_4=0;

                        printf("\nValores dos Canais Analógicos\r\n");       //Mensagem confirmando a solicitação externa

                        set_adc_channel(0);
                        delay_us(50);
                        conversao=read_adc();
```

```
                conversao_ad_tensao();
                printf("AD1= %2.2fV\r\n",conversao);

                set_adc_channel(1);
                delay_us(50);
                conversao=read_adc();
                conversao_ad_temp();
                printf("AD2= %2.2f C\r\n",conversao);

                set_adc_channel(2);
                delay_us(50);
                conversao=read_adc();
                conversao_ad_press1();
                printf("AD3= %2.2fmca\r\n",conversao);

                set_adc_channel(3);
                delay_us(50);
                conversao=read_adc();
                conversao_ad_press2();
                printf("AD4= %2.2fmca\r\n",conversao);
                }//fim do if ((dados_rx[$]=='L')&&(dados_rx[1]=='0')&&(dados_rx[2]=='1'))

//Rotina que lê as entradas digitas e envia pela serial
if      ((dados_rx[0]=='$')&&(dados_rx[1]=='0')&&(dados_rx[2]=='2'))
                {
                printf("\nEstado das Entradas Digitais\r\n");          //Mensagem confirmando a solicitação externa
                if      (in_dig_1)
                        {
                        printf("Entrada Digital 1 = ON\r\n");
                        }
                else
                        {
                        printf("Entrada Digital 1 = OFF\r\n");
                        }
                if      (in_dig_2)
                        {
                        printf("Entrada Digital 2 = ON\r\n");
                        }
                else
                        {
                        printf("Entrada Digital 2 = OFF\r\n");
                        }
                if      (in_dig_3)
                        {
                        printf("Entrada Digital 3 = ON\r\n");
                        }
                else
                        {
                        printf("Entrada Digital 3 = OFF\r\n");
                        }
                if      (in_dig_4)
```

```
                {
                printf("Entrada Digital 4 = ON\r\n");
                }
        else
                {
                printf("Entrada Digital 4 = OFF\r\n");
                }
        }//fim do if ((dados_rx[$]=='L')&&(dados_rx[0]=='E')&&(dados_rx[2]=='2'))

//Rotina que inicia a gravação dos dados na memória externa
if      ((dados_rx[0]=='$')&&(dados_rx[1]=='0')&&(dados_rx[2]=='3'))
        {
        printf("\nIniciar Gravação dos Dados\r\n");          //Mensagem confirmando a solicitação externa
        gravar=1;
        disable_interrupts(GLOBAL);
        write_eeprom(0,gravar);
        enable_interrupts(GLOBAL);
        endereco=0;
        }//fim do ((dados_rx[0]=='$')&&(dados_rx[1]=='0')&&(dados_rx[2]=='3'))

//Rotina que pára a gravaçào dos dados na memória externa.
if      ((dados_rx[0]=='$')&&(dados_rx[1]=='0')&&(dados_rx[2]=='4'))
        {
        printf("\nParar Gravação dos Dados\r\n");          //Mensagem confirmando a solicitação externa
        gravar=0;
        disable_interrupts(GLOBAL);
        write_eeprom(0,gravar);
        enable_interrupts(GLOBAL);
        }//fim do ((dados_rx[0]=='4')&&(dados_rx[1]=='0')&&(dados_rx[2]=='4'))

//Rotina que liga o Rele 1
if      ((dados_rx[0]=='$')&&(dados_rx[1]=='0')&&(dados_rx[2]=='5'))
        {
        printf("\nLigado Rele 1\r\n");   //Mensagem confirmando a solicitação externa
        rele1=true;
        estado_r1=1;
        disable_interrupts(GLOBAL);
        write_eeprom(7,estado_r1);
        enable_interrupts(GLOBAL);
        }

//Rotina que desliga o Rele 1
if      ((dados_rx[0]=='$')&&(dados_rx[1]=='0')&&(dados_rx[2]=='6'))
        {
        printf("\nDesligado Rele 1\r\n");          //Mensagem confirmando a solicitação externa
        rele1=false;
        estado_r1=0;
        disable_interrupts(GLOBAL);
        write_eeprom(7,estado_r1);
        enable_interrupts(GLOBAL);
        }
```

```
//Rotina que liga o Rele 2
if          ((dados_rx[0]=='$')&&(dados_rx[1]=='0')&&(dados_rx[2]=='7'))
            {
            printf("\nLigado Rele 2\r\n");   //Mensagem confirmando a solicitação externa
            rele2=true;
            estado_r2=1;
            disable_interrupts(GLOBAL);
            write_eeprom(8,estado_r2);
            enable_interrupts(GLOBAL);
            }

//Rotina que desliga o Rele 2
if          ((dados_rx[0]=='$')&&(dados_rx[1]=='0')&&(dados_rx[2]=='8'))
            {
            printf("\nDesligado Rele 2\r\n");          //Mensagem confirmando a solicitação externa
            rele2=false;
            estado_r2=0;
            disable_interrupts(GLOBAL);
            write_eeprom(8,estado_r2);
            enable_interrupts(GLOBAL);
            }

//Rotina que solicita leitura dos dados armazenados na EEPROM externa
if          ((dados_rx[0]=='$')&&(dados_rx[1]=='0')&&(dados_rx[2]=='9'))
            {
            printf("\nEnviando Dados da EEPROM Externa\r\n");          //Mensagem confirmando
                                                                        //a solicitação externa
            flag_envia_dados=true;
            }

//Rotina que escreve 0XFF na EEPROM externa
if          ((dados_rx[0]=='$')&&(dados_rx[1]=='1')&&(dados_rx[2]=='0'))
            {
            printf("\nEscrevendo 0xFE na EEPROM Externa\r\n");         //Mensagem confirmando
                                                                        // a solicitação externa
            printf("\nAguarde...\r\n");       //Mensagem confirmando a solicitação externa

            for         (endereco=1;endereco <= 1000;endereco++)        //envia dados até encontrar o $
                                                                        //q finaliza a informação.
                        {
                        ctrl_escr=ctrl_escr_low;
                        dado=0xff;
                        write_eeprom_ext(); //Envia dado para ser gravado na posição de memória
                                            // passada pelo endereco
                        }
            printf("\nTerminou Escrita 0xFE\r\n");    //Mensagem confirmando a solicitação externa
            }
//Rotina que altera o valor do tempo para escrita na memória externa
if          ((dados_rx[0]=='$')&&(dados_rx[1]=='1')&&(dados_rx[2]=='1'))
            {
            disable_interrupts(GLOBAL);
```

```
                                write_eeprom  (9,dados_rx[3]);
                                write_eeprom  (10,dados_rx[4]);
                                write_eeprom  (11,dados_rx[5]);
                                write_eeprom  (12,dados_rx[6]);
                                enable_interrupts(GLOBAL);

                                tempo_grava1=read_eeprom(9);
                                tempo_grava2=read_eeprom(10);
                                tempo_grava3=read_eeprom(11);
                                tempo_grava4=read_eeprom(12);

                                var1=(tempo_grava1-0x30);
                                var2=(tempo_grava2-0x30);
                                var3=(tempo_grava3-0x30);
                                var4=(tempo_grava4-0x30);
                                var1=var1*1000;
                                var2=var2*100;
                                var3=var3*10;

                                tempo_gravar=(var1+var2+var3+var4);

                                printf("\nIntervalo de Gravação dos Dados é de %lu\r\n",tempo_gravar); //Mensagem
                                                                                                    //confirmando a solicitação externa
                                }
                        }//Fim do if            (flag_trata_rx)
        }//fim do for(;;)
}//fim do main()

/*=============================================================================
                Fim do Programa
=============================================================================*/

/*=============================================================================
                INTERRUPÇÃO RDA - SERIAL
=============================================================================*/
#INT_RDA
void serial_teste()
{
        timeout_rx=50;                  //Tempo maximo entre bytes do mesmo frame.
        dados_rx[ContaBytes] = getc();  //Salva.
        if      (ContaBytes<50)
                {
                ContaBytes++;           //Evita buffer overrun.
                }
        else
                {
                ContaBytes=100;
                }
}
```

```
/*================================================================================
                    INTERRUPÇÃO TIMER1
================================================================================*/
#INT_TIMER1
void timer1_teste()
{
        static int conta10ms=10,conta100ms=10,conta1s=10;

        set_timer1(64558+get_timer1());                         // carraga timer1
//================================================================================
//Tarefas a cada 1 milisegundo:

//Trata timeou RX.
        if      (timeout_rx)
                {
                if      (!--timeout_rx)
                        {
                        flag_trata_rx=True;    //Avisa pra tratar dados recebidos.
                        }
                }

//Fim das tarefas a cada 1 milisegundo:
//================================================================================

   if(!--conta10ms){
                conta10ms=10;
//================================================================================
//Tarefas a cada 10 milisegundos:


//Fim das tarefas a cada 10 milisegundos:
//================================================================================

   if(!--conta100ms){
                conta100ms=10;
//================================================================================
//Tarefas a cada 100 milisegundos:


//Fim das tarefas a cada 100 milisegundos:
//================================================================================

   if(!--conta1s){
                conta1s=10;
//================================================================================
//Tarefas a cada 1 segundos:
        flag_ajusta_hora=true;
        if      (!--atualiza)
                {
                atualiza=3;
                flag_atualiza=True;    //Avisa pra tratar dados recebidos.
```

```
                    }
            if      (!--tempo_gravar)
                    {
                    tempo_gravar=(var1+var2+var3+var4);
                    flag_grava=true;    //Rotina que verifica se ja se passaram
                                        //5 minutos para gravar dados na eeprom externa
                    }
//Fim das tarefas a cada 1 segundos:
//=============================================================================
                                                }//if(!--conta1s).
                                        }//if(!--conta100ms).
                                }//if(!--conta1s).
}
```

Anexo 10 – Biblioteca do LCD 4 vias

```
/*=============================================================================

  Biblioteca para Comunicaçao com Dsiplay de Cristal Líquido

  EXEMPLO:

        // inicialização do lcd
        inicializa_lcd();                   //configura lcd
        envia_byte_lcd(0,0x80);             // posiciona o cursor na linha 0, coluna 0
        printf (escreve_lcd, "AGUARDE ");   // imprime menssagem no display

=============================================================================*/

/*=============================================================================
                    PINOS DO LCD
=============================================================================*/
  // Estas são as definições dos pinos que o LCD utiliza.
  //Deverão ser ajustadas de acordo com os pinos a serem utilizados para Comunicação
  //com o LCD
  #define lcd_enable        pin_b1 // pino enable do LCD
  #define lcd_rs            pin_b0 // pino rs do LCD
  #define lcd_db4           pin_b2 // pino de dados d4 do LCD
  #define lcd_db5           pin_b3 // pino de dados d5 do LCD
  #define lcd_db6           pin_b4 // pino de dados d6 do LCD
  #define lcd_db7           pin_b5 // pino de dados d7 do LCD
  /*=============================================================================
                              LÊ O NIBBLE INFERIOR E ENVIA AO LCD
  =============================================================================*/
  //Esta rotina lê o "Nibble" inferior de uma variável e envia para o LCD.
  void envia_nibble_lcd(int dado)
  {
        //Carrega as vias de dados (pinos) do LCD de acordo com o nibble lido
        output_bit(lcd_db4, bit_test(dado,0)); //Carrega DB4 do LCD com o bit DADO<0>
        output_bit(lcd_db5, bit_test(dado,1)); //Carrega DB5 do LCD com o bit DADO<1>
        output_bit(lcd_db6, bit_test(dado,2)); //Carrega DB6 do LCD com o bit DADO<2>
        output_bit(lcd_db7, bit_test(dado,3)); //Carrega DB7 do LCD com o bit DADO<3>
        //Gera um pulso de enable
        output_high(lcd_enable); // ENABLE = 1
        delay_us(1); // Recomendado para estabilizar o LCD
        output_low(lcd_enable); // ENABLE = 0
        return; // Retorna ao ponto de chamada da função
  }
  /*=============================================================================
                              ENVIA DADOS OU COMANDO PARA O LCD
  =============================================================================*/
  // ENDEREÇO = 0 -> a variável DADO será uma instrução
  // ENDEREÇO = 1 -> a variável DADO será um caractere
  void envia_byte_lcd(boolean endereco, int dado)
  {
        output_bit(lcd_rs,endereco); // Seta o bit RS para instrução ou caractere
        delay_us(100); // Aguarda 100 us para estabilizar o pino do LCD
        output_low(lcd_enable); // Desativa a linha ENABLE
        envia_nibble_lcd(dado>>4); // Envia a parte ALTA do dado/comando
        envia_nibble_lcd(dado & 0x0f);// Limpa a parte ALTA e envia a parte BAIXA do
        // dado/comando
        delay_us(40); // Aguarda 40us para estabilizar o LCD
        return; // Retorna ao ponto de chamada da função
  }
  /*=============================================================================
                              ESCREVE UM CARACTER NO LCD
  =============================================================================*/
  void escreve_lcd(char c)
  // envia caractere para o display
  {
        envia_byte_lcd(1,c);
  }
```

```
/*=============================================================================
                                    LIMPA O LCD
=============================================================================*/
void limpa_lcd()
{
        envia_byte_lcd(0,0x01); // Envia instrução para limpar o LCD
        delay_ms(2); // Aguarda 2ms para estabilizar o LCD
        return; // Retorna ao ponto de chamada da função
}
/*=============================================================================
                                    INICIALIZA O LCD
=============================================================================*/
void inicializa_lcd()
{
        output_low(lcd_db4); // Garante que o pino DB4 estão em 0 (low)
        output_low(lcd_db5); // Garante que o pino DB5 estão em 0 (low)
        output_low(lcd_db6); // Garante que o pino DB6 estão em 0 (low)
        output_low(lcd_db7); // Garante que o pino DB7 estão em 0 (low)
        output_low(lcd_rs); // Garante que o pino RS estão em 0 (low)
        output_low(lcd_enable); // Garante que o pino ENABLE estão em 0 (low)
        delay_ms(15); // Aguarda 15ms para estabilizar o LCD
        envia_nibble_lcd(0x03); // Envia comando para inicializar o display
        delay_ms(5); // Aguarda 5ms para estabilizar o LCD
        envia_nibble_lcd(0x03); // Envia comando para inicializar o display
        delay_ms(5); // Aguarda 5ms para estabilizar o LCD
        envia_nibble_lcd(0x03); // Envia comando para inicializar o display
        delay_ms(5); // Aguarda 5ms para estabilizar o LCD
        envia_nibble_lcd(0x02); // CURSOR HOME - Envia comando para zerar o contador de
        // caracteres e retornar à posição inicial (0x80).
        delay_ms(1); // Aguarda 1ms para estabilizar o LCD
        envia_byte_lcd(0,0x28); // FUNCTION SET - Configura o LCD para 4 bits,
        // 2 linhas, fonte 5X7.
        envia_byte_lcd(0,0x0c); // DISPLAY CONTROL - Display ligado, sem cursor
        limpa_lcd(); // Limpa o LCD
        envia_byte_lcd(0,0x06); // ENTRY MODE SET - Desloca o cursor para a direita
        return; // Retorna ao ponto de chamada da função
}
//Fim das rotinas para utilização do LCD
```

Anexo 11 – Biblioteca 24AA1025

```
#define     ctrl_le_hi      0b10101001      // byte de controle da memória p/ leitura na
                                            //parte alta da memória
#define     ctrl_escr_hi    0b10101000      // byte de controle da memória p/ escrita na
                                            //parte alta da memória
#define     ctrl_le_low     0b10100001      // byte de controle da memória p/ leitura na
                                            //parte baixa da memória
#define     ctrl_escr_low   0b10100000      // byte de controle da memória p/ escrita na
                                            //parte baixa da memória

/*============================================================================
                        Rotina para escrita na EEPROM externa
 ============================================================================*/
void    write_eeprom_ext()
{
        delay_ms(10);
        end_low = make8 (endereco,0);       //Quebra enderco em HI e LOW
        end_hi  = make8 (endereco,1);
        #use    i2c(master,sda=pin_c4, scl=pin_c3, FAST, RESTART_WDT)
        disable_interrupts(GLOBAL);
        i2c_start();                                // Condição de início
        i2c_write(ctrl_escr);           // Envia o byte de controle de leitura
        i2c_write(end_hi);                  // Envia endereço baixo
        i2c_write(end_low);                 // Envia endereço baixo
        i2c_write(dado);                    // Escreve dado na E²PROM
        i2c_stop();                                     // Condição de parada
        delay_ms(10);                               // espera a gravação estar completa
        enable_interrupts(GLOBAL);
}

/*============================================================================
                        Rotina para leitura na EEPROM externa
 ============================================================================*/
void    read_eeprom_ext()
{
        end_low = make8 (endereco,0);//Quebra enderco em HI e LOW
        end_hi = make8 (endereco,1);
        #use    i2c(master,sda=pin_c4, scl=pin_c3, FAST, RESTART_WDT)
        disable_interrupts(GLOBAL);
        i2c_start();                                // Condição de início
        i2c_write(ctrl_escr);               // Envia o byte de controle de leitura
        i2c_write(end_hi);                  // Envia endereço baixo
        i2c_write(end_low);                 // Envia endereço baixo
        i2c_start();                                // Nova condição de início
        i2c_write(ctrl_le);                 // Envia o byte de controle de leitura
        dado = i2c_read(0);                 // lê o dado armazenado na E²PROM
        i2c_stop();                                     // Condição de parada
        enable_interrupts(GLOBAL);
}
```

Anexo 12 – Biblioteca DS1307

```
/*
vARIÁVEIS

int dia;
int mes;
int ano;
int hora;
int minuto;

BYTE sec;
BYTE min;
BYTE hrs;
BYTE day;
BYTE month;
BYTE yr;
BYTE dow;

EXEMPLO:

        ds1307_get_date(day,month,yr,dow);
        ds1307_get_time(hrs,min,sec);

        //DATA
        envia_byte_lcd(0,0xC1);
        printf(escreve_lcd,"\%02d/\%02d/\%02d ",day,month,yr);
        //HORA
        envia_byte_lcd(0,0xCA);
        printf(escreve_lcd,"\%02d:\%02d", hrs,min);

/*

void acerta_hora()
{
        ds1307_get_date(day,month,yr,dow);
        ds1307_get_time(hrs,min,sec);
        //DATA
        envia_byte_lcd(0,0xC1);
        printf(escreve_lcd,"\%02d/\%02d/\%02d ",day,month,yr);
        //HORA
        envia_byte_lcd(0,0xCA);
        printf(escreve_lcd,"\%02d:\%02d", hrs,min);
}

//AJUSTAR O RELÓGIO

ds1307_set_date_time(dia,mes,ano,2,hora,minuto,segundo);

*/

////////////////////////////////////////////////////////////////////////

#define RTC_SDA PIN_C4
#define RTC_SCL PIN_C3

#use    i2c(master, sda=RTC_SDA, scl=RTC_SCL)

BYTE bin2bcd(BYTE binary_value);
BYTE bcd2bin(BYTE bcd_value);

void ds1307_init(void)
{
BYTE seconds = 0;

i2c_start();
i2c_write(0xD0); // WR to RTC
i2c_write(0x00); // REG 0
i2c_start();
i2c_write(0xD1); // RD from RTC
```

```
seconds = bcd2bin(i2c_read(0)); // Read current "seconds" in DS1307
i2c_stop();
seconds &= 0x7F;

delay_us(3);

i2c_start();
i2c_write(0xD0); // WR to RTC
i2c_write(0x00); // REG 0
i2c_write(bin2bcd(seconds)); // Start oscillator with current "seconds value
i2c_start();
i2c_write(0xD0); // WR to RTC
i2c_write(0x07); // Control Register
i2c_write(0x80); // Disable squarewave output pin
i2c_stop();
}

void ds1307_set_date_time(BYTE day, BYTE mth, BYTE year, BYTE dow, BYTE hr, BYTE min, BYTE sec)
{
sec &= 0x7F;
hr &= 0x3F;

i2c_start();
i2c_write(0xD0); // I2C write address
i2c_write(0x00); // Start at REG 0 - Seconds
i2c_write(bin2bcd(sec)); // REG 0
i2c_write(bin2bcd(min)); // REG 1
i2c_write(bin2bcd(hr)); // REG 2
i2c_write(bin2bcd(dow)); // REG 3
i2c_write(bin2bcd(day)); // REG 4
i2c_write(bin2bcd(mth)); // REG 5
i2c_write(bin2bcd(year)); // REG 6
i2c_write(0x80); // REG 7 - Disable squarewave output pin
i2c_stop();
}

void ds1307_get_date(BYTE &day, BYTE &mth, BYTE &year, BYTE &dow)
{
i2c_start();
i2c_write(0xD0);
i2c_write(0x03); // Start at REG 3 - Day of week
i2c_start();
i2c_write(0xD1);
dow = bcd2bin(i2c_read() & 0x7f); // REG 3
day = bcd2bin(i2c_read() & 0x3f); // REG 4
mth = bcd2bin(i2c_read() & 0x1f); // REG 5
year = bcd2bin(i2c_read(0)); // REG 6
i2c_stop();
}

void ds1307_get_time(BYTE &hr, BYTE &min, BYTE &sec)
{
i2c_start();
i2c_write(0xD0);
i2c_write(0x00); // Start at REG 0 - Seconds
i2c_start();
i2c_write(0xD1);
sec = bcd2bin(i2c_read() & 0x7f);
min = bcd2bin(i2c_read() & 0x7f);
hr = bcd2bin(i2c_read(0) & 0x3f);
i2c_stop();
}

BYTE bin2bcd(BYTE binary_value)
{
BYTE temp;
BYTE retval;

temp = binary_value;
retval = 0;

while(1)
{
// Get the tens digit by doing multiple subtraction
// of 10 from the binary value.
if(temp >= 10)
```

```
{
temp -= 10;
retval += 0x10;
}
else // Get the ones digit by adding the remainder.
{
retval += temp;
break;
}
}
return(retval);
}

// Input range - 00 to 99.
BYTE bcd2bin(BYTE bcd_value)
{
BYTE temp;

temp = bcd_value;
// Shifting upper digit right by 1 is same as multiplying by 8.
temp >>= 1;
// Isolate the bits for the upper digit.
temp &= 0x78;

// Now return: (Tens * 8) + (Tens * 2) + Ones

return(temp + (temp >> 2) + (bcd_value & 0x0f));
}
```

Anexo 13 – Biblioteca Conversores

```
/*==========================================================================
                    CONVERSÃO PRESSÃO 0 a 50 mca
==========================================================================*/
void conversao_ad_press1()
{
        conversao = (conversao * 50);           // faz regra de 3 para converter o valor,
        conversao = (conversao / 255); // das unidades de AD em Volts.
}
/*==========================================================================
                    CONVERSÃO PRESSÃO 0 a 150 mca
==========================================================================*/
void conversao_ad_press2()
{
        conversao = (conversao * 150);          // faz regra de 3 para converter o valor,
        conversao = (conversao / 255); // das unidades de AD em Volts.
}
/*==========================================================================
                    CONVERSÃO TEMPERATURA 0 a 100 C
==========================================================================*/
void conversao_ad_temp()
{
        conversao = (conversao * 100);          // faz regra de 3 para converter o valor,
        conversao = (conversao / 255); // das unidades de AD em Volts.
}
/*==========================================================================
                     convrsão da Entrada Analógica
==========================================================================*/
void conversao_ad_tensao()
{
        conversao = (conversao * 10);           // faz regra de 3 para converter o valor,
        conversao = (conversao / 255); // das unidades de AD em Volts.
}
```