UNIVERSIDADE FEDERAL DE CAMPINA GRANDE

CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA

COORDENAÇÃO DE ESTÁGIOS E PROJETOS DE ENGENHARIA ELÉTRICA

RELATÓRIO DE ESTÁGIO SUPERVISIONADO

Título do Trabalho: Automação do Sistema de Entrada de Dados do Setor de Expedição

Trabalho Apresentado por: Ravi Agra Marques Matrícula 2031214

Empresa: Felinto Ind. e Com. Ltda

Período: 25/06/2008 a 25/08/2008

Orientador: Prof. Dr. Alexandre Cunha Oliveira

Campina Grande — Paraíba

Setembro/2008

Biblioteca Setorial do CDSA. Março de 2021.

Sumé - PB

Agradecimentos

A Alexandre Menezes Marques das Neves e Klauber Canuto, do Sistema Integrado de Gestão da Felinto, pelo direcionamento nas exigências do projeto, à diretoria da Felinto, por proporcionar a oportunidade de estágio e acreditar no projeto. Ao Engº. Paulo Roberto Campos de Araújo, pela orientação técnica e o apoio durante o estágio. Ao Prof. Dr. Alexandre Cunha Oliveira pela orientação e confiança. E à minha família pelo apoio durante todo o curso.

SUMÁRIO

1 Lista de Figuras e Tabelas

2 Lista de Abreviaturas

ASCII - American Standard Code for Information Interchange

CCD – Charge Coupled Device

CCS – Custom Computer Services

CRC - Cyclic Redundancy Check

FIFO – First In, First Out

LCD – Liquid Crystal Display

MCU – Microcontroller Unit

SPI – Serial Peripheral Interface

TTL – Transistor-Transistor Logic

UART - Universal Asynchronous Receiver/Transmitter

UPC – Universal Product Code

3 Resumo

O presente trabalho teve como objetivo automatizar o processo de inserção dos endereços de páletes alocados no setor da Expedição da empresa Felinto Ind. e Com. Ltda.

Para tal, foi desenvolvido um equipamento portátil, dotado de leitor de código de barras e interface com o usuário, com o qual se insere o endereço (posicionamento na estrutura de armazenagem) do pálete. Tal equipamento utiliza comunicação sem fio (via rádio) para passar esses dados para o sistema central, onde são salvos no banco de dados da empresa.

Tal sistema foi implantado e está trazendo benefícios para a empresa na forma de aumento da confiabilidade dos dados, redução do tempo utilizado pelos funcionários do setor para dar entrada nesses dados e permitindo o acesso e uso desses dados pelo restante da empresa de forma imediata.

4 Introdução

O presente trabalho foi desenvolvido no setor de Expedição (envio de produtos acabados) da Felinto Ind. e Com. Ltda, estabelecida na cidade de Campina Grande-PB, no período de 25 de Junho de 2008 a 25 de Agosto de 2008, totalizando 160 horas de atividades.

A Felinto é uma empresa local do ramo de embalagens flexíveis, no qual atua a mais de 25 anos. Nos últimos anos vem investindo fortemente tanto em expansão como em evolução tecnológica, ampliando sua capacidade produtiva e elevando o perfil de seus produtos.

Dentro desse contexto, no âmbito de empresa, vem-se fazendo diversos projetos de automação e de apoio à gestão, e nesse contexto se insere o presente trabalho.

No setor da Expedição faz-se a armazenagem provisória de páletes (estrado de madeira utilizado para movimentação de cargas) de produto acabado, que se destinam ao envio para os clientes, pelo uso de frota de caminhões. Para tal, dispõe-se de um armazém onde se empilham tais páletes, esse empilhamento é organizado por uma estrutura de endereçamento lógico, onde o galpão é dividido em áreas, cada área dividida em ruas, cada rua contendo algumas quadras (área ocupada por um pálete) e cada quadra tendo vários andares (estantes). Na Figura 1 é apresentada uma planta mostrando essa divisão lógica do setor.

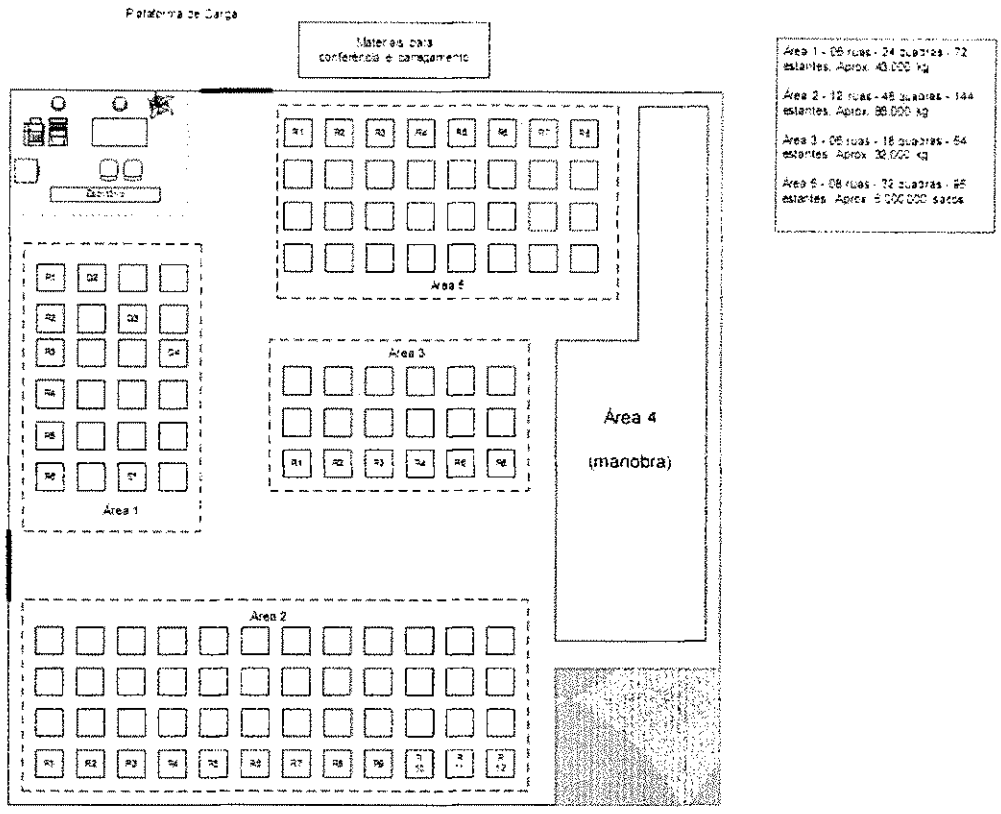


Figura 1 - Planta da Expedição com endereçamento

Antes da implantação do projeto desenvolvido, a inserção dos dados era feita de modo manual, onde os operadores de empilhadeira ou seus auxiliares anotavam a posição do pálete no romaneio (documento que contém informações do pálete, como seu número, produto, cliente e peso) e repassavam tal documento para o supervisor, que os digitava (normalmente apenas no dia seguinte) na tela de acesso ao sistema que gerencia o banco de dados.

Como analisado, tal procedimento implica em uma série de problemas e/ou inconvenientes, os quais foram abordados para justificar o projeto, apontando suas vantagens sobre a situação vigente.

Na forma final do projeto, foi empregada tecnologia de códigos de barra para a identificação do pálete, onde uma etiqueta, contendo um código de uso interno (o número de registro do pálete) é impressa no próprio romaneio (que é anexado fisicamente ao pálete) e, para a inserção do endereço, foi desenvolvido um módulo portátil com uma interface com o usuário que permite a digitação dos campos.

O sistema permite a localização em tempo real dos páletes, a partir do momento em que este é posicionado em seu local, isso elimina o atraso de um dia nas informações antes existente, já que previamente estas deviam ser inseridas manualmente no sistema pelo supervisor no dia seguinte.

Há um considerável aumento na segurança da localização dos páletes, evitando situações em que não se pode localizar de imediato um pálete, seja por extravio das anotações do operador antes destas serem registradas, ou por erros na entrada dos dados.

O sistema também desvincula a tarefa de entrada manual dos dados do supervisor, liberando horas de trabalho deste para efetuar outras tarefas. Por fim, o uso do sistema servirá de base para uma mudança no próprio procedimento do setor, permitindo a alocação dos páletes diretamente pelos empilhadores, com o supervisor trocando de papel, deixando de ter de determinar o endereço de cada pálete e passando a poder simplesmente conferir o devido posicionamento a qualquer instante.

5 Fundamentos Teóricos

5.1 Códigos de Barra:

Códigos de barra já se tornaram a solução ubíqua para identificação de produtos no ambiente comercial. Sua principal vantagem é o preço, além da simplicidade de aplicação, a facilidade de posicionamento da etiqueta em diversas superfícies e a possibilidade de se combinar a leitura por instrumento e por pessoas (adicionando o dado por extenso ao código em si).

Os códigos de barra foram primeiramente registrados na US Patent 2.612.994 (1), de 1952. Esta primeira tecnologia já usava o padrão linear (linhas verticais paralelas) para codificar o dado. Com o tempo, diversas outras formas foram desenvolvidas, incluindo padrões 2D (matrizes de pontos e formas geométricas) até a utilização de cores para aumentar a densidade de codificação.

A grande expansão da tecnologia se deu com o padrão UPC (2), usado até hoje em praticamente todas as embalagens destinadas ao consumidor final. Outros padrões são adotados normalmente em aplicações internas. Alguns mais comuns são o CODE-39 e o CODE-128. Na presente aplicação foi adotado o CODE-39 (3), visto que esse já era o padrão adotado no rastreamento interno da empresa.

Esse padrão codifica cada caractere por um conjunto de 5 barras com 4 espaços entre elas, cada barra ou espaço podendo ser largo ou estreito, informação esta que é traduzida num bit lógico. Em cada caractere, 3 desses 9 elementos têm de ser largos, daí o nome do padrão. Essa representação permite codificar 43 símbolos, usados como 26 letras (maiúsculas), os 10 dígitos numéricos e 7 caracteres simbólicos. Para dados mais complexos existe uma versão estendida que utiliza um caractere duplo.

Os dados utilizados no sistema implantado consistem dos 6 dígitos numéricos que identificam o pálete, um caractere R no início, que serve para que o programa do transmissor só valide etiquetas do próprio sistema (evitando leituras das etiquetas

comerciais presentes nas embalagens) e o valor do CRC no final. Na Figura 2 temos um exemplo de uma etiqueta usando o padrão CODE-39.

Figura 2 - Exemplo de código de barra

Para a leitura dos códigos de barra temos aparelhos que utilizam feixes de luz para varrer o comprimento da etiqueta com o código. Para tal, se utiliza uma fonte de luz, uma lente e um foto-detector, que converte o sinal luminoso em elétrico. Adicionalmente, quase todos os leitores comerciais embutem os decodificadores eletrônicos que convertem os dados para ASCII.

Há, basicamente, três tipos de leitor: a caneta, que contém apenas uma fonte de luz e um foto-diodo, o que requer que o operador mova-a de forma relativamente constante sobre a etiqueta; os leitores ativos, que usam um feixe de laser para varrer a etiqueta horizontalmente, e que permitem o seu uso a grandes distâncias, e os leitores passivos, como o adotado no projeto, que utilizam um flash para iluminar toda a etiqueta, em vez de varrê-la com um feixe, como nos ativos, e capturam a luz refletida através de um array de CCD's, alguns desses CCD's irão receber luz (refletida nos espaços), e outros não, o que codifica as barras (4).

5.2 Comunicação *Wireless* ZigBee

O ZigBee (IEEE 812.15.4) é uma especificação para comunicação digital de alto nível via rádio frequência. É utilizada para sistemas com baixo consumo e alta confiabilidade. Opera na banda ISM, mais especificamente na frequência de 2,4 GHz para a presente aplicação. (5)

Na presente aplicação foi utilizada uma funcionalidade específica do dispositivo empregado, o modo de comunicação ShockBurst, onde a MCU (*Micro-controller Unit*) transfere os dados para um *buffer* no rádio à taxa mais adequada, e ao final, sinaliza para

que este os envie, envio este que é feito a uma taxa de 250 kbps ou 1Mbps, de forma condensada, minimizando o consumo de corrente elétrica e a possibilidade de interferências por outras fontes eletromagnéticas presentes na faixa de espectro utilizada.

Esse modo de comunicação utiliza uma pilha (estrutura FIFO) interna ao *chip* do rádio, aonde os dados recebidos da MCU são escritos, utilizando um *clock* gerado pela própria MCU, e na frequência conveniente à mesma (da ordem de alguns kbps). Quando a MCU possui dados a serem enviados, ativa um pino do rádio (o CE), o que aciona o processamento interno do rádio, então os dados são passados. Ao final, tendo recebido o número de bits de dados configurado, e ao ser desativado o pino CE, o *transceiver* realiza a transmissão desses dados, esvaziando a pilha. O *transceiver* também se encarrega de adicionar um preâmbulo e o CRC (*Cyclic Redundancy Check*) aos dados, o que minimiza ainda mais o esforço computacional da MCU.

Na Figura 3 é apresentado um esquema de como esse procedimento é realizado:

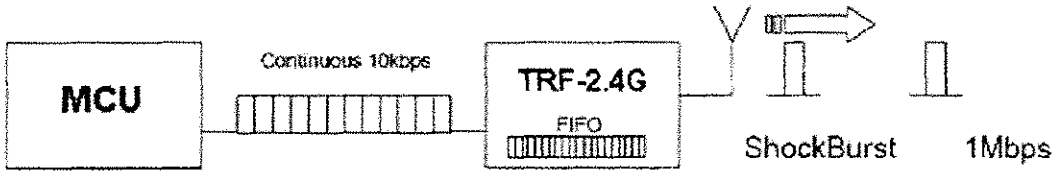


Figura 3 - Modo de transmissão *ShockBurst*

Na recepção, o *transceiver* já se encarrega de eliminar do pacote de dados o preâmbulo e o CRC (desde que este esteja correto) e repassa apenas os dados devidamente tratados à MCU, utilizando novamente para essa comunicação o clock fornecido pela própria MCU, o que evita falha de sincronismo ou perda de bits.

6 Materiais e Métodos

O presente projeto envolveu o desenvolvimento de um equipamento de inserção de dados para interface com usuário, bem como a elaboração de *software* para tal equipamento, na forma de um *firmware* (aplicativo embarcado num dispositivo) para o microcontrolador utilizado – PIC16F877A, e o desenvolvimento do aplicativo para CP (Computador Pessoal) que recebe os dados e os armazena em disco para permitir o acesso pelo banco de dados da empresa. Para o recebimento desses dados, também foi desenvolvido o receptor de rádio que gerencia a comunicação com os módulos remotos e formata os dados obtidos, enviando-os por intermédio de uma porta serial (RS-232) (6) para o CP.

No sistema final foi adotada a solução com Códigos de Barra, houve, no entanto, pesquisa mais abrangente no início das atividades, incluindo um levantamento das tecnologias RFID comerciais. Tal levantamento culminou na recomendação técnica para o uso de etiquetas passivas de apenas leitura, juntamente com a instalação de portais de leitura Geração 2 (7) nos portões de entrada e saída do galpão. Tal solução, contudo, foi rejeitada por critérios econômicos e a opção por leitura de Códigos de Barra foi feita, o que era altamente econômico para a empresa, pelo fato desta já utilizar rastreamento interno por código de barra, e, portanto, já possuir os equipamentos e *softwares* necessários para a geração de seus códigos de barra.

Descreveremos, agora, os recursos utilizados, tanto de *hardware* como de *software* para a realização do presente projeto, como em sua versão final.

6.1 *Hardware*

6.1.1 Leitor de Código de Barra

Como elemento primário de entrada de dados no sistema, a escolha dos equipamentos a serem adotados partiu da escolha do leitor de códigos de barra. Os requisitos para tal escolha foram:

- Portabilidade, já que o projeto previa que a entrada dos dados seria feita *in loco*, à medida que os páletes fossem sendo manipulados;
- Baixo consumo de energia elétrica, por ser alimentado via bateria.
- Facilidade de comunicação com um microcontrolador: este foi o fator decisivo na escolha do modelo adotado, pois a grande maioria dos modelos existentes no mercado utiliza a técnica de substituição do teclado, utilizando o conector PS/2 e usando o mesmo protocolo de comunicação usado em teclados para CP's (8). O modelo adotado, no entanto, utiliza comunicação RS-232, através de um conector DB-9, o que facilita consideravelmente a comunicação com o microcontrolador.

Atendendo a tais requisitos, e acrescentando-se a viabilidade econômica, optou-se portanto pelo Leitor de Códigos de Barra CCD Bematech BR 310 (9).

Na Figura 4 temos uma foto desse modelo.

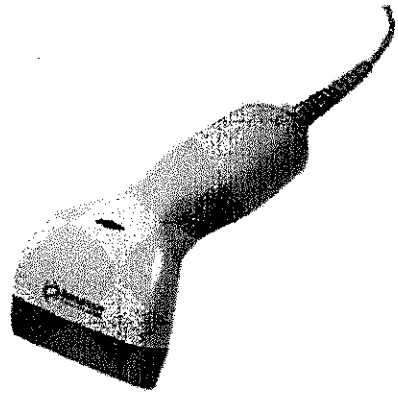


Figura 4 - Leitor de código de barra Bematech BR-310

6.1.2 *Transceiver Wireless*

Na definição do elemento de comunicação sem fio, a escolha partiu imediatamente para o uso de rádios, já que outras tecnologias, como Infra-Vermelho, dependem de linha de visada direta, o que é inviável dada a disposição do ambiente.

Feita essa escolha, passamos à decisão do dispositivo em si. Os requisitos levados em conta nessa seleção foram:

- Alcance: o rádio utilizado deveria ter um alcance de cerca de 50 m, para que cobrisse uma área razoável do galpão;
- Imunidade a interferências: devido a possível presença de equipamentos industriais, o ambiente pode apresentar ruídos, por isso foi escolhida a frequência de trabalho de 2,4 GHz, notadamente imune a ruídos;
- Encapsulamento: o elemento escolhido deveria ser capaz de se comunicar em alto nível com o micro-controlador, evitando a execução anterior de atividades de baixo nível, como a modulação do sinal;
- Comunicação *transceiver*: ou seja, permitir a comunicação em ambos os sentidos num único dispositivo, necessário para se fazer um protocolo de comunicação mais seguro;
- Baixo consumo: pois se trata de aplicação remota;

O dispositivo adotado foi o Transceiver TRF-2,4G da Laipac Tech, que possui antena embutida e capacidade de modulação e geração de CRC internas, o que facilita sua interação com a MCU. Este dispositivo permite o uso do modo ShockBurst, que permite uma taxa de transmissão de até 1 Mbps. Como a maioria dos equipamentos ZigBee, opera com potência de saída de 0dBm (1mW), podendo hibernar. O consumo total é menor que 90 mW. (10)

Esse dispositivo se comunica com o microcontrolador pelo protocolo SPI (11), possuindo endereçamento dinâmico e é totalmente configurável em tempo de execução pelo micro-controlador.

Na Figura 5 temos uma fotografia desse dispositivo.

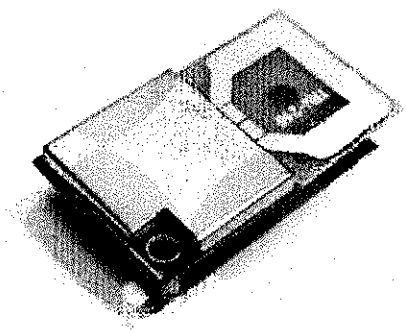


Figura 5 - *Transceiver* TRF 2,4G

Esse dispositivo apresentou um desempenho relativamente insatisfatório no quesito alcance, devido principalmente à presença de estruturas metálicas (os *racks* dos páletes) no galpão. Nos testes de aplicação, a zona de comunicação confiável estabelecida entre os módulos remotos e a base ficou restrita a cerca de 15 m. Isso sugere uma posterior modificação do projeto, o que já foi proposto à empresa para avanços futuros. Para tal modificação, uma possível solução é o Módulo XBee-Pro, com alcance em ambientes fechados de mais de 100 m (12). Tal dispositivo já foi testado no local e fornece um alcance adequado à utilização geral.

Tal modificação, contudo, só está prevista para os próximos setores, o que exigiu que a limitação anterior fosse contornada, no presente caso, através de modificações no protocolo de comunicação. Foi incluída a checagem de enlace, com a notificação para o usuário de quando a comunicação com a base foi estabelecida, permitindo, dessa forma, que ele envie os dados apenas quando adentra a zona de comunicação. Para isso, foi necessário também utilizar um *buffer* nos módulos, onde fosse possível armazenar as informações inseridas nas zonas sem comunicação. Com essas modificações implantadas, o sistema final atendeu as expectativas e pôde ser ativado sem mais problemas.

6.1.3 Interface com o operador

Para o desenvolvimento de uma interface com o operador, há a necessidade de um display, e de uma forma de entrada, visto que na presente forma do sistema, o endereço do pálete é informado pelo operador.

Para o display, foi escolhido o LM016L (13), um LCD de 16x2 caracteres com backlight verde, esse dispositivo é de fácil integração com o micro-controlador adotado (já existindo inclusive bibliotecas de código para ele), bem como de fácil incorporação ao projeto da placa de circuito impresso.

Para a entrada, foi adotado um modelo particular de teclado de membrana, desenvolvido pela Ylleus – Serigrafias LTDA., que possui 16 teclas, incluindo as 10 numéricas, o ponto decimal, e 5 teclas de função. Tal teclado tem longa vida útil, é flexível e auto-adesivo, permitindo sua fixação sobre a superfície do módulo. Sua operação é similar a um teclado de matriz de botões, operando com contato seco. Sua ligação é feita via um cabo flat conectado a uma barra de 8 pinos na placa de circuito impresso.

O esquema desse teclado é mostrado na Figura 6.

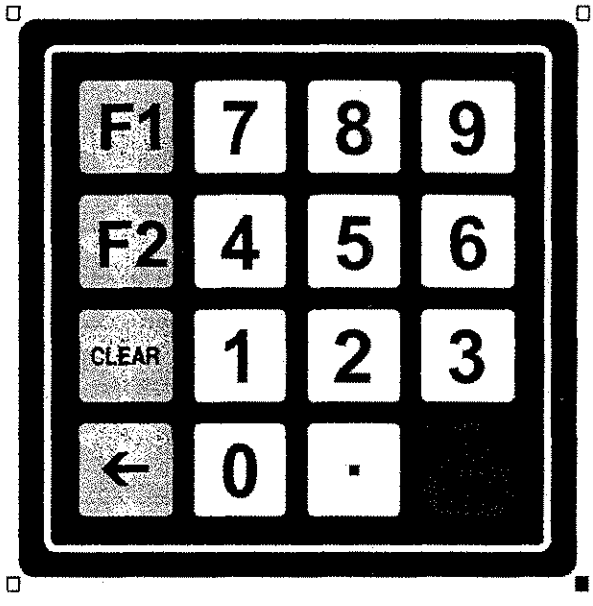


Figura 6 - Teclado empregado

6.1.4 Unidade de Processamento (Micro-controlador)

Para a escolha da unidade micro-controladora, alguns aspectos foram considerados:

- Facilidade de programação e desenvolvimento: a plataforma adotada deveria ter recursos de programação disponíveis e de fácil domínio. No caso a plataforma de

desenvolvimento CCS PCW fornece amplos recursos numa linguagem amplamente estudada, C, com diversas bibliotecas e apoio para o desenvolvimento voltado para os micro-controladores da família PIC.

- Recursos de comunicação: para a comunicação com o leitor de códigos de barra, a MCU (Unidade Micro-controlada) precisa ter uma porta UART (14), e para a comunicação com o módulo de rádio adotado, suporte ao padrão SPI.
- Número de pinos razoável: para a comunicação com os diversos dispositivos e outros usos internos da MCU, foi necessário um total de 31 pinos, então esta MCU deveria necessariamente ter ao menos essa quantidade de pinos;

Levando-se em conta estes aspectos, o micro-controlador adotado foi o PIC 16F877A, de 40 pinos, alimentado em 5V e utilizando um *clock* de 4 MHz.

6.1.5 Desenvolvimento da Placa de Circuito Impresso

Para o desenvolvimento da placa de circuito impresso a ser utilizada para a montagem dos módulos de entrada de dados remota, foram usados diversos sub-circuitos, os quais apresentamos abaixo separadamente, por questão de facilidade de compreensão e separação funcional:

6.1.5.1 Alimentação:

Para a alimentação da placa foi utilizado um circuito regulador de tensão, que toma os 12 V advindos da bateria (no caso dos módulos remotos) ou uma tensão qualquer na faixa 7 – 18V advinda da fonte regulada no caso do receptor base e fornece a tensão de 5V.

O componente empregado foi o regulador de tensão 7805 com capacitores na entrada e na saída para minimizar flutuações na tensão. Esse estabelece a tensão de 5 V. Como há também necessidade de dispor de uma tensão de 3.3 V para o elemento de rádio, fez-se a opção por usar o recurso simples de adicionar 3 diodos em série, o que produz uma

tensão de cerca de 3 V, suficiente para o rádio (10). Essa opção evita a inserção de novos reguladores e minimiza o uso de componentes. O circuito completo é apresentado na Figura 7:

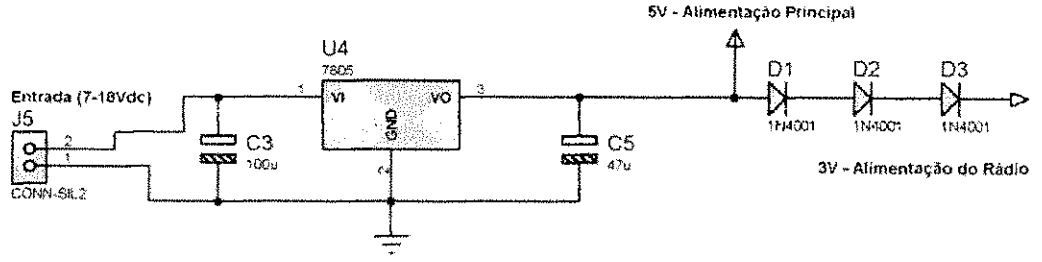


Figura 7 – Alimentação

6.1.5.2 Comunicação Serial RS-232

O microcontrolador utilizado já possui uma UART que realiza todo o protocolo de comunicação serial, mas este utiliza sinais em nível TTL. Como tanto o CP que se comunica com a estação base, como os leitores de código de barra, que se comunicam com os módulos remotos, utilizam o padrão RS-232, há a necessidade de utilizar um circuito de conversão. Tal circuito utilizado foi o MAX232, conforme apresentado na Figura 8.

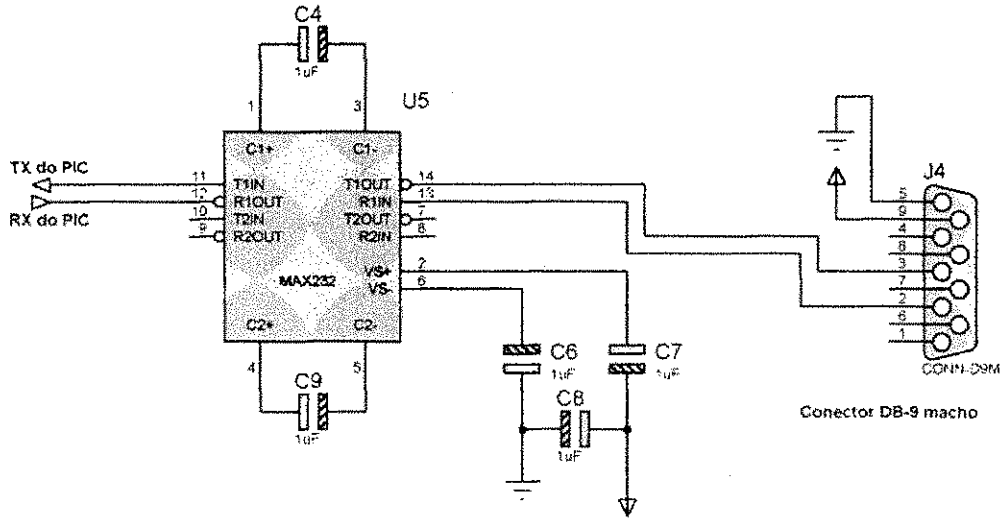


Figura 8 - Conversão TTL - RS232

6.1.5.3 Comunicação Wireless

Para a comunicação entre a MCU e o *transceiver* 2,4 GHz, precisamos de algum condicionamento. Primeiramente, pelo fato do rádio utilizar 3V e o PIC, 5V, precisamos de um *buffer* intermediário que converta os níveis lógicos de um para os do outro. Para tal fim foi utilizado o 7407 Hex-Buffer. Além disso, como o pino de dados do *transceiver* é bidirecional, foi utilizado um *latch* para coordenar a direção dos dados, habilitando apenas o sentido determinado pela MCU. Para tal, foi utilizado o 74LS244.

Os pinos de comunicação/controle do elemento de rádio são:

- WR: habilita (no 74LS244), a passagem de dados da MCU para o *transceiver*, é nível baixo ativo;
- RD: habilita (no 74LS244), a passagem de dados do *transceiver* para a MCU, é nível baixo ativo;
- Din: pino de recepção dos dados no PIC;
- Dout: pino de envio dos dados no PIC;
- CLK: *clock* utilizado pelo PIC para sincronizar a transmissão/recepção dos dados;
- CS: *Chip Select*, ativa o modo de configuração;
- CE: *Chip Enable*, seleciona o modo de operação RX/TX;

O circuito completo da ligação entre a MCU e o elemento de rádio é apresentado na Figura 9.

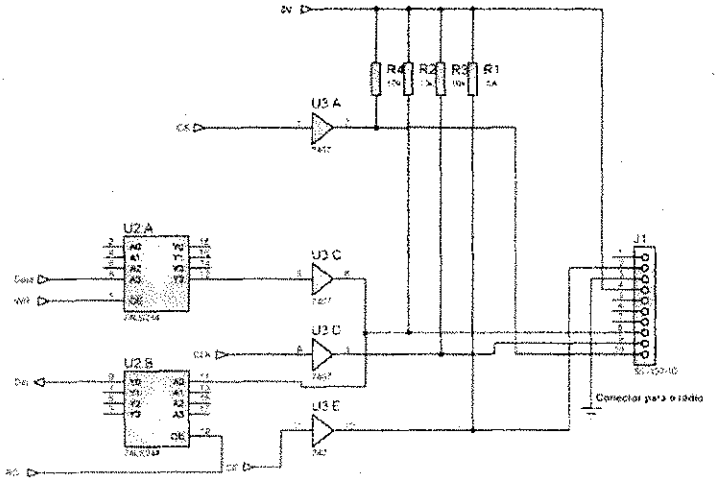


Figura 9 - Condicionamento dos sinais do rádio

6.1.5.4 Interface e MCU

A ligação com os elementos de interface com o usuário (teclado e display) é mais direta, requerendo apenas resistores de *pull-up* para o teclado e um divisor de tensão resistivo para o ajuste do contraste do display LCD.

Para o correto funcionamento do microcontrolador empregado é necessário, além de sua alimentação, que este possua o cristal oscilador devidamente conectado, no caso, o cristal utilizado foi de 4 MHz, bem como ter seu pino de *Master Clear* (*reset*) devidamente polarizado, para evitar *resets* não-intencionais.

Na Figura 10 são apresentados esses sub-circuitos.

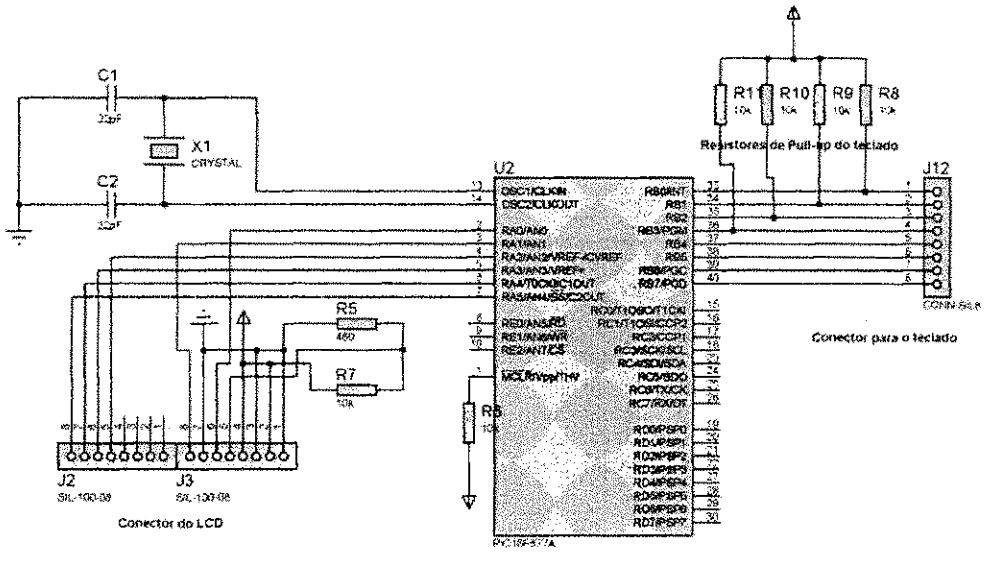


Figura 10 - Teclado, LCD, e MCU

No Apêndice A é apresentado o circuito completo e no Apêndice B o layout da placa confeccionada.

6.1.6 Caixa de acomodação

Conforme os requisitos de área determinados pela placa de circuito impresso, e levando em conta que uma caixa para a acomodação do módulo de interface necessita dos seguintes aspectos:

- Ser constituída de material plástico: por facilitar os cortes e perfurações necessários à disposição das conexões externas e por não interferir na transmissão do sinal de rádio;
- Ter tampa plana e lisa para a aplicação do teclado auto-adesivo;
- Ser leve e portátil, por precisar ser carregada pelo operador;
- Ser robusta o suficiente para uso prolongado em condições reais;

Foi feita a escolha, atendendo a todas essas especificações, pelo modelo PB-114/2 da Patola (15). Uma ilustração dessa caixa, com suas dimensões, é apresentada na Figura 11.

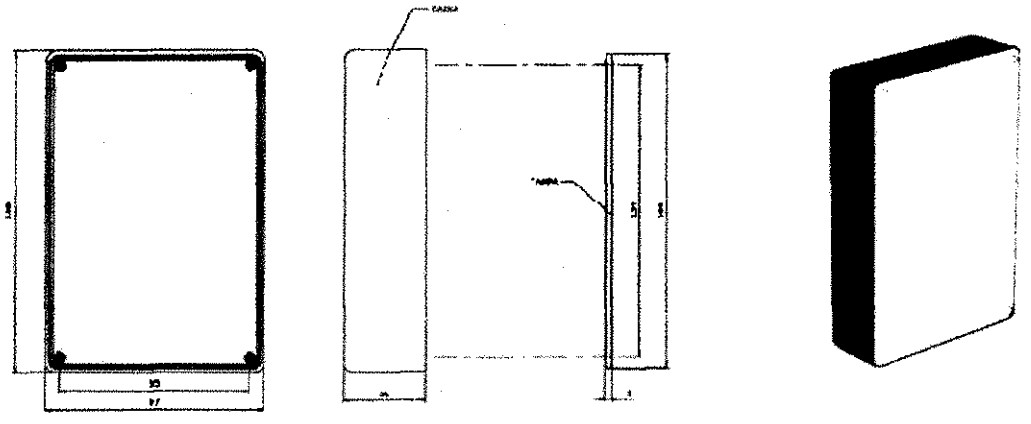


Figura 11 - Ilustração da caixa de acomodação empregada

6.1.7 Fonte de alimentação (Bateria)

Para a alimentação dos módulos remotos se fez necessária a escolha de uma bateria, pois requer-se a autonomia móvel, por se tratar justamente de utilizar o benefício de mobilidade aferido pela comunicação *wireless*.

Para a escolha dessa bateria, tivemos de adotar uma solução de compromisso entre os dois principais fatores: a carga útil, que define a autonomia de uso, e o peso da bateria, pois esta deveria ser portada pelo operador.

Com um consumo medido de cerca de 230 mA, o módulo completo (incluindo o leitor de códigos de barra) requer uma bateria de pelo menos 2 Ah para um ciclo de funcionamento/carregamento satisfatório.

Outro fator determinante é que a bateria escolhida deve ser do tipo recarregável, com amplo tempo de vida. Os testes iniciais, ainda em bancada, foram feitos utilizando baterias de Ni-MH de 9 V com 200 mAh, o que, obviamente, não tem aplicação no campo real.

A bateria finalmente empregada, foi a de Chumbo-Ácido, de 12 V, com 4Ah, o que atende plenamente a exigência de autonomia. Como carregador, foi utilizado um modelo de 800 mA, com flutuação, ou seja, a suspensão da carga uma vez que a bateria se encontre carregada. Tal carregador permite a carga completa da bateria empregada em cerca de 5 h. Levando-se em conta que a mesma tem um tempo de trabalho superior a 16 h, temos uma excelente relação tempo de uso/tempo de carga, o que permite que esta seja utilizada por toda a jornada diária (7h às 22h) e recarregada apenas durante a noite. Tal bateria, contudo, tem um grande volume (105x91x75 mm) e vem se mostrando, no seu presente uso, insatisfatória para os operadores.

Na Figura 12 é apresentada uma foto dessa bateria:

Figura 12 - Bateria adotada

Uma sugestão apresentada, e em vias de aprovação, é a utilização de baterias de Íon de Lítio, cujo modelo apresentado é de 11,1 V, com 1900 mAh. Esse modelo reduz a menos da metade a autonomia, podendo vir a ser necessária sua recarga durante o expediente (exigiria o uso de baterias de reserva), mas possui um volume quase 10 vezes menor (98x38x20 mm) do que o da de chumbo-ácido. A necessidade de recarga poderia ser amenizada mediante o desligamento do módulo após a transmissão de cada conjunto de dados.

Na Figura 13 é apresentada a foto da bateria proposta:

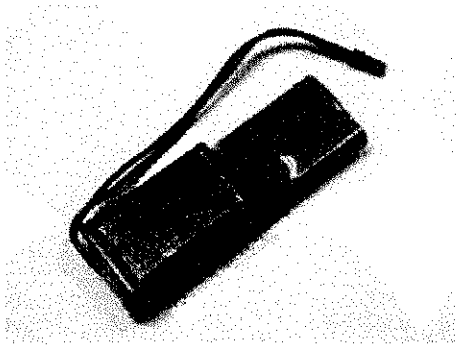


Figura 13 - Bateria alternativa proposta

6.2 *Software*

6.2.1 Transmissores

Descrevemos agora o código desenvolvido para o PIC16F877A para os módulos remotos (entradas de dados). Para isso, dividimos o código e apresentamos suas várias partes funcionais comentadas e explanadas.

6.2.1.1 Configuração Geral

Definição dos *fuses* (parâmetros de configuração de *hardware* da MCU), a maioria foi deixada como o *default* na versão do compilador utilizada (CCS 4.038). Como modificações, tivemos a escolha do tipo de oscilador, cristal de 4 MHz, no nosso caso, e a utilização do

Power Up Timer, que faz com que o código propriamente dito só se inicie após alguns instantes da energização do dispositivo.

Estas configurações são feitas nas linhas de 1 a 14 do código que pode ser visto no Apêndice C.

Adicionalmente, temos as configurações que são executadas no início do programa, as quais foram quase todas mantidas como o *default*, à exceção da configuração da comunicação SPI, que é utilizada para a interface com o *transceiver*. Estas configurações adicionais estão nas linhas 66 a 75 do código em apêndice.

6.2.1.2 Configuração da UART (Serial)

Para a comunicação serial foi adotado o esquema com um *Baud Rate* de 9600 bits/s, sem paridade, com uma palavra de dados de 8 bits, e sem controle de fluxo.

Para configurar tais parâmetros no dispositivo, é utilizado o seguinte comando:

```
#use rs232(baud=9600,parity=N,xmit=PIN_D0,rcv=PIN_D1,bits=8)
```

A comunicação serial é utilizada para a recepção dos dados do leitor, para o tratamento desses dados, foi escrita a função *le_barra()*, cujo fluxograma encontra-se representado na Figura 14 e cujo código encontra-se nas linhas 288 a 295 do Apêndice C.

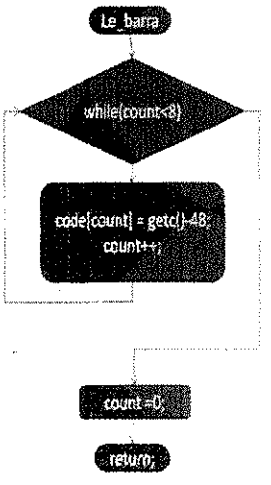


Figura 14 - Fluxograma da função le_barra()

Esta função recebe os dados da serial (através da função getc()) e os armazena numa variável de 8 caracteres (code[]), sendo esses 8 caracteres correspondentes ao caractere inicial 'R', os 6 dígitos de identificação do pálete e o CRC.

6.2.1.3 Conjunto de instruções para utilização do teclado

O teclado utilizado possui 16 teclas, organizadas em 4 colunas por 4 linhas, para efetuar a leitura, precisa-se de 8 pinos disponíveis do controlador. Para facilitar o entendimento, tais pinos foram definidos com seus respectivos identificadores, listados nas linhas 17 a 24 do Apêndice C.

O valor definido como *latch* (ver linha 26 do Apêndice C) se refere ao tempo de resposta das teclas, e é dado em milissegundos, isso significa que um toque único numa tecla é caracterizado por uma retenção de menos de 300 ms, caso o operador mantenha essa tecla pressionada por mais do que o tempo definido em *latch*, o programa interpreta essa nova informação como um novo pressionar da tecla. Esse valor foi testado e é o que melhor se ajusta ao digitar normal do usuário, tornando o procedimento mais natural.

Para a leitura propriamente dita da tecla, utiliza-se o seguinte método: escreve-se um *nibble(4 bits)* nas linhas do teclado (pinos L1 a L4), contendo 3 bits em um nível (0 ou 1) e 1 bit no outro nível. No nosso caso, o nível ativo escolhido foi o 0. Faz-se esse bit percorrer ciclicamente as 4 linhas. Enquanto isso, monitora-se o estado dos pinos correspondentes às colunas do teclado. Caso uma tecla esteja pressionada, o contato é feito entre a trilha de sua linha com a de sua coluna. Dessa forma, se há uma tecla pressionada, esta será identificada na leitura das colunas, associando-se o bit ativo das colunas com o que se encontra ativo nas linhas, tem-se a posição da tecla pressionada. Para uma melhor visualização, a Figura 15 ilustra o funcionamento desse tipo de teclado.

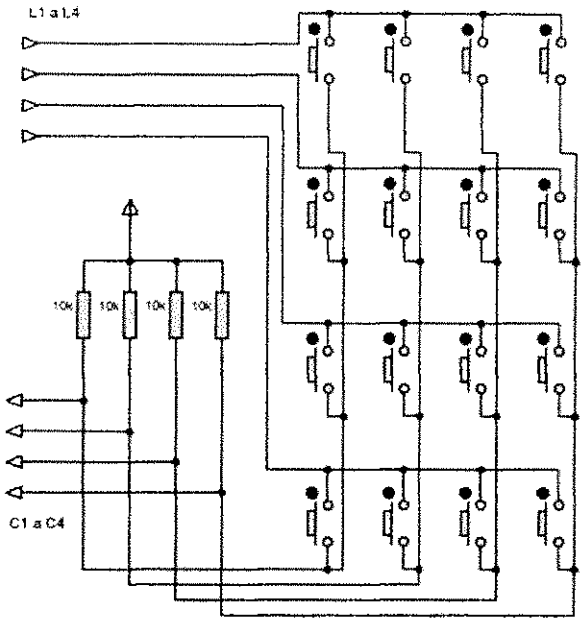


Figura 15 - Esquema elétrico do teclado matricial

Esse método é implementado pela função get_key, cujo fluxograma encontra-se representado na Figura 16 e código nas linhas 297 a 359.

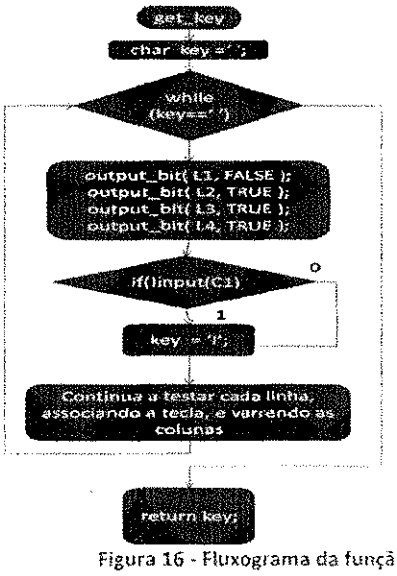


Figura 16 - Fluxograma da função get_key()

6.2.1.4 Configuração do Transceiver

Para a configuração do TRF 2,4G, utiliza-se uma palavra de configuração de 120 bits, essa palavra deve ser enviada ao rádio sempre que se deseje alterar algum parâmetro de

seu funcionamento, inclusive no início da operação para garantir que este opere da forma desejada.

Esse dispositivo possui dois canais de comunicação, cada um com frequência configurável (faixas de 1 MHz cada a partir da frequência base de 2,4 GHz), além de endereços e comprimentos de dados independentes. No nosso sistema, utilizamos apenas um desses canais, na faixa de 2430 MHz.

Para um funcionamento correto, tanto o transmissor como o receptor devem estar operando no mesmo canal, à mesma frequência e com o mesmo comprimento da palavra de dados. Pela exigência do tamanho dos dados que são enviados (operação, nº do pálete, e seu endereço), o comprimento da palavra de dados adotado foi fixado em 8 bytes.

Como o modo de operação (Rx/Tx) é configurado já nessa etapa, duas palavras completas foram utilizadas, uma para configurar o rádio como transmissor e outra para configurá-lo como receptor, em ambas os demais parâmetros são os mesmos.

Abaixo temos a declaração dessas variáveis, e a seguir, um descritivo completo de cada bit que as compõem:

```
unsigned char string_config[]={0xC8,0x40,0x00,0x00,0x00,0x00,0xc8,0x00,0x00,0x00,0x55,0xAA,0x43,0x4F,0x3C};  //Modo Transmissor
unsigned char string_config2[]={0xC8,0x40,0x00,0x00,0x00,0x00,0xc8,0x00,0x00,0x00,0x55,0xAA,0x43,0x4F,0x3D}; //Modo Receptor
```

A função de cada um dos bits é descrita logo em seguida, de acordo com a Tabela 1, obtida em ***(10)***.

	Bit position	Number of bits	Name	Function
ShockBurst configuration	143:120	24	TEST	Reserved for testing
	119:112	8	DATA2_W	Length of data payload section RX channel 1
	111:104	8	DATA1_W	Length of data payload section RX channel 1
	103:64	40	ADDR2	Up to 5 bytes address for channel 2
	63:24	40	ADDR1	Up to 5 bytes address for channel 1
	23:18	6	ADDR_W	Number of address bits(both RX channels)
	17	1	CRC_L	8 or 16 bits CRC
	16	1	CRC_EN	Enable on-chip CRC generation/checking
General device configuration	15	1	RX2_EN	Enable two channel receive mode
	14	1	CM	Communication mode (Direct or ShockBurst)
	13	1	RFDR_SB	RF data rate (1Mbps requires 16MHz crystal)
	12:10	3	XO_F	Crystal frequency (Factory default 16MHz crystal mounted)
	9:8	2	RF_PWR	RF output power
	7:1	7	RF_CH#	Frequency channel
	0	1	RXEN	RX or TX operation

Tabela 1 - Disposição da palavra de configuração dos *Transceivers*

Nessas palavras, os 16 primeiros bits indicam os comprimentos das palavras de dados, sendo os 8 primeiros para o canal 2, e os 8 seguintes para o canal 1. Usamos apenas o 1, e para este, adotamos uma palavra de 8 bytes (64 bits, 0x40).

Os próximos 80 bits são os endereços para os dois canais, os 40 primeiros para o canal 2, e os 40 seguintes para o canal 1, novamente, usamos apenas o 1, e para este, adotamos o endereço de 0x55AA ou 21.930, tal escolha minimiza a possibilidade de um endereçamento ser falsamente interpretado por interferência, já que um padrão alternado de bits 0 e 1 (como em 55 e AA) dificilmente será reproduzido num ruído aleatório, bem como, uma interferência é facilmente distinguível, do contrário do que seria se tivéssemos vários bits idênticos em sequência.

Os próximos 6 bits indicam a largura do campo de endereço, no nosso caso usamos endereço com 32 bits. Os próximos 2 bits configuram o CRC, o primeiro selecionando o CRC com 16 bits e o segundo habilitando a geração do CRC no próprio chip do rádio.

O próximo bit (primeiro bit do trecho 0x6F) seleciona o modo de recepção em apenas um canal. O seguinte habilita o modo *ShockBurst* (em contraposição ao modo de transmissão direta). O próximo define a taxa de transmissão, utilizamos a taxa de 250 kbps, com o intuito de maximizar o alcance da comunicação, visto que, conforme explicitado em (10), a utilização dessa taxa aumenta em até 10 dB a sensibilidade do receptor em relação a utilizando 1 Mbps.

Os próximos 3 bits definem a frequência do cristal usado internamente no *chip* do rádio, deixamos como default, (bits 011), o que corresponde a uma frequência de 16 MHz.

Os 2 bits seguintes definem a potência do sinal enviado, adotamos a maior possível, que para este dispositivo é de 0 dBm (1 mW), visando novamente maximizar o alcance.

No último byte, configuramos a frequência do canal (7 primeiros bits), que no nosso caso, foi adotada em 2430 MHz. Finalmente, o último bit, conforme o comentário, define a direção da comunicação, habilitando a recepção com 1 ou desabilitando-a com 0, ou seja, habilitando a transmissão.

A passagem dessa palavra de configuração para o rádio é feita pela função config, que primeiro coloca o rádio em modo de configuração, através da ativação correta dos pinos CS e CE, de acordo com a Tabela 2.

Mode	CE	CS
Active (RX /TX)	1	0
Configuration	0	1
Stand by	0	0

Tabela 2 - Funções dos pinos de configuração

A função config também habilita o latch 74LS244 a passar os dados para o rádio (bloqueando o sentido inverso) e só então, através da função de escrita spi_write(), escreve a devida palavra de configuração no rádio. Após a configuração, o rádio é retornado ao modo de recepção. O fluxograma dessa função é apresentado na Figura 17, e seu código nas linhas 249 a 272 do Apêndice C.

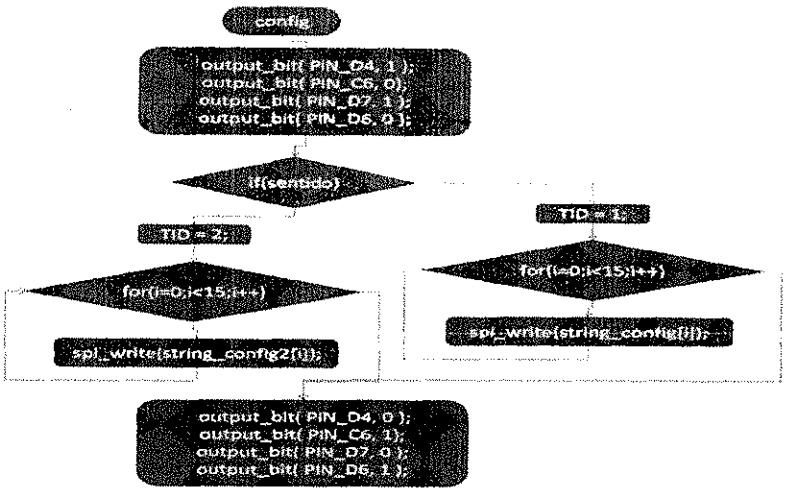


Figura 17 - Fluxograma da função config()

Nesse fluxograma, aparecem os pinos do PIC conforme sua nomenclatura interna, a função desses pinos na interface é a seguinte:

- PIN_D4: Sinal CS;
- PIN_C6: Sinal CE;
- PIN_D7: Sinal RD (habilita passagem de dados do rádio para o PIC);
- PIN_D6: Sinal WR (habilita passagem de dados do PIC para o rádio);

6.2.1.5 Comunicação Wireless

A comunicação com o *transceiver*, depois que este se encontra configurado, é feita por duas funções, semelhantes à *config*, uma de transmissão, a *envia()*, que configura o rádio como transmissor (configurando os pinos CE e CS de acordo) e habilita o modo de escrita no 74LS244 para então escrever os dados a serem enviados pela função *spi_write()*, a mesma usada para escrever a palavra de configuração. Ao sair, essa função retorna o estado dos pinos ao anterior, tornando o estado default do módulo o de recepção.

Já a de recepção, sendo o estado *default* do módulo receptor, precisa apenas fazer a leitura dos dados chegando do rádio e armazená-los, faz isso lendo 8 bytes (o comprimento da palavra de dados) da interface SPI, através da função *spi_read()*. Em adição a isso, a função de recebimento, *handle()*, já checa se os dados recebidos são os esperados advindos

da base, mediante checagem do primeiro byte recebido (que deve ser 0x11, ou 17, pois esse é o valor que a estação base emite como sinal de sua presença), usando isso para determinar se há conexão presente. Os fluxogramas de ambas encontram-se representados nas Figuras 18 e 19, e seus códigos nas linhas 52 a 64, para a de recepção, e 336 a 356 para a de transmissão.

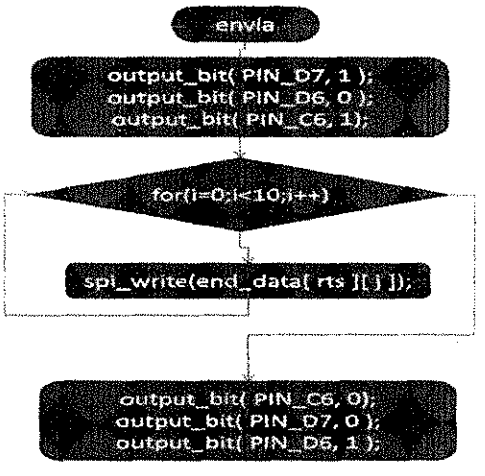


Figura 18 - Fluxograma da função envia()

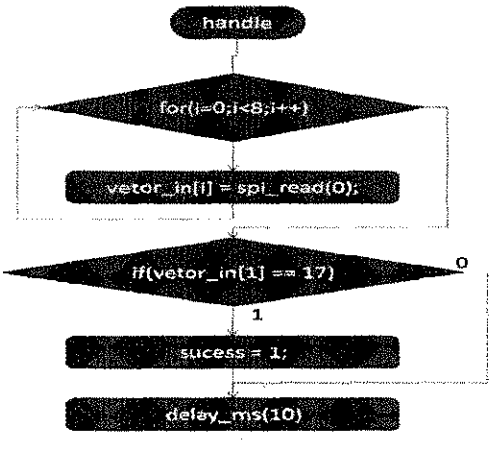


Figura 19 - Fluxograma da função handle()

6.2.1.6 Lógica de Operação

Tendo descrito todas as funções utilizadas, aqui explanamos a lógica global do programa, e como esta é percebida pelo operador.

Logo de início, o programa realiza todas as configurações necessárias, incluindo a do LCD, e a configuração do rádio como receptor. A primeira tela exibida para o operador requisita que este selecione uma operação dentre as possíveis: Entrada, Movimentação e Saída de páletes; a cada uma dessas opções, está associada uma tecla. Ao selecionar a operação, uma segunda tela é exibida, pedindo que o operador efetue a leitura do código de barras, feita essa leitura, o programa checa se o código lido é válido, e ainda exibe-o para o operador, pedindo que este confirme sua exatidão.

Após a confirmação do operador, o sistema passa por uma série de telas onde é requisitado, passo a passo, que o operador entre o endereço lógico do pálete, informando (e confirmando) a área, rua, quadra e estante onde este se encontra.

Encerrando a entrada, o sistema checa se há conexão com a base, havendo, permite que o usuário já envie aquele registro, não havendo, armazena esse registro num *buffer*, e aguarda até que este se encontre numa zona com conexão, dentro dessa zona, a tela indica que há conexão e o número de registros presentes no *buffer*, para que o usuário possa enviá-los.

Dada a extensão do código, remetemos o leitor ao Apêndice C, que contém o código completo do transmissor, comentado.

6.2.2 Receptor

O código do receptor, assim como sua placa, é apenas uma rápida modificação do transmissor, utilizando as mesmas funções. As distinções são: toda a parte de interface com o usuário é ausente, em seu lugar, é colocado um ciclo que periodicamente envia uma *string*

conhecida, é essa *string* que é reconhecida pelos módulos remotos, notificando-os de que a estação base está presente e ativa.

A outra modificação diz respeito à recepção em si, que é feita à chamadas periódicas à função *handle(),* esta função, ao receber os dados advindos de qualquer dos módulos remotos, checa sua validade (testando caracteres específicos da *stream*), e, caso válidos, envia-os de modo formatado para o CP via serial, como mostrado abaixo.

```
if((vetor_in[0] == 'I') || (vetor_in[0] == 'M') || (vetor_in[0] == 'O')){
  printf("%c %02d%02d%02d %c %02d %c %c X", vetor_in[0], vetor_in[1], vetor_in[2],
      vetor_in[3], vetor_in[4], vetor_in[5], vetor_in[6], vetor_in[7]);
}
```

Para o código completo, remetemos o leitor ao Apêndice D.

6.2.3 Programa do CP

No CP, conectado ao receptor, temos o programa que gerencia a recepção dos dados e salva os registros num sistema de arquivos que pode ser lido facilmente pelo sistema de banco de dados já utilizado pela empresa.

Tal programa simplesmente lê os dados recebidos pela porta serial, que já se encontram devidamente formatados pelo receptor, e, ao final de cada registro válido, salva-o no seu respectivo arquivo.

A arquitetura do sistema de arquivos adotada foi discutida com os responsáveis, na empresa, pelo desenvolvimento e manutenção dos softwares e sistemas. A solução empregada consiste em isolar os registros de cada dia em um diretório específico, dentro desse diretório, cada registro tem seu próprio arquivo, cujo nome é o horário em que ele foi lido, com hora, minuto e segundo. Tal sistema insere uma estampa de tempo no registro, permitindo o rastreamento de quando cada pálete passou por operações no setor.

A estratégia de arquivos com registro único, além de evitar que uma possível falha na escrita de um registro comprometa outros, evita choques de leitura/escrita pelo sistema operacional. O sistema de gerenciamento do banco de dados está programado para não ler

o arquivo mais recente na pasta do dia, pois este se trata do arquivo que possivelmente se encontra no processo de escrita do registro pelo programa de salvamento.

O código completo do programa de salvamento, comentado linha a linha, encontra-se no Apêndice E.

O código do sistema de gerenciamento do banco de dados é de propriedade da empresa, e não foi alvo do presente trabalho.

7 Resultados e Discussão

O presente sistema encontra-se em uso há cerca de um mês (no momento da confecção desse relatório), incluindo a primeira semana de testes, durante esse período houve modificações e correções no sistema, assim como ajustes do pessoal ao seu uso.

A migração para o sistema está sendo feita de forma gradual, à medida que novos páletes vão entrando no sistema, já com seus códigos de barra, e os antigos, sem o sistema, vão saindo. Prevê-se que no início do mês de Outubro será feita uma última revisão do estoque, incorporando etiquetas aos últimos páletes restantes ainda sem elas, para que o sistema possa então passar a ser utilizado como única forma de entrada de dados.

A incorporação dos dados advindos do sistema ao banco de dados já foi implementada e já é utilizada pelos responsáveis pelo setor como fonte de informação, com o benefício da atualização da informação no momento em que esta é modificada pelo operador no galpão.

Como ilustração do processo, na Tabela 3 são apresentados exemplos de registros salvos pelo sistema:

Dia	Horário	Registro
27/08/2008	08:01:48	I 045431 0 00 0 0
27/08/2008	10:06:26	M 045431 3 10 1 1
28/08/2008	07:34:09	I 045489 2 11 5 1
28/08/2008	11:41:19	M 045514 2 09 3 1
30/08/2008	10:50:24	I 045644 1 02 3 1
30/08/2008	10:51:20	I 045653 1 02 3 2
30/08/2008	10:51:52	I 045604 2 05 1 3

Tabela 3- Exemplos de registros do sistema

Na Tabela 3, o campo registro é formado da seguinte forma: o primeiro caractere representa a operação efetuada, I = entrada, M = movimentação, O = saída. Em seguida está o número do pálete, seguido de seu endereço, na forma de Área, Rua, Quadra e Estante.

Após a absorção desses dados pelo banco de dados, os supervisores do setor podem gerar relatórios com as operações efetuadas, filtrando por dia, tipo da operação, cliente, etc. bem como rastrear todas as operações já efetuadas sobre um pálete específico. Um exemplo de tal relatório, refletindo alguns dos exemplos da Tabela 3, é apresentado na Tabela 4.

PALETE_EXP_ID	QUANT_MILHEIROS	PESO_LIQ	DATA_ENTRADA	DATA_FABRICACAO	AREA	RUA	QUADRA	ESTANTE	STATUS	CLIENTE	DESTAQUE	OF_FK	ROMANEIO_FK
36600	432.801	477.515	28/8/08	28/8/08	2	9	3	1	I	CIA DE BEBIDAS DAS AMERICAS AMBEV	PEPSI COLA 2L ENROLADOS-ANTI-HORARIO	16176	45514
36676	0	247.485	30/8/08	29/8/08	2	5	1	3	I	DUCOCO PRODUTOS ALIMENTICIOS S/A- (ES)	COCORALADO MENINA 50G-1818	15917	45604
36587	18.507	282.61	28/8/08	27/8/08	2	11	5	1	I	OMNI GROUP INTERNATIONAL, LLC	MUSHROOMS	15984	45489

Tabela 4 - Exemplo de relatório do sistema

8 Conclusão

O presente trabalho mostrou-se uma ferramenta e uma oportunidade de alto valor para a formação profissional, permitindo não apenas uma vasta aplicação dos conhecimentos teóricos adquiridos durante o curso, mas também uma visão em proximidade dos aspectos práticos do trabalho de um engenheiro de projetos. Dentre tais aspectos, estão as análises de custos, a ponderação do fator econômico nas decisões, a utilização do mercado estabelecido para escolha de soluções e, com grande importância, a complexidade da negociação, com clientes, fornecedores, usuários do sistema e outros desenvolvedores, atividades fundamentais na elaboração, desenvolvimento, implantação e adequação do produto/sistema desenvolvido às necessidades reais de sua utilização.

9 Referências Bibliográficas

1. Bar Code History Page. *BarCode1.* [Online] 02 de 02 de 2007. [Citado em: 20 de Agosto de 2008.] http://www.adams1.com/pub/russadam/history.html.

2. Universal Product Code (UPC). *BarCode1.* [Online] 07 de 04 de 2008. [Citado em: 20 de Agosto de 2008.] http://www.adams1.com/pub/russadam/upccode.html.

3. Code 39 Specification Page. *BarCode1.* [Online] 02 de 02 de 2007. [Citado em: 20 de Agosto de 2008.] http://www.adams1.com/pub/russadam/39code.html.

4. Bar Code Readers Page. *BarCode1.* [Online] 02 de 02 de 2007. [Citado em: 20 de Agosto de 2008.] http://www.adams1.com/pub/russadam/readers.html.

5. ZigBee Resource Guide. *ZigBee Alliance.* [Online] 2008. [Citado em: 20 de Agosto de 2008.] http://www.zigbee.org/imwp/idms/popups/pop_download.asp?contentID=12945.

6. **Axelson, Jan.** *Serial Port Complete.* Madison, WI : Lakeview Research, 1999.

7. EPCglobal Class 1 Gen 2 RFID Specification. [Online] 2005. [Citado em: 21 de Agosto de 2008.] http://www.ship2save.com/page_images/wp_alien_gen2.pdf.

8. Interfacing the AT keyboard. *Beyond Logic.* [Online] 15 de 06 de 2005. [Citado em: 21 de Agosto de 2008.] http://www.beyondlogic.org/keyboard/keybrd.htm.

9. Manual de Programação BR-310. *Bematech.* [Online] [Citado em: 21 de Agosto de 2008.] http://www.bematech.com.br/suporte/downloads/manuais/6678.pdf.

10. TRF-2.4G Reference Guide. *SparkFun.* [Online] [Citado em: 20 de Agosto de 2008.] http://www.sparkfun.com/datasheets/RF/RF-24G.pdf.

11. SPI Overview and Use of the PICmicro Serial Peripheral Interface. *Microchip.* [Online] [Citado em: 21 de Agosto de 2008.] http://ww1.microchip.com/downloads/en/DeviceDoc/spi.pdf.

12. XBee™/XBee-PRO™ OEM RF Modules. *Digi.* [Online] 13 de 10 de 2006. [Citado em: 21 de Agosto de 2008.] http://ftp1.digi.com/support/documentation/manual_xb_oemrfmodules_802.15.4.pdf.

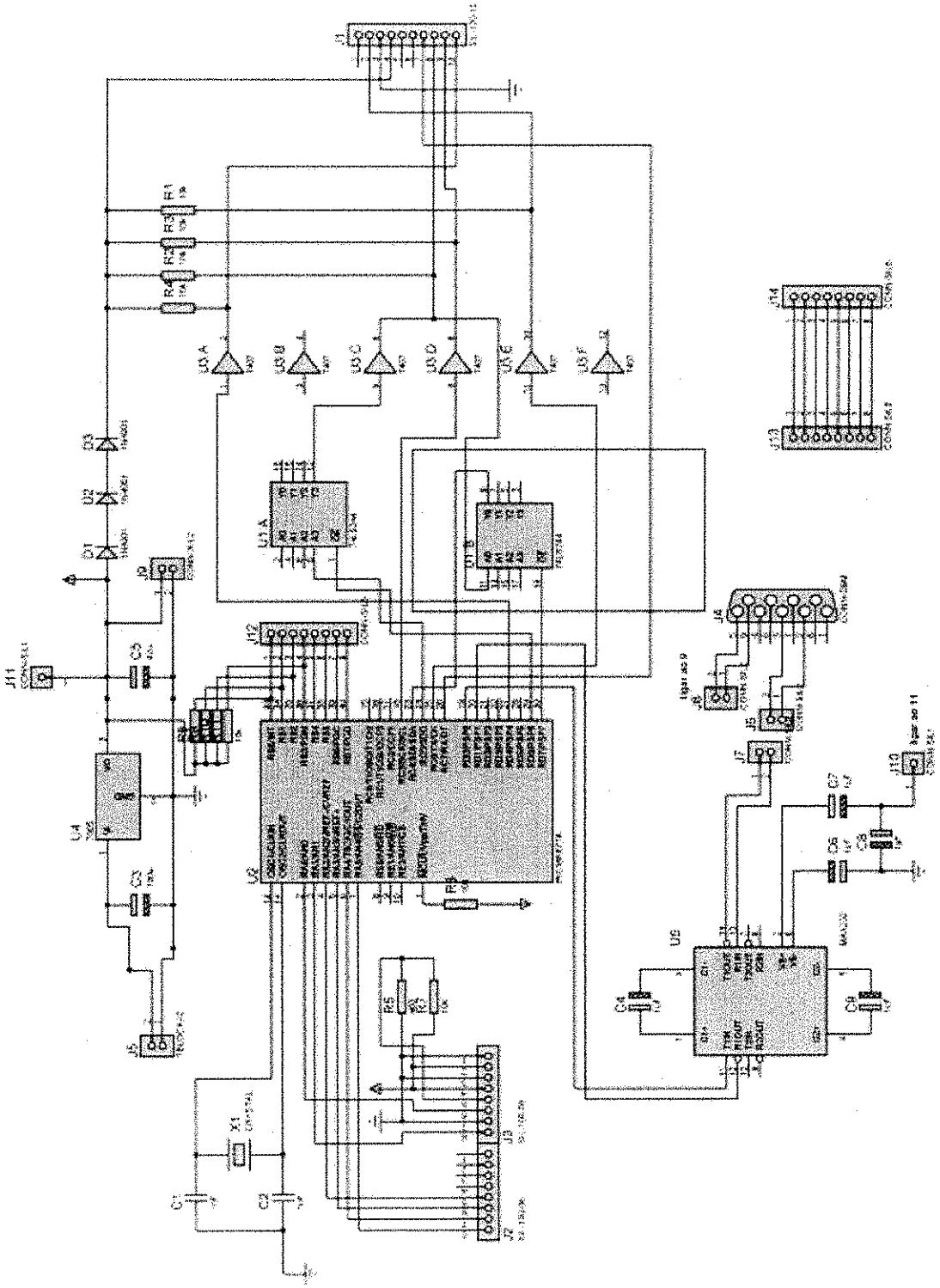
13. LM016L - 16 character x 2lines - Hitachi Semiconductor. *AllDataSheet.* [Online] [Citado em: 21 de Agosto de 2008.] http://www.alldatasheet.com/datasheet-pdf/pdf/146552/HITACHI/LM016L.html.

14. Serial and UART Tutorial. *Freebsd.* [Online] 13 de 01 de 1996. [Citado em: 23 de Agosto de 2008.] http://www.freebsd.org/doc/en_US.ISO8859-1/articles/serial-uart/.

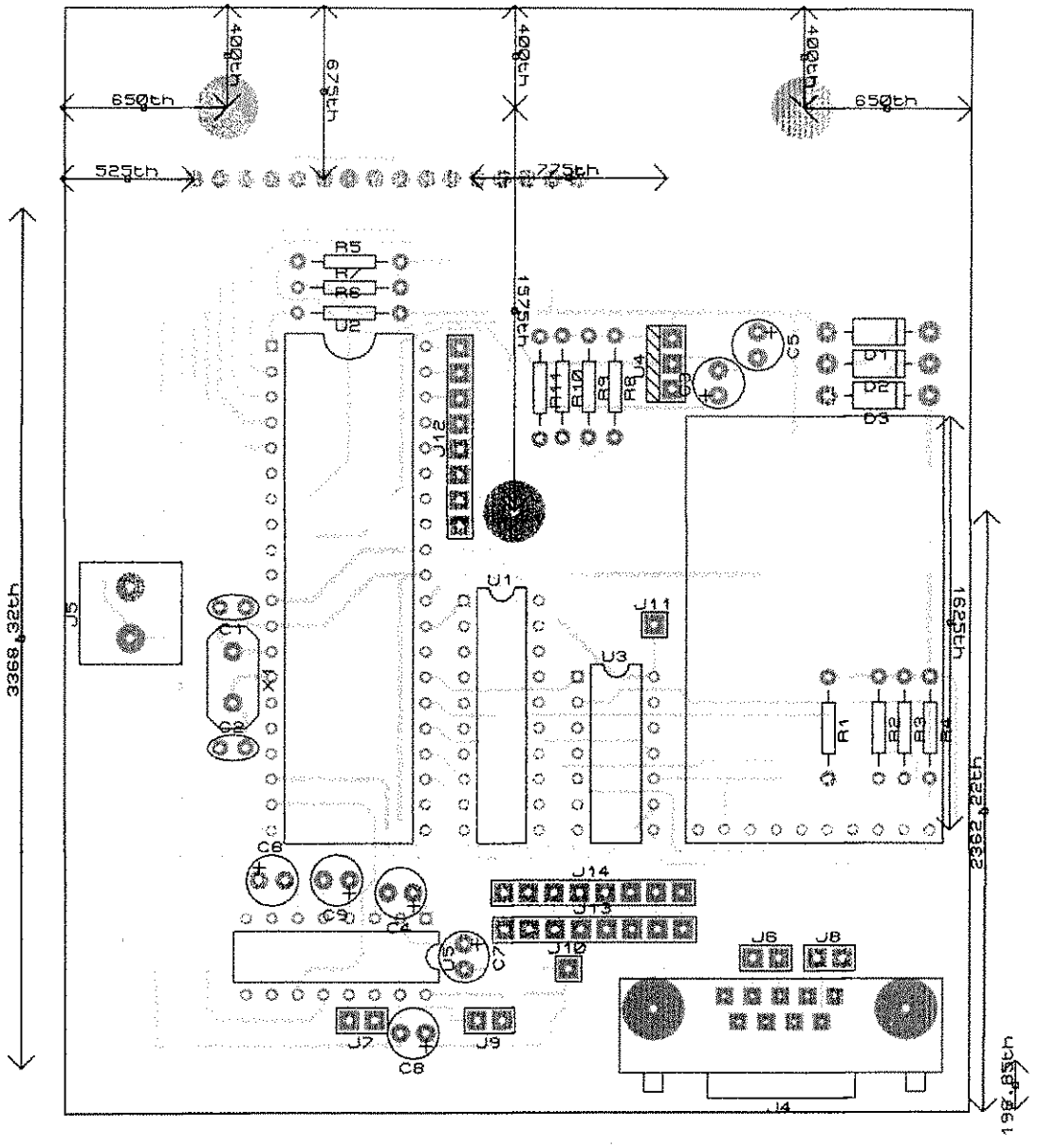
15. PB-114/2. *Patola.* [Online] [Citado em: 23 de Agosto de 2008.] http://patola.com.br/?pagina=info_produto.php&produto_id=60&titulo=PB-114/2.

Apêndices

Apêndice A – Circuito completo dos módulos remotos



Apêndice B - Layout da placa de circuito impresso dos módulos (remoto e base)



Apêndice C – Código completo do transmissor

```
001 #include <16F877A.h>
002 #device adc=8
003
004 #FUSES NOWDT                  //Desabilita o Watch Dog Timer
005 #FUSES XT                     //Usa o Crystal osc <= 4mhz
006 #FUSES PUT                    //Habilita o Power Up Timer
007 #FUSES NOPROTECT              //Leitura do código não protegida
008 #FUSES NODEBUG                //Desabilita o modo Debug
009 #FUSES NOBROWNOUT             //Não reinicia a MCU quando detecta um brownout
010 #FUSES NOLVP                  //Sem programação com baixa tensão
011 #FUSES NOCPD                  //Sem proteção EE
012 #FUSES WRT_50%                //Primeira metade da memória com escrita protegida
013
014 #use delay(clock=4000000)
015 #use rs232(baud=9600,parity=N,xmit=PIN_D0,rcv=PIN_D1,bits=8)
016
017 #define C1              PIN_B3
018 #define C2              PIN_B2
019 #define C3              PIN_B0
020 #define C4              PIN_B1
021 #define L1              PIN_B4
022 #define L2              PIN_B5
023 #define L3              PIN_B6
024 #define L4              PIN_B7
025
026 #define latch  300
027
028 #include <lcd_radio.c>
029
030 unsigned char string_config[]={0xC8,0x40,0x00,0x00,0x00,0x00,0xc8,0x00,0x00,0x00,0x00,0x55,0xAA,0x43,0x4F,0x3C};   //Modo
Transmissor
031 unsigned char string_config2[]={0xC8,0x40,0x00,0x00,0x00,0x00,0xc8,0x00,0x00,0x00,0x00,0x55,0xAA,0x43,0x4F,0x3D};  //Modo Receptor
032 unsigned char end_data[9][10];
033
034 int TN = 1;
035 int TID = 1;
036
037 char tecla, t_area, t_rua_hi, t_rua_lo, t_quadra, t_estante;
038 int code[8];
039 int count = 0;
040 int vetor_in[8];
041 int i = 0, j = 0;
042 int sucess = 0;
043 int rts = 0;
044
045 void handle(void)
046 {
047   for(i=0; i<8; i++){
048     vetor_in[i] = spi_read(0);
049   }
050
051   if(vetor_in[1] == 17)    //dado enviado periodicamente pelo receptor
052     sucess = 1;            //sua recepção indica que o receptor está ativo
053   delay_ms(10);
054
055 }
056
057 char get_key (void);
058 char get_first_key(void);
059 void le_barra (void);
060 void config(int1);
061 void envia(void);
062
063 void main()
064 {
065
066   setup_adc_ports(NO_ANALOG5);
```

```
067  setup_adc(ADC_OFF);
068  setup_psp(PSP_DISABLED);
069  setup_spi(SPI_MASTER|SPI_L_TO_H|SPI_XMIT_L_TO_H|SPI_CLK_DIV_16);
070  //configura a comunicação SPI usada para o transceiver
071  setup_timer_0(RTCC_INTERNAL|RTCC_DIV_1);
072  setup_timer_1(T1_DISABLED);
073  setup_timer_2(T2_DISABLED,0,1);
074  setup_comparator(NC_NC_NC_NC);
075  setup_vref(FALSE);
076
077
078  LCD_init();
079  printf(LCD, "\f->");
080  delay_ms(100);
081
082  config(1);
083
084  for(i = 0;i<=9;i++){
085    end_data[i][0] = 0x55;
086    end_data[i][1] = 0xAA;
087  }
088
089  while(1){
090  if(1)
091  {
092
093
094
095  printf(LCD, "\nEntre a operacao");
096  tecla = get_first_key();
097  if(tecla == 'I'){        //Corresponde à operação de Entrada
098   end_data[rts][2] = 'I';
099   entrada:
100   printf(LCD, "\fEntrada\n");
101   printf(LCD, "Leia a etiqueta");
102   le_barra();
103   if(code[0] == 34){      //checa se o primeiro caractere é 'R'
104   printf(LCD, "\fPallet %d%d%d%d%d%d\nConfirma ?", code[1], code[2], code[3], code[4], code[5], code[6]);
105   tecla = get_key();      //espera confirmação do pálete lido pelo operador
106   if(tecla == 'E'){       //tecla usada pelo operador para confirmar o dado
107    end_data[rts][3] = 10*code[1]+code[2];   //aloca o código lido no buffer a ser enviado
108    end_data[rts][4] = 10*code[3]+code[4];
109    end_data[rts][5] = 10*code[5]+code[6];
110    area:                    //aqui se inicia a inserção do endereço do pálete
111    printf(LCD, "\fEntre a area: ");   //entrada da área
112    delay_ms(latch);
113    t_area = get_key();
114    printf(LCD, "%c\nConfirma ?", t_area);
115    delay_ms(latch);
116    tecla = get_key();
117    if(tecla == 'E'){
118     end_data[rts][6] = t_area;
119     rua:                    //entrada da rua
120     printf(LCD, "\fEntre a rua: ");
121     delay_ms(latch);
122     t_rua_hi = get_key();
123     printf(LCD, "%c", t_rua_hi);
124     delay_ms(latch);
125     t_rua_lo = get_key();
126     printf(LCD, "%c\nConfirma ?", t_rua_lo);
127     delay_ms(latch);
128     tecla = get_key();
129     if(tecla == 'E'){
130      end_data[rts][7] = 10*(t_rua_hi-48)+(t_rua_lo-48);
131      quadra:                 //entrada da quadra
132      printf(LCD, "\fEntre a quadra: ");
133      delay_ms(latch);
134      t_quadra = get_key();
135      printf(LCD, "\n%c Confirma ?", t_quadra);
136      delay_ms(latch);
137      tecla = get_key();
```

```
138         if(tecla == 'E'){
139           end_data[rts][8] = t_quadra;
140           estante:              //entrada da estante
141           printf(LCD, "\fEntre a estante:");
142           delay_ms(latch);
143           t_estante = get_key();
144           printf(LCD, "\n%c Confirma ?", t_estante);
145           delay_ms(latch);
146           tecla = get_key();
147           if(tecla == 'E'){
148             end_data[rts][9] = t_estante;
149             rts ++;         //indica que tem dados para enviar
150           }
151         }
152         else
153           goto quadra;
154       }
155       else
156         goto rua;
157     }
158     else
159       goto area;
160   }
161   else{
162    goto entrada;
163   }
164  }
165  else goto entrada;
166  }
167  else if(tecla == 'M'){      //Corresponde à operação de movimentação
168   end_data[rts][2] = 'M';
169   movimentacao:
170   printf(LCD, "\fMovimentacao\n");
171   printf(LCD, "Leia a etiqueta");
172   le_barra();
173   if(code[0] == 34){
174   printf(LCD, "\fPallet %d%d%d%d%d%d\nConfirma ?", code[1], code[2], code[3], code[4], code[5], code[6]);
175   tecla = get_key();
176   if(tecla == 'E'){
177    end_data[rts][3] = 10*code[1]+code[2];
178    end_data[rts][4] = 10*code[3]+code[4];
179    end_data[rts][5] = 10*code[5]+code[6];
180    goto area;    //como o procedimento é idêntico ao de entrada, procede para o mesmo trecho de código
181   }
182   else
183    goto movimentacao;
184   }
185   else goto movimentacao;
186  }
187  else if(tecla == 'O'){      //Corresponde à operação de saida
188   end_data[rts][2] = 'O';
189   saida:
190   printf(LCD, "\fSaida\n");
191   printf(LCD, "Leia a etiqueta");
192   le_barra();
193   if(code[0] == 34){
194   printf(LCD, "\fPallet %d%d%d%d%d%d\nConfirma ?", code[1], code[2], code[3], code[4], code[5], code[6]);
195   tecla = get_key();
196   if(tecla == 'E'){
197    end_data[rts][3] = 10*code[1]+code[2];
198    end_data[rts][4] = 10*code[3]+code[4];
199    end_data[rts][5] = 10*code[5]+code[6];
200    end_data[rts][6] = 'O';          //Na saida, não faz sentido usar endereço
201    end_data[rts][7] = 0;            //pois o palete está deixando seu endereço atual
202    end_data[rts][8] = 'O';          //e este já consta no sistema
203    end_data[rts][9] = 'O';
204    rts++;         //indica que tem dados para enviar
205   }
206   else
207    goto saida;
208   }
```

```
209    else goto saida;
210   }
211
212
213
214   if(TID == 2)
215     handle(i.
216
217   TN++;
218   if(TN == 1){
219     config(1);
220     sucess = 0;
221   }
222
223   if(TN == 200){   //uma vez a cada 200 ciclos, configura como transmissor
224     config(0);
225     if(sucess){   //checa se durante esses 200 ciclos, houve contato com a base
226       printf(LCD, "\fConectado %02d", rts); //notifica o operador que este pode se comunicar
227       if(rts){                //checa se há dados a serem transmitidos no buffer
228         printf(LCD, "\nEnvia?"); //indaga do operador se esses dados devem ser transmitidos
229         delay_ms(latch);
230         tecla = get_key();
231         if(tecla == 'E'){
232           rts--;
233           envia();     //envia os dados, decrementando o buffer
234         }
235       }
236     }
237     else{
238       printf(LCD, "\fSem Sinal %02d", rts); //caso esteja numa zona sem conexão
239     }
240     TN = 0;
241   }
242
243   }
244
245   }
246
247 }
248
249 void config(int1 sentido){
250   output_bit(PIN_D4,1);  //Sinal CS do rádio (habilita configuração)
251   output_bit(PIN_C6,0);  //habilita recepção CE
252   output_bit(PIN_D7,1);  //desabilita 74LS244 a passar os dados do rádio pro pic
253   output_bit(PIN_D6,0);  //Sinal WR* do 74LS244 habilita o envio dos dados do pic pro rádio
254
255   if(sentido){
256   TID = 2;
257   for(i=0;i<15;i++){
258    spi_write(string_config2[i]);  //configura como receptor
259   }
260   }
261   else{
262   TID = 1;
263   for(i=0;i<15;i++){
264    spi_write(string_config[i]);  //configura como transmissor
265   }
266   }
267
268   output_bit(PIN_D4,0);  //Sinal CS do rádio
269   output_bit(PIN_C6,1);  //habilita recepção CE
270   output_bit(PIN_D7,0);  //habilita 74LS244 a passar os dados do rádio pro pic
271   output_bit(PIN_D6,1);  //Sinal WR* do 74LS244 bloqueia o envio dos dados do pic pro rádio
272 }
273
274 void envia (void){
275   output_bit(PIN_D7,1);  // desabilita 74LS244 a passar os dados do rádio pro pic
276   output_bit(PIN_D6,0);  // Ativa sinal WR* do 74LS244 (habilita escrita de dados para rádio)
277   output_bit(PIN_C6,1);  // Ativa sinal CE do rádio (habilita o rádio a receber dados)
278
279   for(j=0; j<10; j++){
```

```
280    spi_write(end_data[rts][j]);    // Escreve dados para o rádio
281   }
282
283   output_bit(PIN_C6,0);  // Desativa sinal CE do rádio (inicia a transmissão em si dos dados)
284   output_bit(PIN_D6,1);  // Desativa sinal WR* do 74LS244 (desabilita escrita de dados para rádio)
285   output_bit(PIN_D7,0);  // habilita 74LS244 a passar os dados do rádio pro pic
286 }
287
288 void le_barra(void){
289   while(count < 8){          //código tem 8 caracteres ('R' + 6 de ID + CRC)
290     code[count] = getc()-48;  //converte de ASCII para os valores numéricos
291     count++;
292   }
293   count = 0;
294   return;
295 }
296
297 char get_key (void)
298 {
299   char key = ' ';
300   while(key == ' '){       //faz a leitura ciclicamente
301   output_bit(L1, FALSE);    //ativa a linha 1
302   output_bit(L2, TRUE );
303   output_bit(L3, TRUE );
304   output_bit(L4, TRUE );
305
306   if(!input(C1))          //lê o estado das colunas, associando uma resposta
307     key = '*';           //à tecla correspondente
308   if(!input(C2))
309     key = '7';
310   if(!input(C3))
311     key = '8';
312   if(!input(C4))
313     key = '9';
314
315   output_bit(L1, TRUE );
316   output_bit(L2, FALSE);    //o procedimento é repetido para as demais linhas
317   output_bit(L3, TRUE );
318   output_bit(L4, TRUE );
319
320   if(!input(C1))
321     key = 'M';
322   if(!input(C2))
323     key = '4';
324   if(!input(C3))
325     key = '5';
326   if(!input(C4))
327     key = '6';
328
329   output_bit(L1, TRUE );
330   output_bit(L2, TRUE );
331   output_bit(L3, FALSE);
332   output_bit(L4, TRUE );
333
334   if(!input(C1))
335     key = 'D';
336   if(!input(C2))
337     key = '1';
338   if(!input(C3))
339     key = '2';
340   if(!input(C4))
341     key = '3';
342
343   output_bit(L1, TRUE );
344   output_bit(L2, TRUE );
345   output_bit(L3, TRUE );
346   output_bit(L4, FALSE);
347
348   if(!input(C1))
349     key = '0';
350   if(!input(C2))
```

```
351     key = '0';
352   if(!input(C3))
353     key = '.';
354   if(!input(C4))
355     key = 'E';
356   }
357
358   return key;          //retorna a tecla pressionada
359 }
360
361 char get_first_key(void)
362 {
363   char key = ' ';
364   output_bit(L1, FALSE);
365   output_bit(L2, TRUE );
366   output_bit(L3, TRUE );
367   output_bit(L4, TRUE );
368
369   if(!input(C1))
370     key = 'I';
371   if(!input(C2))
372     key = '7';
373   if(!input(C3))
374     key = '8';
375   if(!input(C4))
376     key = '9';
377
378   output_bit(L1, TRUE );
379   output_bit(L2, FALSE);
380   output_bit(L3, TRUE );
381   output_bit(L4, TRUE );
382
383   if(!input(C1))
384     key = 'M';
385   if(!input(C2))
386     key = '4';
387   if(!input(C3))
388     key = '5';
389   if(!input(C4))
390     key = '6';
391
392   output_bit(L1, TRUE );
393   output_bit(L2, TRUE );
394   output_bit(L3, FALSE);
395   output_bit(L4, TRUE );
396
397   if(!input(C1))
398     key = 'D';
399   if(!input(C2))
400     key = '1';
401   if(!input(C3))
402     key = '2';
403   if(!input(C4))
404     key = '3';
405
406   output_bit(L1, TRUE );
407   output_bit(L2, TRUE );
408   output_bit(L3, TRUE );
409   output_bit(L4, FALSE);
410
411   if(!input(C1))
412     key = 'O';
413   if(!input(C2))
414     key = '0';
415   if(!input(C3))
416     key = '.';
417   if(!input(C4))
418     key = 'E';
419
420   return key;
421 }
```

Apêndice D – Código completo do receptor

```
001 #include <16F877A.h>
002 #device adc=8
003
004 #FUSES NOWDT                  //No Watch Dog Timer
005 #FUSES XT                     //Crystal osc <= 4mHz
006 #FUSES PUT                    //Power Up Timer
007 #FUSES NOPROTECT              //Code not protected from reading
008 #FUSES NODEBUG                //No Debug mode for ICD
009 #FUSES NOBROWNOUT             //No brownout reset
010 #FUSES NOLVP                  //No low voltage prgming, B3(PIC16) or B5(PIC18) used for I/O
011 #FUSES NOCPD                  //No EE protection
012 #FUSES WRT_50%                //Lower half of Program Memory is Write Protected
013
014 #use delay(clock=4000000)
015 #use rs232(baud=9600,parity=N,xmit=PIN_D0,rcv=PIN_D1,bits=8)
016
017 #define C1              PIN_B3
018 #define C2              PIN_B2
019 #define C3              PIN_B0
020 #define C4              PIN_B1
021 #define L1              PIN_B4
022 #define L2              PIN_B5
023 #define L3              PIN_B6
024 #define L4              PIN_B7
025
026
027 unsigned char string_config[]={0xC8,0x40,0x00,0x00,0x00,0x00,0xc8,0x00,0x00,0x00,0x00,0x55,0xAA,0x43,0x4F,0x3C};
028 unsigned char string_config2[]={0xC8,0x40,0x00,0x00,0x00,0x00,0xc8,0x00,0x00,0x00,0x00,0x55,0xAA,0x43,0x4F,0x3D};
029 unsigned char end_data[]={0x55,0xAA,0x11,0x11,0x11,0x11,0x11,0x11,0x11,0x11};
030
031 int TN = 1;
032 int TID = 1;
033
034 char tecla;
035 int code[8];
036 int count = 0;
037 int vetor_in[8];
038 int i = 0, j = 0;
039
040 void handle(void)
041 {
042   for(i=0; i<8; i++){
043     vetor_in[i] = spi_read(0);
044   }
045
046   tecla = vetor_in[1];
047   if((vetor_in[0] == 'I') || (vetor_in[0] == 'M') || (vetor_in[0] == 'O')){
048     printf("%c %02d%02d%02d %c %02d %c %c X", vetor_in[0], vetor_in[1], vetor_in[2], vetor_in[3], vetor_in[4], vetor_in[5],
vetor_in[6], vetor_in[7]);
049   }
050   delay_ms(10);
051
052 }
053
054 void config(int1);
055 void envia(void);
056
057 void main()
058 {
059
060   setup_adc_ports(NO_ANALOGS);
061   setup_adc(ADC_OFF);
062   setup_psp(PSP_DISABLED);
063   setup_spi(SPI_MASTER|SPI_L_TO_H|SPI_XMIT_L_TO_H|SPI_CLK_DIV_16);
```

```
064  setup_timer_0(RTCC_INTERNAL|RTCC_DIV_1);
065  setup_timer_1(T1_DISABLED);
066  setup_timer_2(T2_DISABLED,0,1);
067  setup_comparator(NC_NC_NC_NC);
068  setup_vref(FALSE);
069
070  config(1);
071
072  while(1){
073  if(1)
074  {
075    if(TID == 2)
076      handle();
077
078  TN++;
079  if(TN == 1){
080    config(1);
081  }
082
083  if(TN == 200){
084    config(0);
085    envia();
086    TN = 0;
087  }
088
089  }
090
091  }
092
093 }
094
095 void config(int1 sentido){
096  output_bit(PIN_D4,1);  //Sinal CS do rádio (habilita configuração)
097  output_bit(PIN_C6,0);  //habilita recepção CE
098  output_bit(PIN_D7,1);  //desabilita 74LS244 a passar os dados do rádio pro pic
099  output_bit(PIN_D6,0);  //Sinal WR* do 74LS244 habilita o envio dos dados do pic pro rádio
100
101  if(sentido){
102  TID = 2;
103  for(i=0;i<15;i++){
104   spi_write(string_config2[i]);
105  }
106  }
107  else{
108  TID = 1;
109  for(i=0;i<15;i++){
110   spi_write(string_config[i]);
111  }
112  }
113
114  output_bit(PIN_D4,0);  //Sinal CS do rádio
115  output_bit(PIN_C6,1);  //habilita recepção CE
116  output_bit(PIN_D7,0);  //habilita 74LS244 a passar os dados do rádio pro pic
117  output_bit(PIN_D6,1);  //Sinal WR* do 74LS244 bloqueia o envio dos dados do pic pro rádio
118 }
119
120 void envia (void){
121  output_bit(PIN_D7,1);  // desabilita 74LS244 a passar os dados do rádio pro pic
122  output_bit(PIN_D6,0);  // Ativa sinal WR* do 74LS244 (habilita escrita de dados para rádio)
123  output_bit(PIN_C6,1);  // Ativa sinal CE do rádio (habilita o rádio a receber dados)
124
125  for(j=0; j<10; j++){
126   spi_write(end_data[j]);    // Escreva dados para o rádio
127  }
128
129  output_bit(PIN_C6,0);  // Desativa sinal CE do rádio (desabilita o rádio a receber dados)
130  output_bit(PIN_D6,1);  // Desativa sinal WR* do 74LS244 (desabilita escrita de dados para rádio)
131  output_bit(PIN_D7,0);  // habilita 74LS244 a passar os dados do rádio pro pic
132 }
```

Apêndice E – Código do aplicativo de salvamento (PC)

```
#include <bios.h>
#include <conio.h>
#include <stdio.h>
#include <dos.h>
#include <string.h>
#include <process.h>

#define COM1     0
#define DATA_READY 0x100
#define TRUE     1
#define FALSE    0
#define SETTINGS (_COM_9600 | _COM_CHR8 | _COM_STOP1 | _COM_NOPARITY)

int main(void)
{
 unsigned in, out, status;
 int count = 0;
 char code[20] = "";
 char last[20] = "";
 char nome[20] = "";
 char command[12];
 FILE *save;
 struct time t;
 struct date d;

 _bios_serialcom(_COM_INIT, COM1, SETTINGS);                    //configura porta serial utilizada

 clrscr();
 cprintf("... Programa receptor dos códigos de barra ...\r\n");

 for (;;)
 {
  status = _bios_serialcom(_COM_STATUS, COM1, 0);                //lê o status da porta serial
  if (status & DATA_READY)
          //caso haja dados no buffer de entrada
    if ((out = _bios_serialcom(_COM_RECEIVE, COM1, 0) & 0x7F) != 0){    //lê esses dados para a variável reservada
          code[count] = out;                                     //anexa a última leitura na stream
          printf("%c", code[count]);                             //exibe em tela cada caractere lido
          if(code[count] == 'X')                                 //o X marca o fim do registro
                  printf("\n");
          count++;
          }
          if(code[count-1] == 'X'){                              //se o registro foi completado
          gettime(&t);                                           //lê o tempo
          getdate(&d);                                           //e a data do sistema
          sprintf(command, "md %02d%02d%02d", d.da_year, d.da_mon, d.da_day);
          //cria um comando para o núcleo do DOS que cria um diretório cujo nome é a data atual
          system(command);
          //executa o comando acima por passagem de comando do sistema
          sprintf(command, "cd %02d%02d%02d", d.da_year, d.da_mon, d.da_day);
          //muda o diretório atual para o recém criado
          system(command);
          sprintf(nome, "%02d%02d%02d.txt", t.ti_hour, t.ti_min, t.ti_sec);
          //cria o nome do arquivo a ser salvo com a hora (hora, minuto e segundo)
          save = fopen(nome, "a+");                              //abre, criando, o arquivo onde será salvo o registro lido
          code[18] = '\0';                                       //fecha a stream a ser salva
          if(strcmp(code, last)){                                //compara se o último registro já não foi lido
              if((code[0] == 'I') || (code[0] == 'M') || (code[0] == 'O'))    //testa se o primeiro caractere do registro é válido
                  fprintf(save, "%s\n", code);
                  //caso ambos os testes acima validem, salva o registro no arquivo aberto
          }
          strcpy(last, code);                                    //armazena o último registro salvo para comparação
```

```
            fclose(save);                                   //fecha o arquivo desse registro
            system("cd..");                                 //retorna para o diretório onde se encontra o aplicativo
            count = 0;                                      //limpa a stream de recepção
    }
   if (kbhit())
   {
    if ((in = getch()) == '\x1B')                           //detecta o pressionamento do 'ESC', para deixar o programa
      break;
    _bios_serialcom(_COM_SEND, COM1, in);
   }
  }
  return 0;
}
```