UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO

(Bacharelado)

PROTÓTIPO DE *SOFTWARE* PARA COMUNICAÇÃO DE DADOS SEM FIO COM UMA AGENDA ELETRÔNICA

TRABALHO DE CONCLUSÃO DE CURSO SUBMETIDO À UNIVERSIDADE REGIONAL DE BLUMENAU PARA A OBTENÇÃO DOS CRÉDITOS NA DISCIPLINA COM NOME EQUIVALENTE NO CURSO DE CIÊNCIAS DA COMPUTAÇÃO — BACHARELADO

MARLO ALEXANDRE BACK

BLUMENAU, DEZEMBRO/2000

2000/2-39

PROTÓTIPO DE *SOFTWARE* PARA COMUNICAÇÃO DE DADOS SEM FIO COM UMA AGENDA ELETRÔNICA

MARLO ALEXANDRE BACK

ESTE TRABALHO DE CONCLUSÃO DE CURSO, FOI JULGADO ADEQUADO PARA OBTENÇÃO DOS CRÉDITOS NA DISCIPLINA DE TRABALHO DE CONCLUSÃO DE CURSO OBRIGATÓRIA PARA OBTENÇÃO DO TÍTULO DE:

BACHAREL EM CIÊNCIAS DA COMPUTAÇÃO

_____

Prof. Miguel Alexandre Wisintainer — Orientador na FURB

_____

Prof. José Roque Voltolini da Silva — Coordenador do TCC

BANCA EXAMINADORA

_____

Prof. Miguel Alexandre Wisintainer - Orientador

_____

Prof. Antonio Carlos Tavares

_____

Prof. Sérgio Stringari

AGRADECIMENTOS

Gostaria de deixar registrado o quando sou grato a uma pessoa que foi fundamental para o desenvolvimento deste trabalho – porque sem ele, este protótipo jamais teria saído do papel – meu orientador, o Professor Miguel Alexandre Wisintainer. Agradeço também pelas suas horas extras de dedicação ao meu trabalho, pela sua extrema atenção e dedicação em resolver os problemas decorrentes no trabalho.

Quero agradecer ao Professor Sérgio Stringari, por ter me inspirado na criação deste trabalho com seus ensinamentos sobre comunicação de dados. E também aos demais professores do Curso de Ciências da Computação, do qual tenho orgulho de ser formando.

Um agradecimento especial para minha noiva, Luiza Izaura Wiedemer, que me ajudou e incentivou durante todos estes semestres da faculdade, desde o primeiro dia.

Também quero agradecer à WF Automação pela ajuda na montagem dos módulos, ao colega Ronaldo Martins pelas dicas fundamentais na elaboração dos circuitos, ao colega Nelson Besen pela idéia do sistema de transmissão sem fios.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

ACK	*Acknowledgement*: confirmação positiva
AM	*Amplitude Modulation*: modulação por amplitude
ASCII	*American Standard Code for Information Interchange*: código de padrão americano para intercâmbio de informações
ASK	*Amplitude Key Shift Keying*: amplitude por chaveamento
bit	*Binary Digit*: dígito binário
bps	*Bits* por segundo
BSC	*Binary Synchronous Communication*: comunicação síncrona binária
CCITT	*Comité Consultatif Internationale de Telegraphie et Telephonie*: órgão que define padrões de telecomunicações.
CRC	*Cyclical Redundancy Check*: checagem redundante cíclica
CW	*Continuos Wave*: onda contínua
EBCDIC	*Extended Binary Coded Decimal Interchange Code*: código de intercâmbio decimal de codificação binária extendida.
EIA	*Eletronics Industries Association*: associação das indústrias de eletrônica
EHF	*Extremely High Frequency*: freqüência extremamente alta
EOT	*End Of Text*: fim do texto
FCC	*Federal Communications Commission*: comissão federal de comunicações
FM	*Frequency Modulation*: modulação por freqüência
FSK	*Frequency Shift Keying*: freqüência por chaveamento

FTP	*File Transfer Protocol*: protocolo de transferência de arquivos
GHz	Gigahertz
GPS	*Global Positioning System*: sistema de posicionamento global
HF	*High Frequency*: freqüência alta
IBM	*Internacional Business Machine*
IEEE	*Institute of Electrical and Eletronic Engineers*: instituto de engenharia elétrica e eletrônica
IP	*Internet Protocol*: protocolo da Internet
ISO	*International Standards Organization*: organização de padrões internacionais
Kbps	Kilobits por segundo
KHz	Kilohertz
LAN	*Local Area Network*: rede de computadores local
LF	*Low Frequency*: freqüência baixa
LRC	*Longitudinal Redundancy Check*: checagem redundante longitudinal
Mbps	Megabits por segundo
MF	*Medium Frequency*: freqüência média
MHz	Megahertz
NACK	*Negative Acknowledgement*: confirmação negativa
OHM	Ohms
OSI	*Open System Interconnection*: interconexão de sistemas abertos
PC	*Personal Computer*: computador pessoal

PM	*Phase Modulation*: modulação por fase
PSK	*Phase Shift Keying*: fase por chaveamento
PWM	*Pulse Width Modulation*: modulação por largura de pulso
SDLC	*Synchronous Data Link Communication*: comunicação ligada a dados sincronizados
SHF	*Super High Frequency*: freqüência super alta
SOH	*Start Of Header*: início do cabeçalho
SOT	*Start Of Text*: início do texto
STP	*Shielded Twisted Pair*: par trançado blindado
TCP	*Transmission Control Protocol*: protocolo para controle de transmissão
THF	*Tremendly High Frequency*: freqüência tremendamente alta
THz	Terahertz
TP	*Twisted Pair*: par trançado
USART	*Universal Synchronous Asynchronous Receiver Transmitter*: transmissor/receptor universal síncrino e assíncrono
UHF	*Ultra High Frequency*: freqüência ultra elevada
UTP	*Unshielded Twisted Pair*: par trançado não blindado
V	Volts
VHF	*Very High Frequency*: freqüência muito elevada
VRC	*Vertical Redundancy Check*: checagem redundante vertical
WAN	*Wide Area Network*: redes de computadores de longo alcance

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

Através do uso da tecnologia de comunicação de dados sem fios (rádio-freqüência), este trabalho demonstra uma interação entre uma agenda eletrônica e um PC. A agenda é utilizada para enviar requisições ao PC, que por sua vez, processa e retorna a informação requisitada. O trabalho também apresenta técnicas para comunicação serial com rádio-freqüência e documentação do protocolo nativo da agenda.

ABSTRACT

By using wireless communication technology (radio-frequency), this works demonstrates an interaction between a digital diary and a PC. The digital diary is used to send solicitations to PC, that when receiving, processes and returns the requested information. The works also presents techniques for serial communication with radio-frequency and documentation of the native protocol of the digital diary.

1 INTRODUÇÃO

Com a evolução acentuada da eletrônica verificada nos últimos anos, cada vez mais máquinas vêm incorporando a tecnologia de processamento eletrônico de dados, tornando assim, o computador uma das figuras mais imprescindíveis no cotidiano de uma sociedade organizada. Além disso, estes avanços tem propiciado uma contínua redução de custos dos equipamentos de processamento de dados, o que também força as empresas a continuamente reverem suas instalações.

Particularmente na área de informática, pode-se assistir, em apenas alguns anos, a evolução de computadores de enormes dimensões e reduzida capacidade de processamento, chegarem a máquinas poderosíssimas, que praticamente cabem na palma da mão. Mesmo assim, os computadores de grande porte ou os microcomputadores (PCs) não conseguem mais isoladamente atender às necessidades pessoais ou empresariais do mundo de hoje. Essas máquinas precisam “falar” entre si. É necessário que elas se comuniquem, troquem informações, travem um diálogo, enfim, permitindo que a coisa mais importante do mundo dos negócios, a informação, esteja sempre disponível no momento e locais desejados ([ALD2000]).

A necessidade de portabilizar esses equipamentos, vem se tornando cada vez mais comum em organizações de todos os tamanhos, inclusive em lares. A vantagem em mobilizar ou transportar um determinado *hardware,* para atender às necessidades das pessoas, pode ser visto como uma boa redução de tempo ao executar tarefas e com isso redução de custos, além de toda a comodidade.

A comunicação de dados e informações através de equipamentos que não utilizam fios como meio de transmissão, é conhecida como *Wireless*. O ambiente *Wireless* está emergindo como uma grande opção também para rede de computadores. Como essa tecnologia ainda está amadurecendo, os comerciantes estão oferecendo mais produtos e com preço mais atrativos, o que significa aumento nas vendas e na demanda. Com a demanda aumentando, essa tecnologia está evoluindo cada vez mais ([MIN1998]).

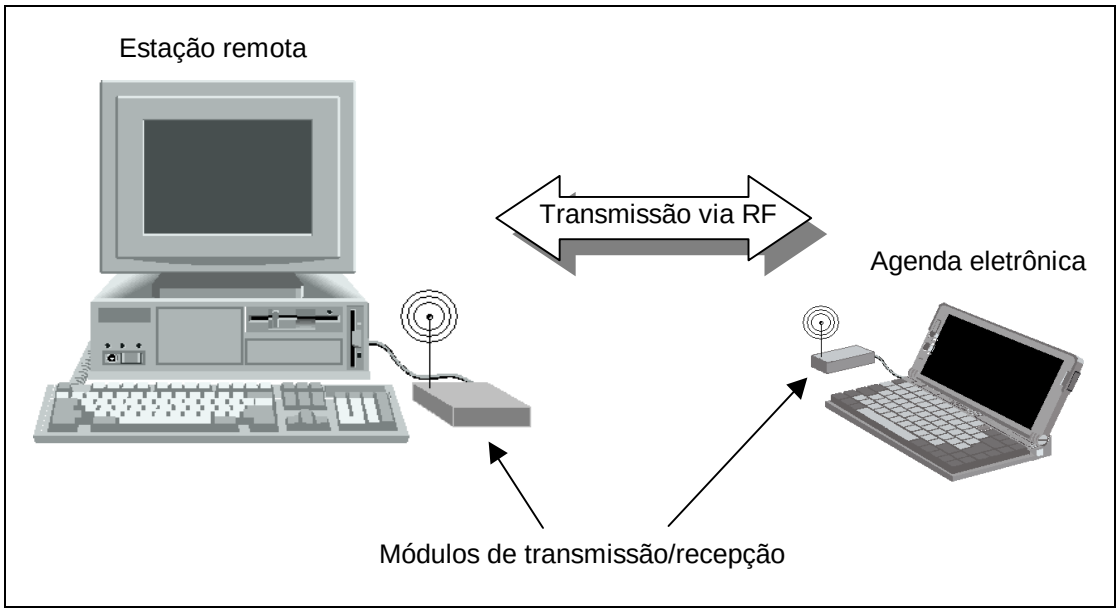
1.1 ORIGEM E MOTIVAÇÃO

A idéia deste trabalho surgiu a partir de uma experiência de comunicação entre uma agenda eletrônica (fig. 1) e um PC. Existem certos tipos de agenda eletrônica que possuem um conector que permite que a mesma seja ligada a um micro, através de um cabo especial, para fazer *backup* dos dados contidos em sua memória.

FIGURA 1 – AGENDA ELETRÔNICA.

Neste protótipo é demonstrado uma maneira de interagir com o computador utilizando a agenda (fig. 2). Utilizando-se de recursos de comunicação, é possível transformar a agenda eletrônica em um terminal do PC. Com isso é possível enviar “instruções” ao computador para que o mesmo execute determinadas tarefas, como por exemplo, requisitar alguma informação contida em seu banco de dados, ou mesmo controlar algum outro periférico que esteja conectado ao mesmo.

FIGURA 2 – INTERAÇÃO ENTRE AGENDA E O PC.



1.2 ÁREA

Este trabalho é caracterizado pelo estudo da comunicação de dados, em especial, a "comunicação de dados" sem fio (*wireless*) utilizando **rádio-freqüência**. Também é abordado o conceito de **protocolos**, que é um item fundamental para o desenvolvimento do protótipo.

1.3 PROBLEMA

A idéia de interligar uma agenda eletrônica com um micro pode ser bastante curiosa e interessante, porém não seria nem um pouco prática devido a necessidade de existir um cabo entre os dois equipamentos.

No caso deste protótipo, seria necessário manter a comunicação ativa entre os dois, mesmo que estivessem distantes entre si. Aumentar o tamanho do cabo não seria a solução ideal, pois haveria o incômodo causado pelo mesmo.

Então para contornar este problema, teria que se elaborar algo que pudesse transmitir as informações pelo ar, utilizando técnicas de comunicação sem fio, em especial a rádio-freqüência.

Outro grande problema, apesar de não aparente, é a "linguagem" de comunicação da agenda com o PC. O fabricante da mesma afirma oficialmente, em [CAS2000], não disponibilizar a documentação sobre o mesmo.

Neste trabalho será demonstrado, através de recursos computacionais e componentes eletrônicos, uma solução simples e prática para "contornar" os problemas acima mencionados.

1.4 PROTÓTIPO

Algumas pessoas devem questionar-se o porquê da utilização de uma agenda eletrônica para interagir com um microcomputador. A resposta é simples, ao contrário do que se pensa, a agenda utilizada neste protótipo não é utilizada para transmitir e receber registros telefônicos, mas sim textos que podem ser interpretados como uma lista de comandos. Essa característica permite uma interação muito mais eficiente entre as duas partes, se comparadas, por exemplo, a um coletor de dados comercial.

Através da pesquisa e do estudo do processo de comunicação de dados, é possível entender e documentar o funcionamento da comunicação entre a agenda e o PC. Com isso, é perfeitamente possível criar vários sistemas para uso comercial e industrial a partir do resultado deste trabalho.

1.5 OBJETIVOS DO TRABALHO

Este trabalho apresenta como objetivo demonstrar uma interação entre uma agenda eletrônica e um microcomputador, fundamentando-se na pesquisa e estudo teórico sobre a comunicação de dados sem fio, enfatizando a rádio-freqüência, e também protocolos.

1.6 ESTRUTURA DO TRABALHO

Este trabalho está divido em 6 partes (capítulos). O primeiro capítulo apresenta a contextualização e a justificativa para o desenvolvimento da proposta do trabalho.

O segundo capítulo aborta a fundamentação teórica da comunicação de dados, bem como representação dos dados, transferência dos dados, meios físicos, modulação e interface serial.

O terceiro capítulo trata sobre comunicação de dados sem fio (*wireless*), mencionando as principais tecnologias utilizadas, com destaque na rádio-freqüência.

O quarto capítulo apresenta a teoria dos protocolos, seus tipos e classificações.

O quinto capítulo refere-se ao desenvolvimento do trabalho, relacionando os componentes, metodologias e a especificação do protótipo.

O sexto capítulo apresenta a conclusão e as considerações finais, abrangendo as dificuldades encontradas, limitações e as sugestões para próximos trabalhos.

2 COMUNICAÇÃO DE DADOS

A comunicação é formada de meios e regras pelos quais, a mensagem, o elemento fonte desta comunicação, é o componente principal. Os aspectos que devem ser seriamente observados em uma comunicação são ([TAF1996]):

a) fonte da transmissão (transmissor);
b) informação a ser transmitida;
c) canal ou meio de transmissão;
d) destino da informação transmitida (receptor).

A transferência de informação entre dois pontos indica a existência de um transmissor (fonte) e um receptor (destino). Nesses pontos, pode-se ter, por exemplo, pessoas ou equipamentos, que se intercomunicam utilizando a mesma linguagem de comunicação. No caso de equipamentos, a linguagem utilizada entre ambos, fonte e destino, é chamada de protocolo ([SOU1996]).

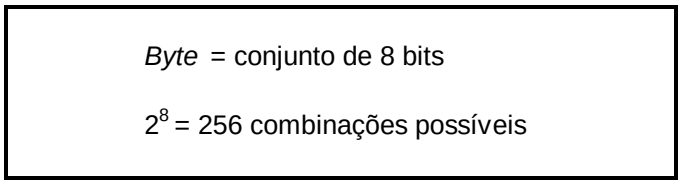
2.1 REPRESENTAÇÃO DOS DADOS

Em processamento de dados, o sistema empregado para representar os dados é o sistema binário, que, em linhas gerais, é a condição de *bit* (*binary digit*) ligado ou desligado (“0” e “1”). Desta forma através da combinação “0” ou “1”, é possível representar qualquer caracter alfanumérico (letra, número ou símbolo) com significativa eficiência e economia de tamanho com relação ao sistema decimal ([ALD2000]).

Como a essência dos códigos é sempre base dois (“0” e “1”), pode-se afirmar que a quantidade de *bits* usados determinará a quantidade de combinações possíveis. A representação matemática para esta fórmula é: “ $\mathbf{2^n = Q}$ ”, onde “**n**” é o número de *bits* usados e “**Q**” é o número de combinações possíveis ([ALD2000][TAF1996]).

Usando o sistema de *bits* para representar letras e números, criou-se o conceito de *byte*, que agrupa o total de 8 *bits* (fig. 3).

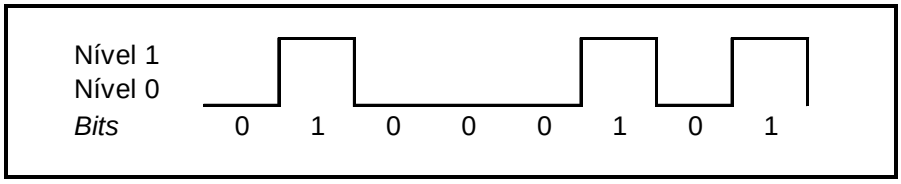
FIGURA 3 – REPRESENTAÇÃO DO CÓDIGO



Fonte: [TAF1996]

Acompanhando o raciocínio, cada *bit* pode apresentar duas posições possíveis, logo um *byte* apresenta 2^8 estados e, dessa maneira, teremos um total de 256 combinações possíveis em um *byte*. Esta convenção, foi padronizada com o código denominado ASCII (*American Standard Code for Information Interchange*). Além do código ASCII existe também o EBCDIC (*Extended Binary Coded Decimal Interchange Code*) que são os dois mais difundidos no ambiente de processamento de dados. Na figura 4, é mostrada uma forma de representação dos *bits* ([ALD2000][TAF1996]).

FIGURA 4 – REPRESENTAÇÃO GRÁFICA DE UM CONJUNTO DE *BITS*.

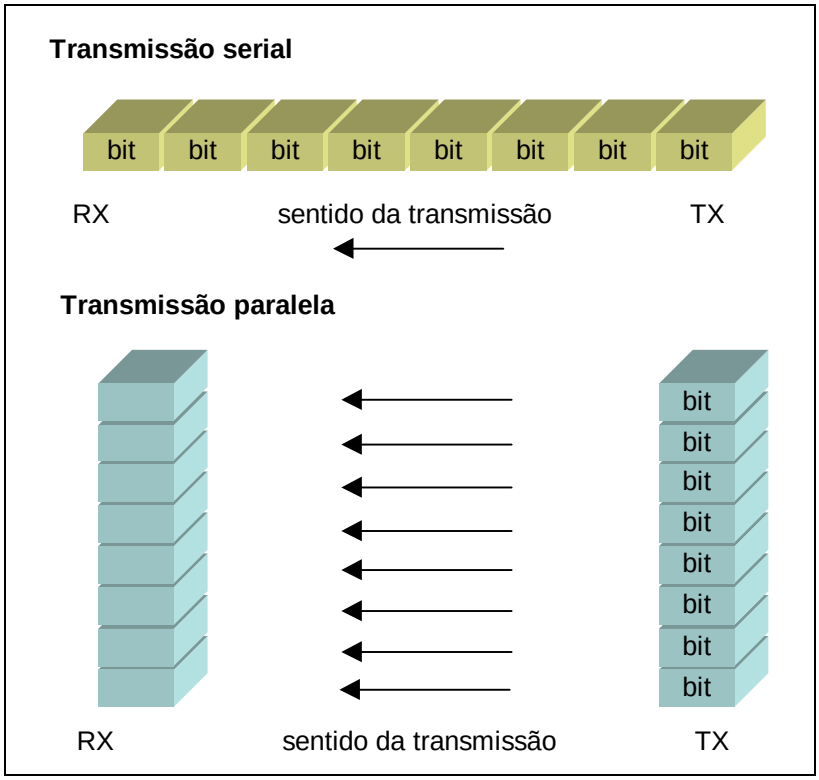


Fonte: [ALD2000]

2.2 TRANSFERÊNCIA DE DADOS

A transferência de dados entre a fonte e o destino pode ser realizada, basicamente, de duas formas: transmissão serial ou paralela (fig. 5). Na transmissão serial, os *bits*, que representam uma informação, trafegam seqüencialmente através de um meio físico. Esta é a forma de transmissão mais utilizada na comunicação de dados entre computadores. No caso da transmissão paralela, que é empregada em curtas distâncias entre computadores ou, ainda, para operações internas em um computador e na comunicação deste com alguns periféricos, os *bits*, trafegam simultaneamente, através de diversos suportes físicos em paralelo ([TAF1996]).

FIGURA 5 – TRANSMISSÃO SERIAL E PARALELA.

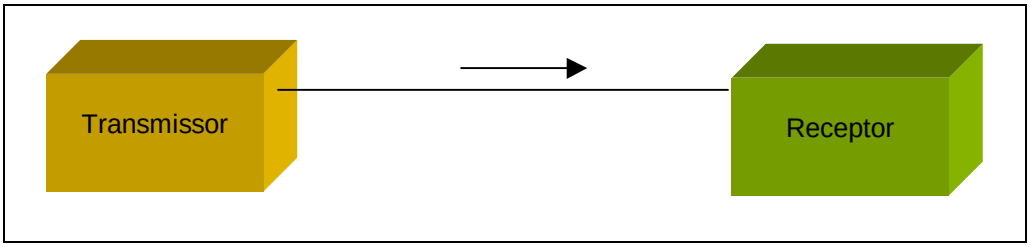


Fonte: [TAF1996]

A transmissão de informações, considerando a conexão entre dois computadores, é classificada conforme o sentido da transmissão, que determina a figura do transmissor e do receptor. Desta forma, a comunicação pode ser classificada como ([TAF1996][SOU1996][OLI1987]):

a) *simplex*: os dados são transmitidos em um único sentido (origem – destino), podendo, entretanto, existir um transmissor e vários receptores. Este tipo de comunicação é pouco utilizada, principalmente por não possibilitar que o receptor informe à origem se a informação foi ou não recebida (fig. 6).

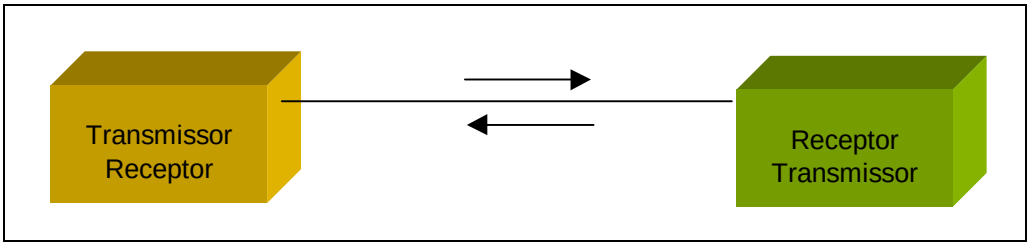
FIGURA 6 – TRANSMISSÃO *SIMPLEX*.



Fonte: [TAF1996]

b) *half-duplex*: a transmissão de dados ocorre nos dois sentidos, porém não simultaneamente. A origem e o destino dos dados alternam-se durante o processo de comunicação, caso seja necessário. Atualmente, é o mecanismo mais implementado nas aplicações de comunicação de dados (fig. 7);

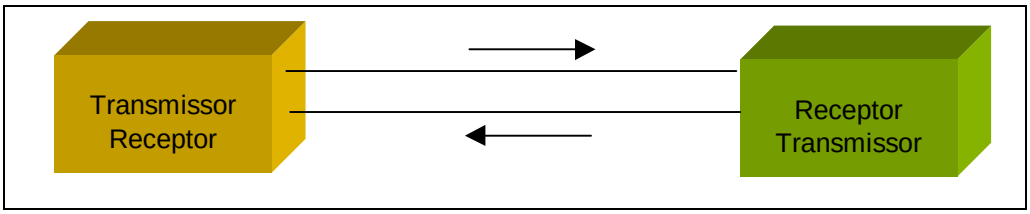
FIGURA 7 – TRANSMISSÃO *HALF-DUPLEX*.



Fonte: [TAF1996]

c) *full-duplex*: possibilita que os dados sejam transmitidos e recebidos simultaneamente, geralmente, através de dois caminhos distintos, um para cada sentido (fig. 8). No caso de modems *full-duplex*, é utilizada uma freqüência para transmissão e outra para recepção, através de um mesmo meio físico.

FIGURA 8 – TRANSMISSÃO *FULL-DUPLEX*.

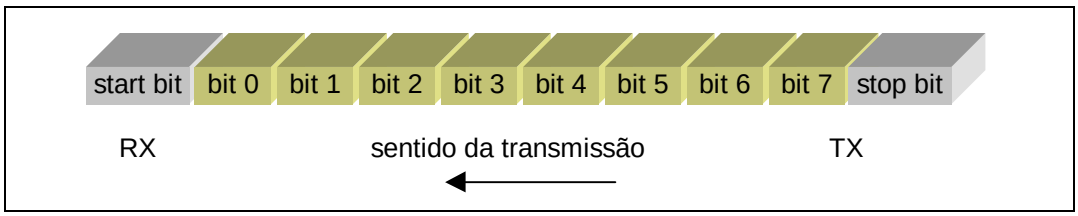


Fonte: [TAF1996]

2.2.1 TRANSMISSÃO ASSÍNCRONA

É caracterizada pela possibilidade de ser iniciada a qualquer tempo, sem limitação de tamanho de mensagem (quantidade de caracteres). Nesse tipo de transmissão, cada caracter (independente do código adotado) recebe *bits* adicionais, que indicarão o início e o fim dos mesmos, chamados *start bit* (adicionado antes do caracter, para indicar início) e *stop bit*. (adicionado após o caracter, para indicar fim), conforme demonstra a figura 9. ([OLI1987][TAF1996]).

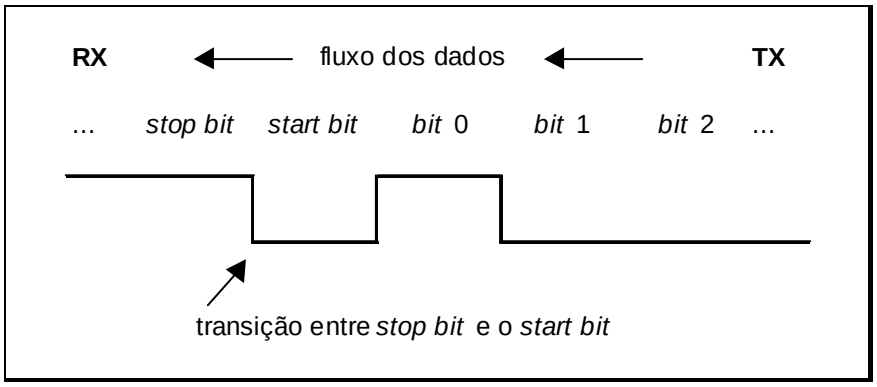
FIGURA 9 – TRANSMISSÃO SERIAL ASSÍNCRONA.



Fonte: [TAF1996]

Quando há a transição entre um sinal de *stop* e outro de *start* (fig. 10), o receptor percebe o início de um novo caracter e inicia a contagem do número de *bits* (“1”s ou “0”s), o que depende do código adotado. Quando conta o número de *bits* correspondentes, o que deve coincidir com a chegada do próximo *stop*, considera o caracter como recebido. ([OLI1987][TAF1996]).

FIGURA 10 – TRANSIÇÃO ENTRE UM SINAL *STOP* E OUTRO *START*.

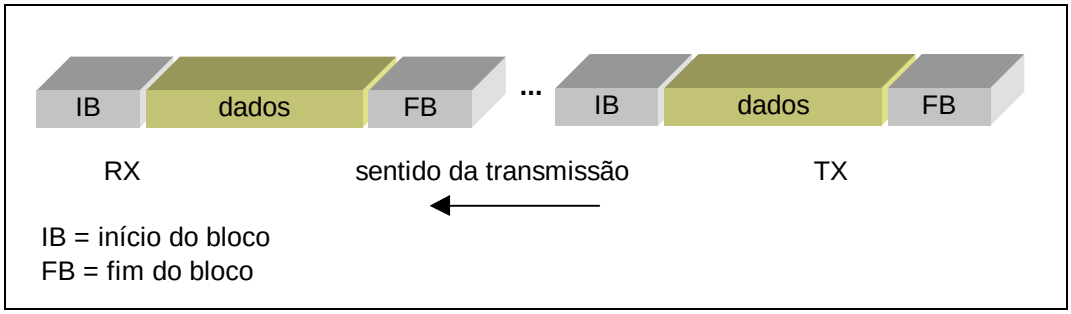


Fonte: [TAF1996]

2.2.2 TRANSMISSÃO SÍNCRONA

Sua principal característica é a possibilidade de transmitir blocos de dados com a adição de controles no começo e no fim destes blocos (fig. 11). Este modo de transmissão baseia-se no estabelecimento de uma cadência (*clock*) fixa para o transmissão dos *bits*. Esta cadência deve ser compatibilizada pelos dois elementos envolvidos no processo, o transmissor e o receptor. Mesmo não havendo dados a serem transmitidos, o transmissor envia caracteres especiais para manter o sincronismo ([TAF1996][SOU1996]).

FIGURA 11 – TRANSMISSÃO SERIAL SÍNCRONA.



Fonte: [TAF1996]

2.3 MEIOS DE TRANSMISSÃO

O meio de transmissão é, basicamente, um suporte físico por onde as informações trafegam, durante o processo de comunicação. Os meios físicos mais implementados como suporte para a transmissão são ([TAF1996][SOU1996]):

a) par trançado;
b) cabo coaxial;
c) fibra óptica;
d) rádio-freqüência.

2.3.1 PAR TRANÇADO

O par trançado é o meio de transmissão mais antigo e ainda mais usado para aplicações de comunicações. Consiste em dois fios idênticos de cobre, enrolados em espiral, cobertos por um material isolante, tendo ambos a mesma impedância para a terra, sendo desse modo um meio equilibrado. Essa característica ajuda a diminuir a susceptibilidade do cabo a ruídos de cabos vizinhos e de fontes externas por toda sua extensão ([SPE1997]).

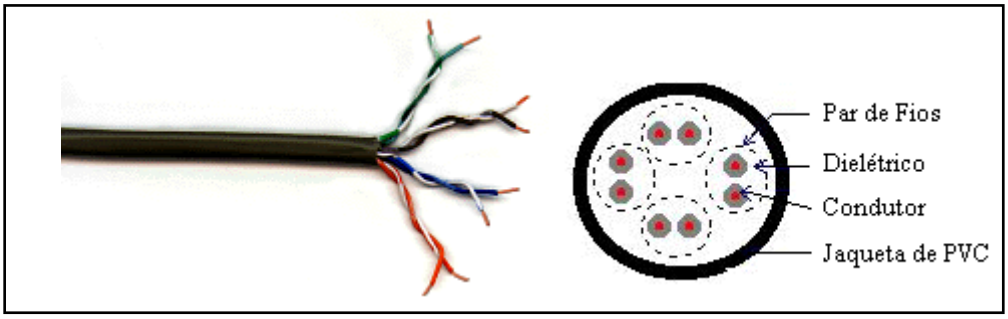
O par trançado utilizado em redes locais é chamado de cabo TP (*twisted pair*). Sua principal vantagem, em relação a outros, é seu baixo custo, entretanto sofre influências eletromagnéticas e sua atenuação é diretamente proporcional à distância ([SPE1997] [SOU1996][TAF1996]).

Existem dois tipos de par trançado: par trançado sem blindagem (UTP – *unshielded twisted pair*) e par trançado blindado (STP – *shielded twisted pair*) ([SPE1997]).

2.3.1.1 PAR TRANÇADO NÃO BLINDADO

O cabo de par trançado não blindado ou UTP (fig. 12), normalmente combina quatro tipos de pares de fios dentro da mesma capa externa. Cada par é trançado com um número diferente de voltas por polegada. O trançamento cancela o ruído elétrico dos pares adjacentes, e tende a evitar o de outros dispositivos existentes no ambiente, como motores, relês e transformadores ([DER1993]).

FIGURA 12 – PAR TRANÇADO NÃO BLINDADO.



Fonte: [SPE1997]

Os UTPs são divididos em cinco categorias (tab. 1), levando em conta o nível de segurança e a bitola do fio, onde os números maiores indicam fios com diâmetros menores.

TABELA 1 – CATEGORIA DOS CABOS.

Tipo	**Uso**
Categora 1	Voz (Cabo telefônico)
Categora 2	Dados a 4 Mpbs (*LocalTalk*)
Categora 3	Transmissão até 16Mhz. Dados a 10 Mbps (*Ethernet*)
Categora 4	Transmissão até 20Mhz. Dados a 20 Mbps (16 Mbps *Token Ring*)
Categora 5	Transmissão até 100Mhz. Dados a 100 Mbps (*Fast Ethernet*)

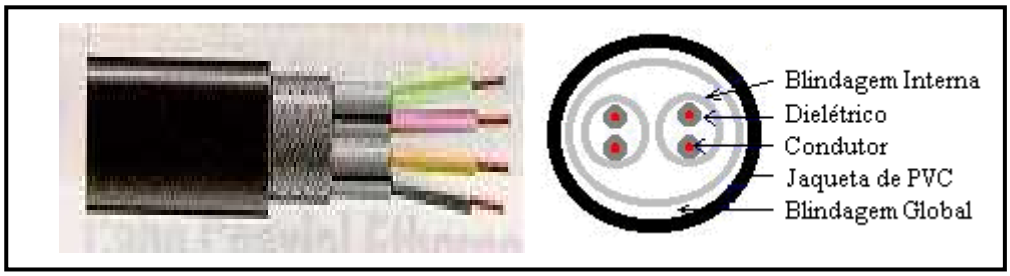
Fonte: [SPE1997]

2.3.1.2 PAR TRANÇADO BLINDADO

Este tipo de cabo possui uma blindagem interna envolvendo cada par trançado que compõe o cabo (fig. 13), cujo objetivo é reduzir o ruído. Um cabo STP geralmente possui dois

pares trançados blindados, uma impedância característica de 150 Ohms e pode alcançar uma largura de banda de 300Mhz em 100 metros de cabo ([SPE1997]).

FIGURA 13 – PAR TRANÇADO BLINDADO.

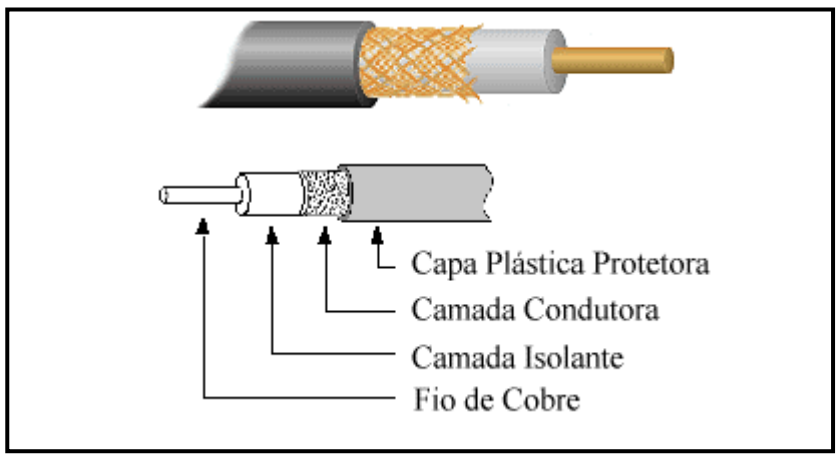


Fonte: [SPE1997]

2.3.2 CABO COAXIAL

É formado por um condutor cilíndrico, possuindo uma malha externa condutora que também serve para protegê-lo de induções eletromagnéticas externas. É considerado um meio de transmissão bastante seguro, caracterizado pelo alto grau de imunidade a ruídos externos. Esta característica fez o cabo coaxial, mostrado na figura 14, ser um meio de transmissão muito utilizado na comunicação de dados, nas redes locais de computadores, bem como redes de TV a cabo ([TAF1996][SOU1996]).

FIGURA 14 – CABO COAXIAL.

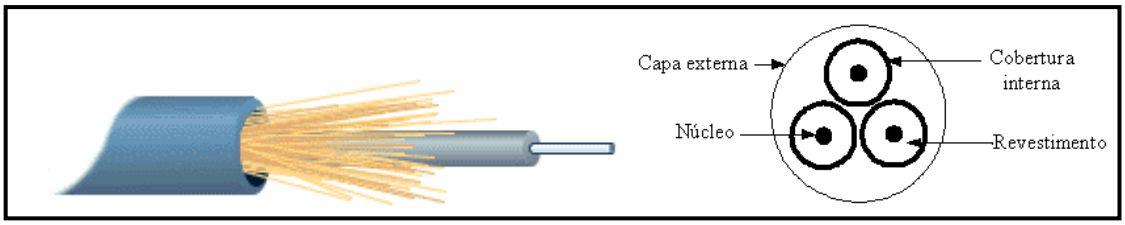


Fonte: [SPE1997]

2.3.3 FIBRA ÓPTICA

Os cabos de fibras ópticas (fig. 15), são condutores de luz infravermelha na freqüência de 10^{12} até 10^{15} Hz, portanto, estes não são afetados por correntes elétricas externas. Cada cabo é composto por dois fios de vidro em capas separadas, pois cada fio passa sinais apenas em uma direção. Cada capa tem um grupo de fibras de Kevlar, para dar resistência, e uma camada de plástico, que serve como reforço, a redor do fio de vidro ([DER1993]).

FIGURA 15 – FIBRA ÓPTICA.



Fonte: [SPE1997]

Os conectores especiais fazem uma conexão opticamente pura com as fibras de vidro e fornecem uma interface para transmissores laser e receptores ópticos. Forma-se assim, um sistema que é composto por um conversor de sinais elétricos para sinais ópticos (luminosos), e vice-versa. Embora a fibra óptica só transmita em um sentido em determinado instante, as taxas de transmissão são bastante altas. A atenuação acontece, ou por dispersão, ou por absorção de luz, logo, a qualidade do material refletor, empregado na sua fabricação, é fundamental para o bom desempenho da transmissão de sinais, consequentemente das informações ([TAF1996][SOU1996][DER1993]).

2.3.4 RÁDIO-FREQÜÊNCIA

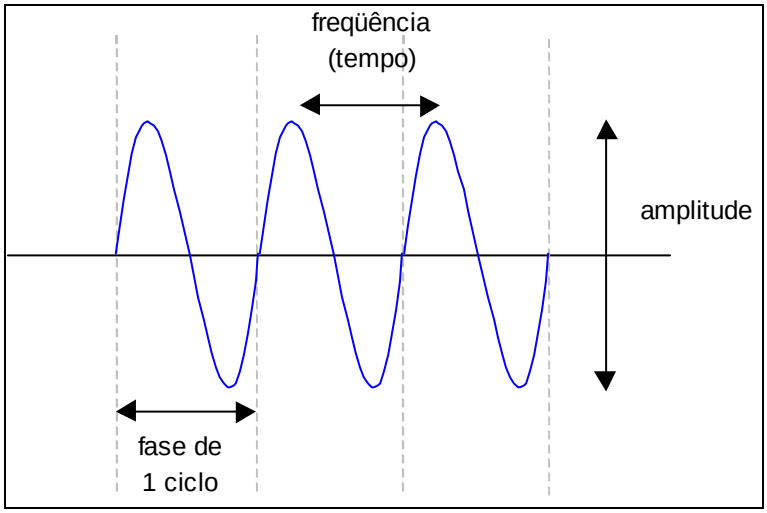
Consiste na transmissão de dados por ondas de rádio-freqüência. A tecnologia de rádio-freqüência, apesar de ser bem conhecida, passou a ser utilizada recentemente como meio de transmissão entre computadores. O uso deste recurso é possível uma vez que o sinal digital seja convertido para sinal de rádio. Para que ocorra a transmissão de forma satisfatória, o sinal deve ser transmitido com potência suficiente para ser recuperado pelo receptor, e com o mínimo de distorções ([ALD2000][TAF1996]).

Maiores detalhes sobre este segmento são abordados adiante no capítulo 3, onde é tratado um assunto específico de comunicação de dados sem fio (*wireless*).

2.4 MODULAÇÃO

O transporte dos sinais elétricos digitais que representam os dados codificados em *bits* “0” e “1” é feito por uma onda analógica (fig. 16) chamada **portadora**, em cima da qual viaja o sinal a ser transmitido, que possui uma determinada faixa de freqüência ([SOU1999]). Os sinais (onda quadrada) são colocados na onda portadora analógica por um processo chamado modulação ([ALD2000][SOU1999]).

FIGURA 16 – ONDA ANALÓGICA.



Fonte: [SOU1999]

É interessante notar que muitas formas não elétricas de comunicação, também envolvem um processo de modulação, como a fala, por exemplo. Quando uma pessoa fala, os movimentos da boca são realizados a taxas de freqüências baixas, na ordem de 10Hz, não podendo a esta freqüência produzir ondas acústicas propagáveis. A transmissão de voz através do ar é conseguida pela geração de tons portadores de alta freqüência nas cordas vocais, modulando estes tons com as ações musculares da cavidade bucal. O que o ouvido interpreta como fala, é portanto, uma onda acústica modulada, similar, em muitos aspectos, a uma onda elétrica modulada ([ALD2000]).

2.4.1 TIPOS DE MODULAÇÃO

Em grande parte, o êxito de um sistema de comunicação depende da modulação, de modo que a escolha do tipo de modulação é uma decisão fundamental em projetos de sistemas para transmissão de sinais.

Muitas diferenças técnicas de modulação são utilizadas para satisfazer as especificações e requisitos de um sistema de comunicação. Independente do tipo de modulação utilizada, o processo de modulação deve ser reversível de modo que a mensagem possa ser recuperada no receptor pela operação complementar da modulação ([ALD2000]).

A princípio, é possível identificar dois tipos básicos de modulação, de acordo com o tratamento da portadora pelo sinal que se deseja transmitir ([ALD2000]):

a) modulação analógica;
b) modulação digital.

Ambos são utilizados nos sistemas de comunicação conforme o tipo de sinal que se quer transmitir. Os dois tipos mencionados acima se subdividem em subtipos de acordo com as necessidades e requisitos do projeto.

2.4.2 MODULAÇÃO ANALÓGICA

Também classificada como modulação de onda contínua (CW – *Continous Wave*), na qual a portadora é uma onda senoidal, e o sinal modulante é um sinal analógico ou contínuo. Há um número infinito de forma de ondas possíveis que podem ser formadas por sinais contínuos. Tratando-se de um processo contínuo, a modulação CW é conveniente para este tipo de sinal. Em modulação analógica, o parâmetro modulado varia em proporção direta ao sinal modulante ([ALD2000]).

As técnicas de modulação para sinais analógicos, mais utilizados são ([ALD2000]):

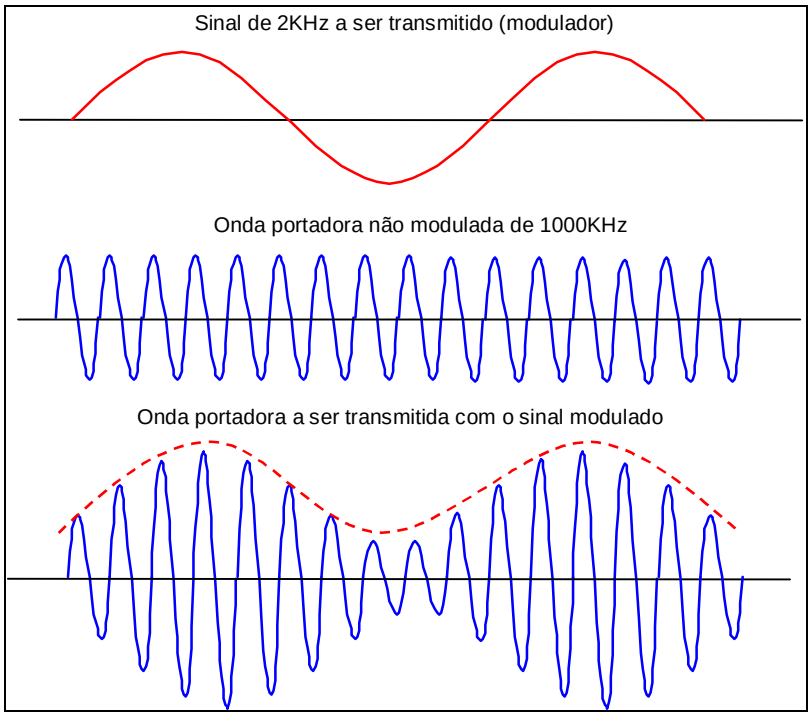
a) modulação em amplitude (AM);
b) modulação em freqüência (FM).

2.4.2.1 MODULAÇÃO EM AMPLITUDE

Na modulação em amplitude, existe uma onda portadora de freqüência fixa que transporta o sinal a ser transmitido, representando-o pela variação de sua amplitude ([SOU1999]).

O exemplo mais típico de modulação em amplitude são as transmissões de rádio AM, onde, por exemplo, uma freqüência portadora de 1000KHz transportando o sinal de voz que tem uma faixa de variação de 5KHz. O sinal de rádio-freqüência de 1000KHz é modulado em amplitude por um sinal de áudio de 2KHz. como exemplificado na figura 17 ([SOU1999]).

FIGURA 17 – VARIAÇÃO EM AMPLITUDE DA PORTADORA DE ACORDO COM O SINAL A SER TRANSMITIDO.



Fonte: [SOU1999]

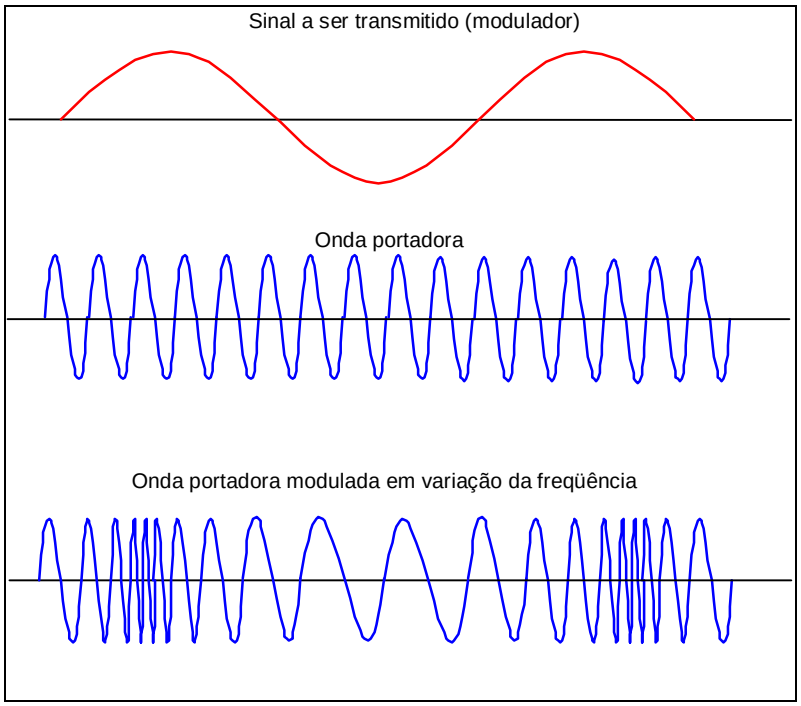
2.4.2.2 MODULAÇÃO EM FREQÜÊNCIA

Na modulação em freqüência, ao invés de variar a amplitude da onda portadora, varia a sua freqüência. A amplitude se mantém constante e a freqüência sofre deslocamentos para mais e para menos, proporcionalmente à amplitude do sinal modulador ([SOU1999]).

Este é o princípio utilizado nas transmissões de rádio FM. A modulação em freqüência permite maior qualidade na transmissão dos sinais, em relação a AM, devido a transmitir uma faixa maior de freqüências. Uma transmissão de rádio modulada em FM consegue transmitir sinais de áudio de 0Hz a 25000Hz (25KHz), ou seja, praticamente todas as freqüências audíveis ([SOU1999]).

No exemplo a seguir (fig.18), é demonstrado a modulação FM em que a freqüência da portadora aumenta com o aumento da intensidade do sinal.

FIGURA 18 – MODULAÇÃO EM FREQÜÊNCIA.



Fonte: [SOU1999]

2.4.3 MODULAÇÃO DIGITAL

Também conhecida como modulação discreta ou codificada. Utilizada em casos em que se está interessado em transmitir uma forma de onda ou mensagem, que um conjunto finito de valores discretos representando um código. No caso da comunicação binária, as mensagens são transmitidas por dois símbolos apenas. Um dos símbolos representado por um pulso correspondendo ao valor binário “1” e o outro pela ausência do pulso (nenhum sinal) representando o dígito binário “0” ([ALD2000]).

Nos sistemas digitais, o problema da modulação, é um problema um pouco mais simples que nos sistema contínuos. Durante a transmissão, as formas de onda da onda portadora modulada são alteradas pelo ruído do canal. Quando este sinal é recebido no receptor, deve-se decidir qual das duas formas de onda possíveis conhecidas foi transmitida. Uma vez tomada a decisão, a forma de onda original é recuperada sem nenhum ruído ([ALD2000]).

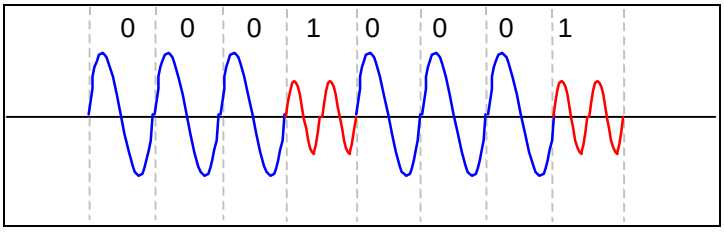
Do mesmo modo que há diversas técnicas de modulação para sinais analógicos, as informações digitais podem ser colocadas sobre a portadora de diferentes modos. As técnicas de modulação para sinais digitais mais utilizadas atualmente são ([ALD2000]):

a) modulação em amplitude por chaveamento (ASK);
b) modulação em freqüência por chaveamento (FSK);
c) modulação em fase por chaveamento (PSK).

2.4.3.1 MODULAÇÃO EM AMPLITUDE POR CHAVEAMENTO

É um método de modular um sinal de dados mudando a amplitude da onda portadora. Uma amplitude maior da portadora representa o bit “1” e uma amplitude menor representa o bit “0” (fig. 19). Esse tipo de modulação é muito sujeito a erros de transmissão, pois qualquer variação ou ruído no meio de uma transmissão pode deteriorar a amplitude do sinal e confundir a representação ([SOU1999]).

FIGURA 19 – MODULAÇÃO EM AMPLITUDE POR CHAVEAMENTO (ASK).

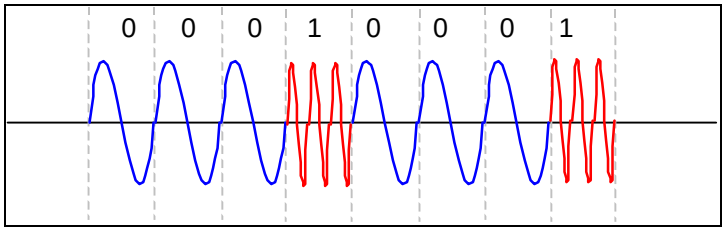


Fonte: [SOU1999]

2.4.3.2 MODULAÇÃO EM FREQÜÊNCIA POR CHAVEAMENTO

A modulação em freqüência para a transmissão de dados é chamada de FSK (*Frequency Shift Keying*) e utiliza duas freqüências para representar o bit “0” e o bit “1” (fig. 20) ([SOU1999]).

FIGURA 20 – MODULAÇÃO EM FREQÜÊNCIA POR CHAVEAMENTO (FSK).

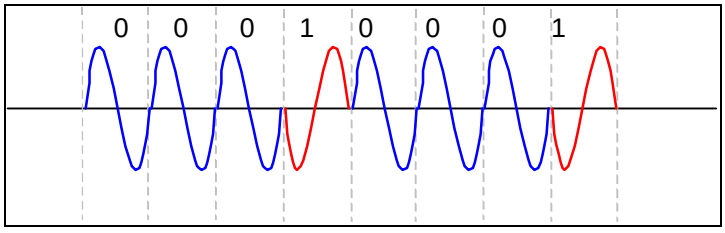


Fonte: [SOU1999]

2.4.3.3 MODULAÇÃO EM FASE POR CHAVEAMENTO

Este é o tipo de modulação mais utilizado para a transmissão de dados. Nele, a mudança de *bit* “0” para “1” e vice-versa é feita pela mudança de fase da portadora (fig. 21) ([SOU1999]).

FIGURA 21 – MODULAÇÃO EM FASE POR CHAVEAMENTO (PSK).



Fonte: [SOU1999]

No exemplo da figura anterior (fig. 21), a modulação dá apenas em duas fases. O equipamento que gera a freqüência da portadora possui dois osciladores geradores de freqüências, em que uma freqüência está defasada 180° da outra. Quando um *bit* passa de “0” para “1” ou vice-versa, o oscilador em operação é desligado e ligado o outro em defasagem em 180° em relação ao anterior ([SOU1999]).

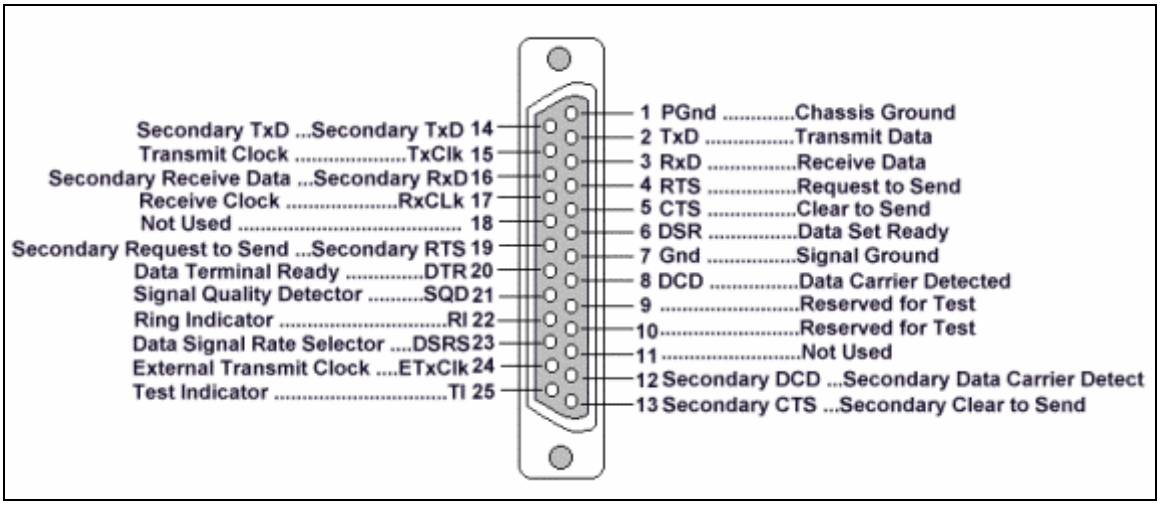
2.5 INTERFACE PADRÃO RS-232C

A interface RS-232C foi elaborada pela EIA (*Eletronic Industries Association*) e estabelece um padrão para o interfaceamento serial com o computador. Os sinais trocados entre os dois equipamentos são normatizados pela CCITT através da recomendação V24, onde são especificados conectores, funções dos pinos e voltagens utilizadas para conectar dois dispositivos de maneira que possam enviar e receber dados ([TAF1996][TAN1994]).

Em 1991, a EIA juntamente com a TIA (*Telecommunications Industry Association*) determinaram uma nova versão para a interface, cujo nome passou para EIA/TIA-232-E. No entanto, muitas pessoas, ainda referenciam a interface como RS-232C ou apenas, RS-232. ([SMC2000]).

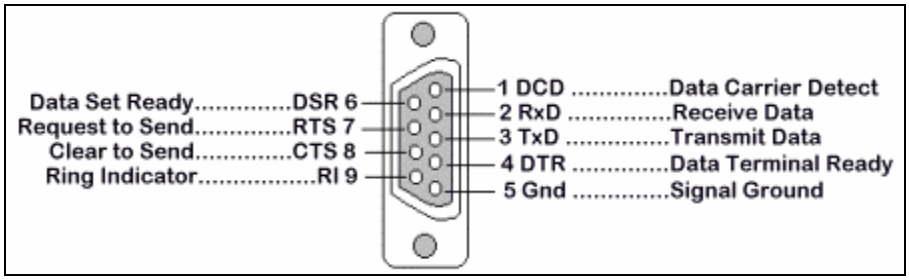
Os sinais trocados pela interface, estão especificados em um conector de 25 pinos tipo "D" (fig. 22). A conexão mecânica é padronizada pela ISO (*International Standard Organization*), através da norma ISO 2593, e realizada através de um conector padrão denominado DB-25P ([TAF1996]). O conector RS-232C pode ter 25 ou 9 pinos. A vantagem do conector de 9 pinos (fig. 23) é que ocupa menos espaço ([SOU1999]), porém é limitado a transmissões tipo assíncronas, devido ausência dos pinos de controle de cadência (*clock*).

FIGURA 22 – CONECTOR DB-25.



Fonte: [SMC2000]

FIGURA 23 – CONECTOR DB-9.



Fonte: [SMC2000]

2.5.1 USART

A USART (*Universal Synchronous Asynchronouos Receiver Transmitter*) é um circuito de E/S (entrada/saída) de dados que faz a interface entre o barramento do computador e a porta serial, efetuando a comunicação entre o computador e o modem em nível físico. A USART é basicamente um *chip* (circuito integrado) que compõe a interface serial. Todos os dados transmitidos do micro para a porta serial, ou vice-versa, passam pela USART, a qual converte os *bytes* em *bits* transmitidos serialmente ([SOU1999]).

Este chip poupa ao programa a tarefa de realizar, através do processador, a transmissão e a recepção de caracteres (*bytes*) pelo canal serial. Para receber e enviar os dados, o programa, simplesmente, lê e escreve *bytes* na USART, que se mostra aos “olhos” do processador, apenas como uma região de memória, ou portas de E/S (endereços) ([TAF19996]).

3 COMUNICAÇÃO DE DADOS SEM FIO (*WIRELESS*)

Muitos sistemas de comunicação fazem a transmissão dos dados utilizando fios de cobre, como par trançado e coaxial, ou fibra óptica. Outros entretanto, transmitem os dados pelo ar, não utilizando qualquer tipo de meio físico. Esse tipo de comunicação é conhecida como comunicação *wireless*.

Comunicação *wireless* se refere a todo tipo de conexão efetuado sem fios como a transmissão de dados via rádio digital, redes locais sem cabeamento físico que utilizam infravermelho ou freqüências de microondas para conexão entre seus nós, sistemas de *paging* e *trucking* via rádio, telefonia celular e outros ([SOU1999]).

Na realidade, o ar (ou espaço livre) constitui-se de um meio natural para a propagação de sinais eletromagnéticos, podem talvez, ser considerado o melhor suporte de transmissão, quando se fala em conectividade. Tal afirmação baseia-se no fato de que o ar provê uma interconexão completa e permite uma grande flexibilidade na localização das estações remotas.

3.1 O ESPECTRO ELETROMAGNÉTICO

Todas as ondas eletromagnéticas são idênticas e apresentam uma independência com relação à existência ou não de um meio de propagação, independentemente de suas características físicas, como sua intensidade, comprimento de onda, freqüência, energia, polarização, etc. ([FON1998]).

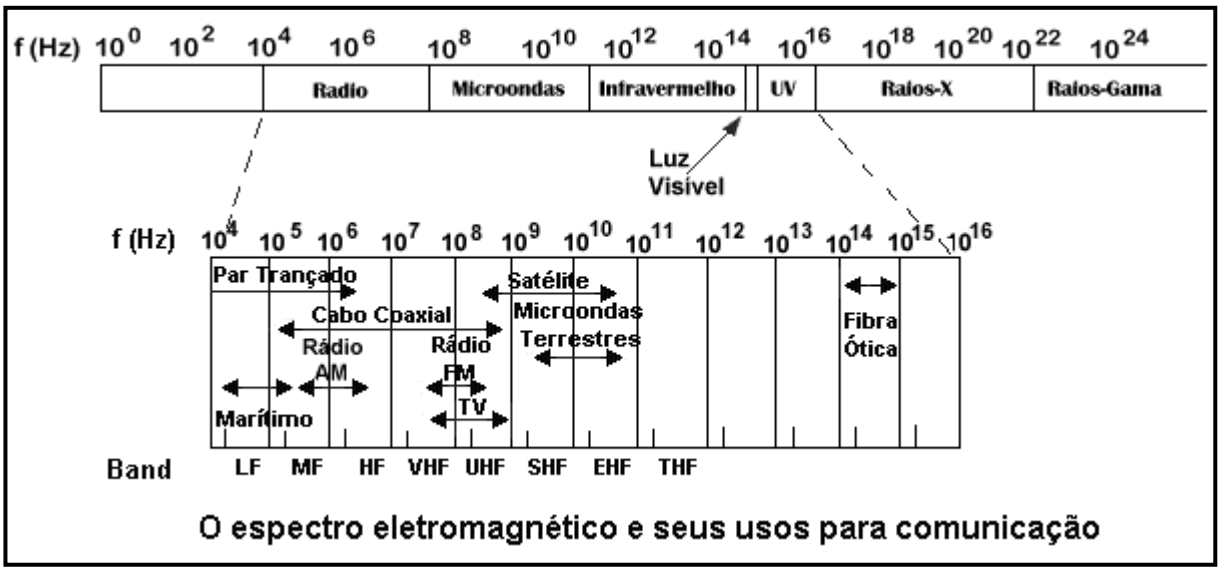
A velocidade de propagação da onda eletromagnética no vácuo é a velocidade da luz, ou seja, 300.000 Km/segundo. O número de ondas que passam por um ponto do espaço num determinado tempo, define a freqüência de radiação, que é diretamente proporcional à velocidade de propagação da radiação. Quando maior a velocidade de propagação da onda, maior o número de ondas que passarão por um ponto num dado tempo e maior será sua freqüência. ([FON1998]).

A faixa de comprimentos de onda ou freqüências que se pode encontrar a radiação eletromagnética é ilimitada. Com a tecnologia atualmente disponível, pode-se gerar ou

detectar a radiação eletromagnética numa extensa faixa de freqüência, que se estende de 1 a 10^{24} Hz, ou comprimentos de onda na faixa de 108 metros a 0.01^{A} (FON1998]).

Este espectro é subdividido em faixas (fig. 24), representando regiões, onde dependendo da região, trabalha-se com energia (*eletron-volt*), comprimentos de onda (micrômetros), ou freqüência (*hertz*) ([VAR1999]).

FIGURA 24 – FAIXAS DO ESPECTRO ELETROMAGNÉTICO.



Fonte: [FON1998]

A característica de propagação das ondas eletromagnéticas em sistemas de comunicações que utilizam a atmosfera como canal de transmissão, é altamente dependente da freqüência de operação ([VAR1999]). Na tabela a seguir (tab. 2), consta um detalhamento maior do espectro eletromagnético e a denominação de cada faixa de freqüência, assim como também o tipo de transmissão em que é utilizada a faixa ([SOU1999]).

TABELA 2 – CLASSIFICAÇÃO DE FREQÜÊNCIA.

Classificação	**Nome Popular**	**Freqüências**	**Utilização**
ELF – Extremely Low Frequency	Ondas longas	300Hz a 10000Hz	Sonares
VLF – Very Low Frequency	Ondas longas	10Khz a 30Khz	Sonares
LF – Low Frequency	Ondas longas	30Khz a 300Khz	Navegação aérea, radiodifusão
MF – Medium Frequency	Ondas médias	300Khz a 3000Khz	Navegação aérea, radiodifusão
HF – High Frequency	Ondas curtas	3Mhz a 30Mhz	Radiodifusão, comunicação marítima
VHF – Very High Frequency	–	30Mhz a 300Mhz	TV, FM, radioamadores
UHF – Ultra High Frequency	Microondas	300Mhz a 3000Mhz	Comunicações públicas e privadas
SHF – Super High Frequency	Microondas	3Ghz a 30Ghz	Comunicações públicas e privadas
EHF – Extremely High Frequency	Microondas	30Ghz a 300Ghz	Comunicações públicas e privadas
Região experimental	–	300Ghz a 1000Ghz	Comunicações públicas e privadas

Fonte: [SOU1999]

3.1.1 RUÍDOS

Ruído é a denominação para sinais elétricos aleatórios e imprevisíveis provenientes de causas naturais tanto externas como internas ao sistema. Quando estes sinais são adicionados

a um sinal que contém informação, esta pode ser parcialmente mascarada ou totalmente eliminada. O ruído é difícil de ser completamente eliminado, constituindo um dos problemas básicos da comunicação elétrica ([ADL2000]).

3.2 TRANSMISSÃO POR RÁDIO-FREQÜÊNCIA

Enquanto sistemas de comunicações convencionais dependem das conexões a cabo, os sistemas *wireless* via rádio utilizam sinais de rádio como meio de transmissão, ou seja, os dados são enviados pelo ar em ondas de rádio ([BLA2000]).

Inicialmente desenvolvidas durante a Segunda Guerra Mundial para transmitir informações seguras via ondas de rádio. O trabalho com sistemas comerciais só começou depois da década de 70. Em meados dos anos 80, o FCC, órgão regulador de telecomunicações dos E.U.A, autorizou o uso de três faixas de rádio-freqüência para finalidades industriais, científicas e médicas sem necessidade de concessão. Isso levou a um novo interesse na tecnologia e os primeiros produtos efetivos para comunicação sem fio apareceram por volta de 1990 ([BLA2000]).

A comunicação via rádio é simples de operar, oferece total mobilidade ao usuário e permite conexão de equipamentos distantes quilômetros entre si, permitindo ao usuário a implantação de uma rede de comunicação de dados completa. Além disso, ondas de rádio são mais simples de serem geradas, podem ir a longas distâncias e atravessar prédios e casas, sendo por isso largamente utilizada em telecomunicações ([BLA2000][FON1998]).

As ondas de rádio são dependentes de sua freqüência, as ondas de baixas freqüências atravessam facilmente obstáculos, mas sua força cai rapidamente conforme a distância do emissor aumenta. Já ondas de alta freqüências viajam em linha reta e ricocheteiam nos obstáculos, porém são absorvidas pela chuva, atingem a ionosfera e são refletidas de volta à terra, podendo sobre certas condições meteorológicas ser refletidas diversas vezes indo a longas distâncias. Sua habilidade de viajar a longas distâncias causa um problema: a interferência entre usuários usando a mesma freqüência. Devido a isto todos os governos regulam rigidamente a licença para uso de rádios transmissores ([BLA2000][FON1998]).

3.2.1 SISTEMA ALOHA POR RADIODIFUSÃO

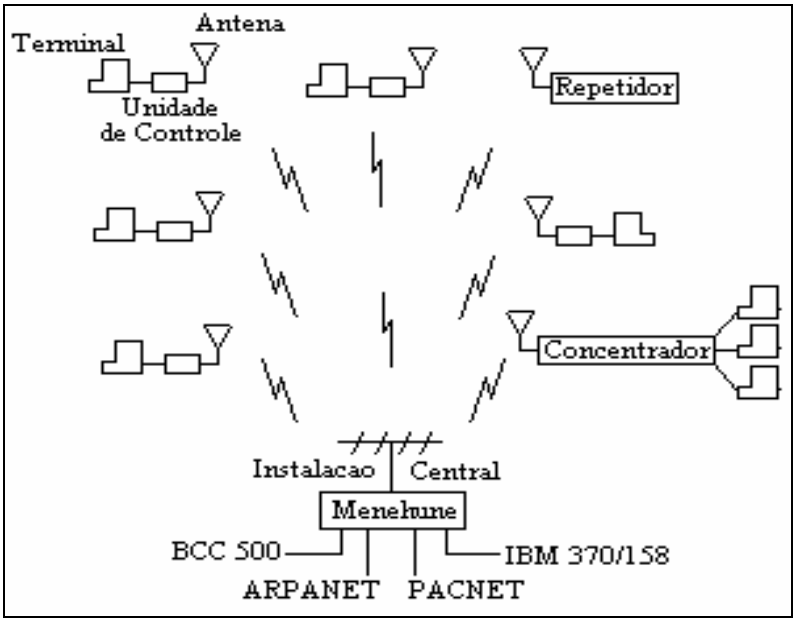
O sistema ALOHA da Universidade do Havaí foi o primeiro sistema de computadores a empregar a técnica de radiodifusão ao invés de cabos ponto a ponto. Na década de 70, quando o projeto foi implantado, as linhas telefônicas disponíveis na ocasião eram caras e pouco confiáveis. Havia a necessidade de interligação de sub-redes da universidade, que estavam espalhadas em quatro ilhas, ao Centro de Computação principal em Oahu ([TAN1994]).

A comunicação foi realizada através da instalação, em cada estação, de um pequeno transceptor de rádio FM, com um alcance suficiente (30Km) para comunicar-se com o transceptor do centro de computação. Mais tarde foi introduzido um repetidor mais potente, aumentando o alcance, teórico, para 500Km ([TAN1994]).

O projeto foi realizado de forma não haver comunicação direta entre estações, apenas de uma estação para o Centro de Computação e deste para uma estação. Foram utilizadas duas faixas de freqüência: uma em 407,305MHz para o tráfego no sentido Centro-Estação, e outra em 413,475MHz para o tráfego no sentido contrário. Possui uma taxa de transmissão de 9600bps ([TAN1994]).

A figura 25 apresenta os elementos básicos do sistema ALOHA. Na instalação central há um computador (chamado Menehune) onde todos os dados, que entram ou saem, passam por ele. O Menehune está conectado ainda a outros dois computadores (BCC500 e IBM370/158) e a outras duas redes (ARPANET e PACNET). Cada estação possui uma unidade de controle que armazena dados e faz as retransmissões ([JAM1998]).

FIGURA 25 – SISTEMA ALOHA DA UNIVERSIDADE DO HAVAÍ.



Fonte: [JAM1998]

3.3 TRANSMISSÃO POR *SPREAD SPECTRUM*

A técnica *Spread Spectrum* ou espalhamento espectral é um sistema no qual a energia média do sinal transmitido é espalhada sobre uma largura de faixa que é muito mais larga do que a largura de faixa que contém a informação. Tais sistemas compensam uma maior largura de faixa de transmissão por uma menor densidade espectral de potência e uma melhora na rejeição aos sinais interferentes operando na mesma faixa de freqüência ([MIN2000]).

Esta tecnologia possui um uso crescente em aplicações para transmissões sem fios. A transmissão por ondas de rádio com propagação do espectro consegue ultrapassar obstáculos com mais eficiência do que outros tipos de transmissão sem fios, isto porque utiliza-se de freqüências menores (902MHz a 928MHz, 2400Mhz a 2483MHz, 5725MHz a 5875MHz) ([SOU1999]). Este tipo de transmissão, disponibiliza velocidades na faixa de Mbps. A potência de transmissão na faixa de 1Watt, alcança até 8Km ([SOU1999]).

Segundo [DAV2000], existem inúmeras razões para se utilizar a técnica de espalhar o espectro e muitos benefícios podem ser obtidos se isto for realizado adequadamente:

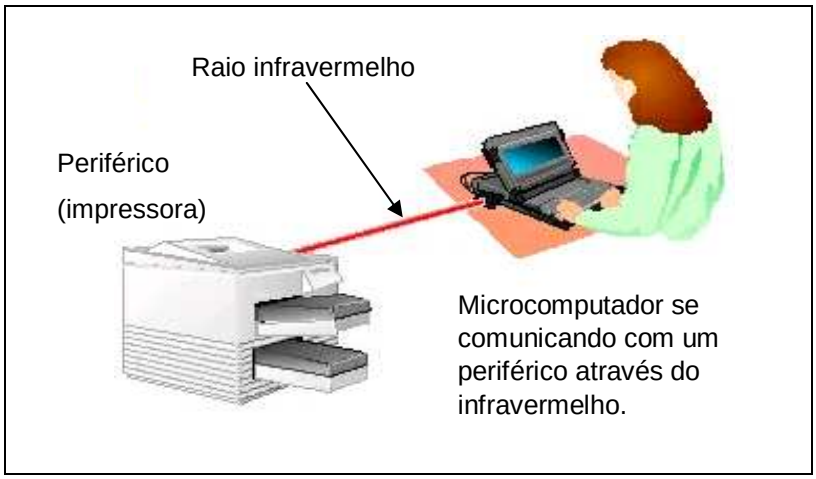
a) **rejeição a interferência intencional (*jamming*)** pois as transmissões alheias na mesma faixa de freqüência do sinal de interesse, e consequentemente descorrelacionadas deste, são fortemente atenuadas no receptor;
b) **rejeição a interferência natural** devido ao ganho de processamento inerente ao processo;
c) **baixa probabilidade de interceptação** por dois aspectos: faixa mais larga a ser monitorada e densidade de potência do sinal a ser detectado reduzida pelo processo de espalhamento de espectro;
d) possibilidade de **comunicação multiusuário de acesso aleatório por endereçamento seletivo** que consiste na transmissão de sinais de vários usuários simultaneamente no tempo, ocupando a mesma faixa de freqüência. Esta é a chamada capacidade de múltiplo acesso por divisão de código (CDMA) dos sistemas de espalhamento espectral;
e) **resistência aos efeitos dos múltiplos percursos** (quando o sinal transmitido percorre mais de um caminho até o receptor, devido a reflexões na superfície, prédios, árvores e atmosfera, produzindo interferências no sinal recebido);
f) **medidas de distância com alta resolução** em sistemas de radar que utilizam técnicas de espalhamento espectral conseguem resolução melhor que os convencionais devido ao ganho de processamento inerente e à capacidade desses sistemas distinguirem pulsos muito próximos oriundos de reflexões no mesmo alvo, recebidos por múltiplos percursos;
g) **referência de tempo universal precisa** apesar de problemas de múltiplo percurso. A distinção de pulsos muito próximos recebidos por múltiplos percursos aliado ao ganho de processamento proporcionado pelas técnicas de espalhamento espectral, torna possível a um transmissor informar o mesmo referencial de tempo (com uma precisão tanto maior quanto mais elaborado for o sistema de espalhamento espectral), a receptores que conheçam o código utilizado no espalhamento.

3.4 TRANSMISSÃO POR INFRAVERMELHO

As comunicações sem fio com transmissão por infravermelho operam usando uma luz infravermelha que transmite os dados entre os dispositivos. Aplicável a redes de uso local e

semi-móvel (fig. 26), a transmissão de dados por infravermelho converte pulsos elétricos de dados em sinais de luz, e retornando à pulsos elétricos no receptor ([SOU1996]).

FIGURA 26 – EXEMPLO DE COMUNICAÇÃO VIA RAIO INFRAVERMELHO.



Fonte: [FON1998]

A conexão para transmissão dos dados por infravermelho nas redes sem fios entre edifícios, por exemplo, é realizada através de um raio de luz infravermelho dirigido e de alta velocidade tendo um alcance máximo relativo as dimensões de um campo de futebol. A segurança na transmissão dos dados é baixa pois basta algo interromper ou desviar o raio infravermelho para que a comunicação seja interrompida. A freqüência irradiada pela transmissão infravermelha pode chegar na faixa de 100THz ([SOU1996]).

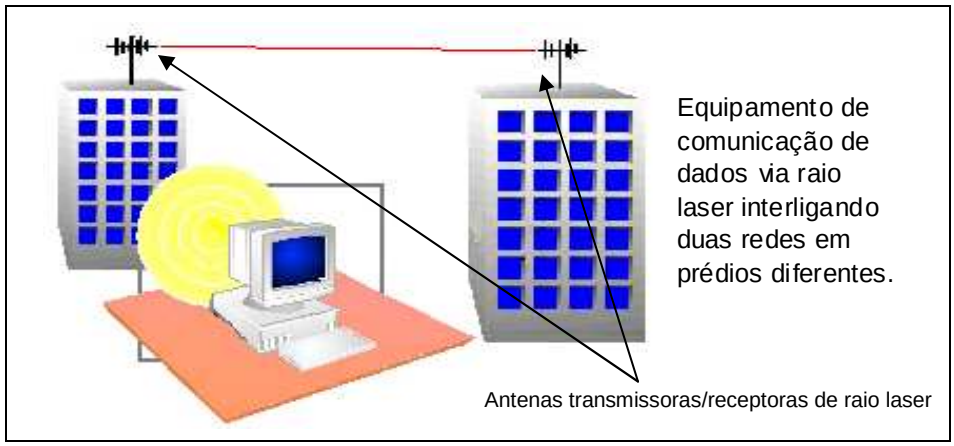
Por outro lado, as ondas infravermelhas não viajam em todas as direções mas apenas em um espaço definido. Por isso não é necessária uma licença do governo para usar sistemas infravermelhos, ao contrário dos sistemas de rádio e microondas, que devem obrigatoriamente ser licenciados.

As transmissões por infravermelho atingem velocidades maiores que os outros tipos de transmissão sem fios (16Mbps). Em contra partida, as comunicações por infravermelho possuem as desvantagens de receberem interferências da iluminação do ambiente, raios do sol, chuva, neblina e não conseguirem atravessar obstáculos como paredes ([SOU1996]).

3.5 TRANSMISSÃO POR LASER

Os sistemas a Laser (*Light Amplified Stimulated Emission Radiation*) são mais utilizados para conexões ponto-a-ponto de longa distância, como a interligação de duas LANs em prédios separados, por exemplo (fig. 27). A distância entre os pontos de conexão é um dos principais fatores que diferenciam a utilização de sistemas *wireless* Laser e sistemas *wireless* infravermelho. O primeiro, como já mencionado, é adequado à longas distâncias, enquanto o segundo é mais utilizado em ambientes internos, onde as distâncias entre os pontos de conexão são bem menores em relação às encontradas em ambientes externos ([SOU1996]).

FIGURA 27 – EXEMPLO DE COMUNICAÇÃO VIA RAIO LASER.



Fonte: [FON1998]

Este tipo de comunicação é totalmente digital e altamente direcional, o que torna o sistema imune a interferências ou grampos. Entretanto, dependendo do comprimento de onda escolhido, fenômenos como chuva ou neblina podem interferir nesta comunicação ([SOU1996]).

3.6 TRANSMISSÃO POR MICROONDAS

São sistemas de transmissão via rádio que operam na faixa de 900Mhz a 30Ghz no espectro de freqüência. Estes sistemas são usados para transportar sinais analógicos ou digitais de voz e dados. Nestas freqüências, as ondas de rádio se comportam praticamente como ondas de luz, desta forma sua propagação segue uma linha reta e podem ser

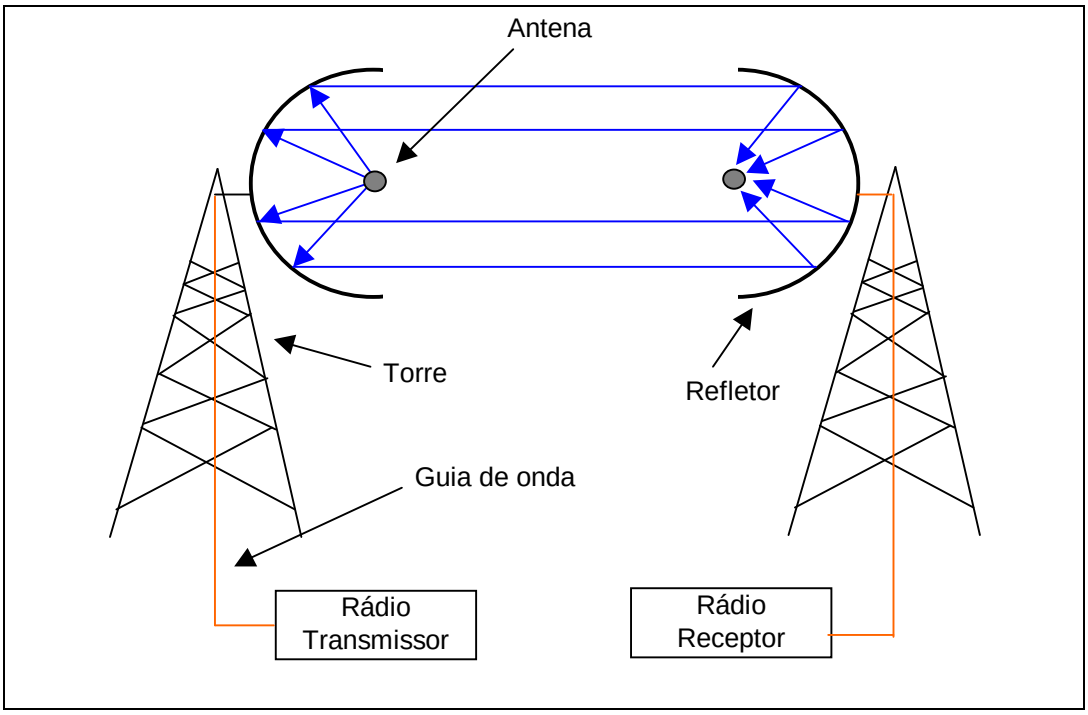
direcionadas com grande precisão, devido a isto as antenas devem estar perfeitamente alinhadas.

Como as microondas viajam em linha reta, as antenas não podem estar muito separadas entre si, ocorre o risco da terra e de obstáculos atravessarem o caminho das ondas, há portando a necessidade de repetidores.

Um sistema de transmissão por microondas (fig. 28), é composto basicamente por ([SOU1996]):

a) torre;
b) antena;
c) guia de onda;
d) rádio transceptor (transmissor e receptor).

FIGURA 28 – ANTENA EM UM SISTEMA DE TRANSMISSÃO EM MICROONDAS.



Fonte: [SOU1996]

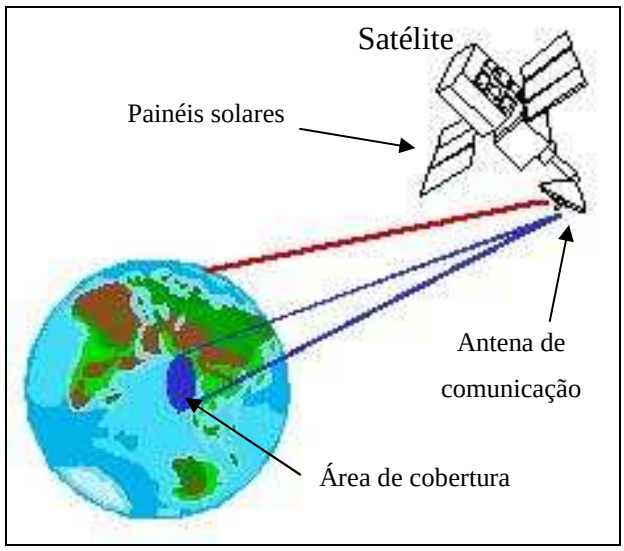
As microondas estão sujeitas a interferências por fenômenos atmosféricos e tempestades. Um dos maiores problemas da transmissão de microondas é o *multipath fading*, ou seja, uma onda pode ser refratada pelas camadas mais baixas da atmosfera e pode demorar

uma fração a mais para chegar ao receptor. Este sinal atrasado pode entrar em fase com o sinal direto e anulá-lo. As ondas são também absorvidas pela chuva. Neste caso, como no *multipath fading*, a única solução é desviar os sinais para uma rota alternativa, contornando a chuva ([SOU1996]).

3.6.1 SATÉLITES

Um satélite de comunicação é basicamente um repetidor de sinais que recebe um sinal a ser transmitido de um ponto na superfície do planeta e retransmite para uma outra região (fig. 29), ou seja, uma estação retransmissora de sinais (microondas) situada no espaço ([SOU1999]).

FIGURA 29 – SATÉLITE EM ÓRBITA TRANSMITINDO E RECEBENDO SINAIS.



Fonte: [FON1998]

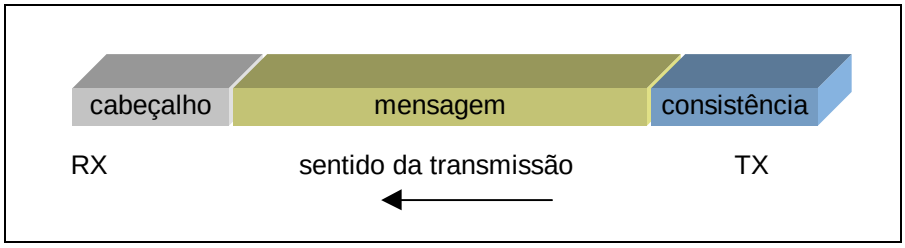
Os satélites são compostos basicamente de um sistema de comunicação (repetidor de sinais) e um sistema de navegação (comando, telemetria e controle do satélite). São energizados por baterias solares e podem retransmitir milhares de canais de voz, serviços de TV, dados, fax, GPS e outros ([SOU1999]).

4 PROTOCOLOS

O protocolo é um conjunto de regras preestabelecidas, cuja função é fazer com que a comunicação de dados entre equipamentos seja realizada com segurança e de forma ordenada. Essas regras obedecem a uma seqüência lógica e padronizada. O formato destes protocolos (fig. 30), geralmente, obedece a seguinte forma ([OLI1987][PEN1988][TAF1996]):

a) cabeçalho: contém as informações de controle do pacote;
b) mensagem: este campo é formado pelas mensagem que serão transmitidas;
c) consistência: são caracteres para verificação de erros.

FIGURA 30 – FORMATO GERAL DOS PROTOCOLOS.



Fonte: [TAF1996]

Um dos objetivos principais do protocolo é detectar e evitar a perda de dados ao longo da transmissão, solicitação a retransmissão deles, caso isto ocorra ([SOU1999]).

4.1 TIPOS BÁSICOS DE PROTOCOLO

Os protocolos podem ser classificados em função dos modos de transmissão quanto aos controles de transmissão. Os modos de operação em que os protocolos podem ser implementados são ([PEN1988][TAF1996]):

a) assíncronos: são do tipo que utilizam *bits* de *start* e *stop* (gerados por *hardware*) para delimitar um caracter, tornando a comunicação pouco eficiente.
b) síncronos: são protocolos em que os dados trabalham sincronamente, ou seja, a estação transmissora e receptora trabalham com o mesmo *clock*, portanto, não há necessidade de usar os *bits* de *start* e *stop*.

4.2 CATEGORIA DE PROTOCOLOS

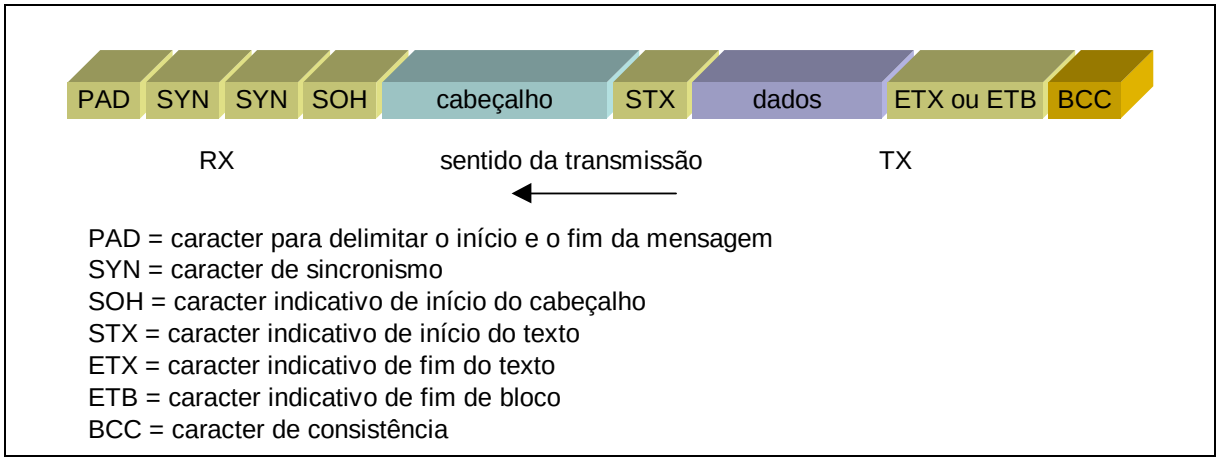
Essa classificação leva em conta o formato das mensagens, que podem ser de duas maneiras: protocolos orientados a caracter(*byte*) e protocolos orientados a *bit*.

4.2.1 PROTOCOLOS ORIENTADOS A CARACTER (*BYTE*)

São protocolos que utilizam caracteres especiais para o controle de operação e do tráfego de suas mensagens. Estes protocolos são os mais antigos e são usados, com freqüência, em linhas de longa distância, por exemplo, o *BSC (Binary Synchronous Communication*) ([PEN1988][TAF1996]).

O formato típico de um bloco de protocolo orientado a caracter (no caso do BSC) pode ser visto na figura 31.

FIGURA 31 – FORMATO TÍPICO DO PROTOCOLO ORIENTADO A *BYTE*.

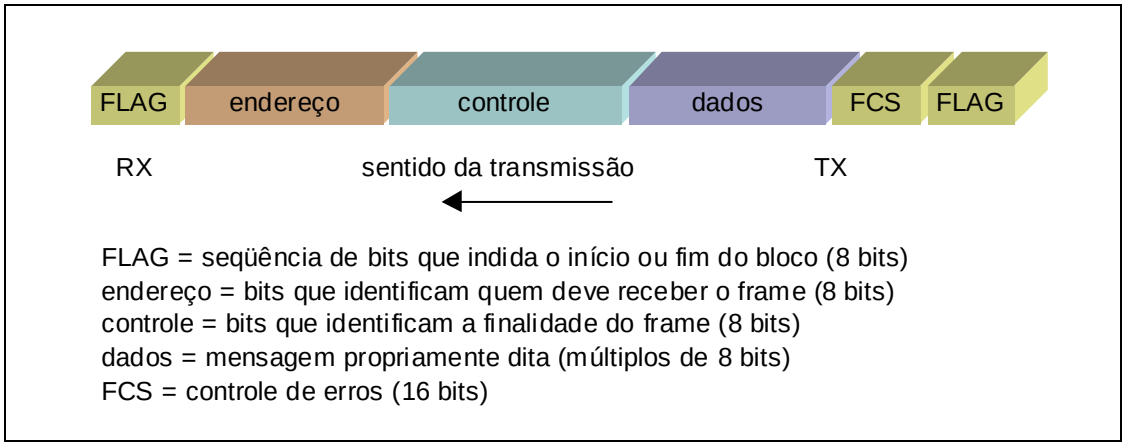


Fonte: [PEN1988]

4.2.2 PROTOCOLOS ORIENTADOS A *BIT*

São protocolos que não utilizam caracteres especiais para delimitar blocos de mensagens. Todo o controle é tratado a nível de *bit* (fig. 32), isto é, campos formados por combinações binárias bem definidas, não existindo caracteres que designem funções especiais, por exemplo, o protocolo X-25 e o protocolo SDLC (*Synchronous Data Link Control*) ([OLI1987][TAF1996]).

FIGURA 32 – FORMATO TÍPICO DO PROTOCOLO ORIENTADO A *BIT*.



Fonte: [OLI1987]

4.3 DETECÇÃO DE ERROS

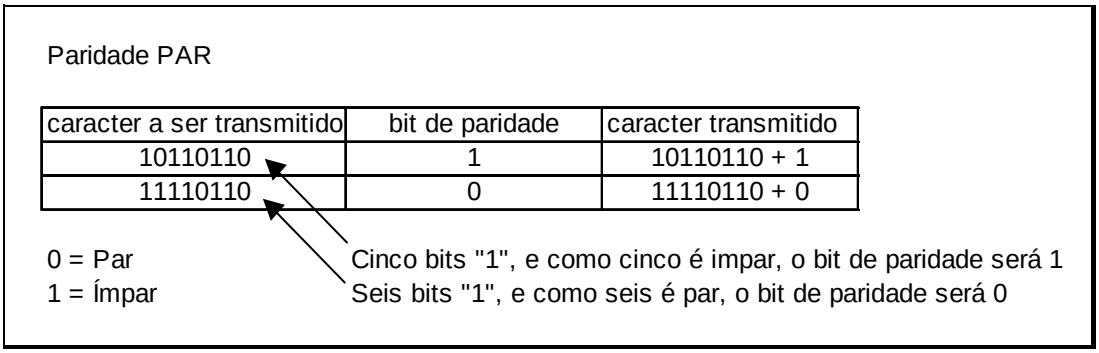
A presença de interferências no canal de transmissão (ruído) pode provocar, eventualmente, uma leitura errônea.

Para detectar a ocorrência de erro foram desenvolvidos vários métodos, baseados na utilização de informações redundantes na comunicação. Uma vez detectado um erro, o procedimento normal é a retransmissão da mensagem. A redundância pode ser colocada a nível de caracter, ou a nível de bloco de caracteres ([PEN1988]).

4.3.1 DETECÇÃO DE ERROS A NÍVEL DE CARACTER

Um método simples e de baixo custo é o código de paridade. Pode-se ter paridade par ou ímpar, obtida pela adição de um *bit* por caracter, fazendo com que o caracter codificado tenha um número par ou ímpar, respectivamente, de *bits* “1” (fig. 33). Esse tipo de controle é chamado VRC (Vertical Redundancy Check) ([PEN1988]).

FIGURA 33 – DEMONSTRAÇÃO DO MÉTODO DE CÁLCULO DA PARIDADE.



Fonte: [TAF1996]

A estação receptora, devidamente configurada de acordo com a estação transmissora, deverá gerar a mesma seqüência de *bits*, somando a quantidade de *bits* "1", verificará se a soma permaneceu par ou impar, detectando assim, a presença ou não de erro.

Existe, ainda, apesar de remota, uma possibilidade de erro que deve ser considerada: um duplo erro sobre o caracter transmitido. Duas trocas de sinal, certamente, mudarão o conteúdo da mensagem, porém, a paridade permanecerá a mesma e, desta forma, a estação receptora é "enganada" pela deformação do sinal ([TAF1996]).

4.3.2 DETECÇÃO DE ERROS A NÍVEL DE BLOCO

Essa forma de detecção é feita anexando um byte redundante no final de um bloco de bytes de informação. Normalmente realiza-se por dois processos: O LRC (*Longitudinal Redundancy Check*) e o CRC (*Cyclical Redundancy Check*).

O LRC consiste na aplicação do processo de paridade em todos os bits que ocupam a mesma posição nos caracteres do bloco, obtendo assim os bits do caracter de controle de erro. A figura 34 mostra um bloco de bits, onde foi aplicado o VRC e o LRC. É comum utilizarem-se os dois métodos simultaneamente ([OLI1987][PEN1988][TAF1996]).

FIGURA 34 – DETECÇÃO DE ERROS A NÍVEL DE BLOCO.

bit	mensagem a ser transmitida					LRC
0	0	1	0	0	1	1
1	0	0	0	0	1	0
2	1	1	0	1	1	1
3	0	0	1	0	1	1
4	1	0	1	1	0	0
5	0	0	0	0	0	1
6	1	1	1	1	1	0
7	0	0	0	0	0	1
VRC	0	0	0	0	0	

Fonte: [TAF1996]

Além do método de LRC, existe também o CRC, que consiste na aplicação de operações aritméticas em todos os *bits* do bloco, para gerar *bytes* de controle de erro, que são transmitidos ao final do bloco ([OLI1987]).

O CRC é o método de controle de erros mais eficiente. Utiliza todos os bits do bloco a ser transmitido para calcular com um algoritmo o resultado que é colocado no final do bloco. O receptor, ao receber o bloco, recalcula o resultado do algoritmo que deve ser igual ao transmitido, caso não seja, solicita a retransmissão do bloco ([SOU1999]).

Devido à complexidade dos algoritmos utilizados, esse método garante praticamente 100% a detecção de erros que ocorram na transmissão([SOU1999]).

4.4 CORREÇÃO DE ERROS

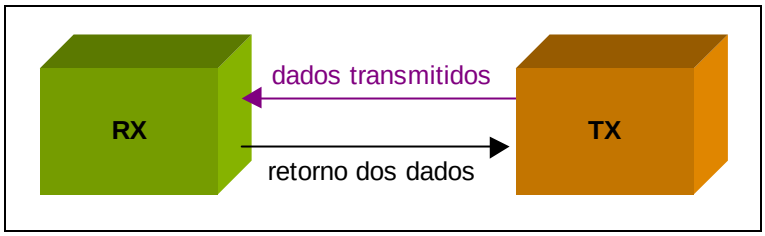
Qualquer processo de detecção de erros traz como conseqüência direta a necessidade de correção. Os métodos de correção, de maneira bem elementar, podem ser classificados em três grupos ([OLI1987]):

a) manual (*echoplexing*);
b) por solicitação;
c) automático.

4.4.1 CORREÇÃO MANUAL

Os dados transmitidos são devolvidos ao transmissor onde deverão ser monitorados e comparados individualmente (fig. 35). Apesar do baixo custo, apresentam a possibilidade de introdução de erros no retorno e a necessidade de uma análise visual interativa. ([OLI1987]).

FIGURA 35 – MÉTODO DE CORREÇÃO MANUAL.



Fonte: [OLI1987]

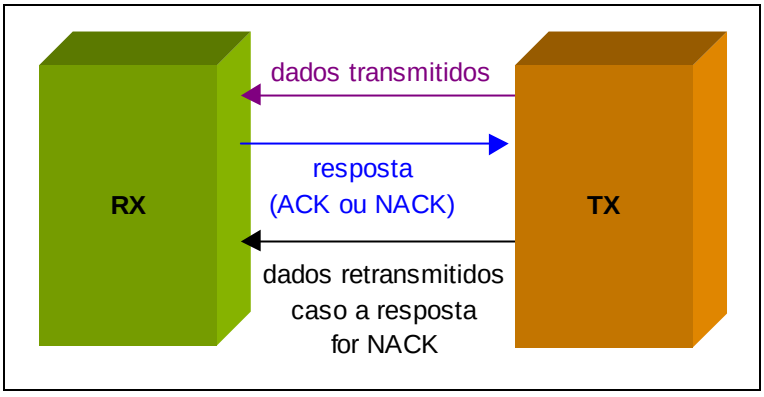
4.4.2 CORREÇÃO POR SOLICITAÇÃO

No método por solicitação, cada bloco de informação é acrescido de um caracter de controle de erro, gerado em função de um código escolhido, como por exemplo o LRC e o VRC. Na recepção, pela análise do caracter de controle de erro, o receptor informa ao transmissor se houve ou não a detecção de erros e, em função disso, solicita ou não a retransmissão ([OLI1987]).

Como exemplo, os caracteres ACK e NACK (fig. 36):

- ACK (*Acknowledgement*) = bloco bem recebido. Favor enviar o próximo bloco.
- NACK (*Negative Acknowledgement*) = bloco recebido com erros. Favor repetir.

FIGURA 36 – MÉTODO DE CORREÇÃO POR SOLICITAÇÃO.

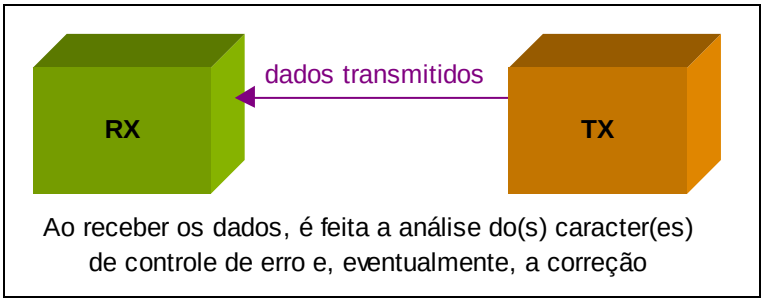


Fonte: [OLI1987]

4.4.3 CORREÇÃO AUTOMÁTICA

Método em que o código escolhido para geração e análise do caracter de controle de erro, em função do dado de informação, elimina a necessidade de retorno para confirmação. O único inconveniente desse código é que grande parte do espaço útil para informações é tomado para fazer o controle necessário e análise (fig. 37) ([OLI1987]).

FIGURA 37 – MÉTODO DE CORREÇÃO AUTOMÁTICA.



Fonte: [OLI1987]

4.4.4 COMPARAÇÃO ENTRE OS MÉTODOS DE DETECÇÃO

A seguir, (tab. 3), é apresentado uma comparação entre os métodos de detecção de erros, apontando vantagens e desvantagens.

TABELA 3 – COMPARAÇÃO ENTRE MÉTODOS DE DETECÇÃO.

	Método *Echoplexing*	**Método por Solicitação**	**Método Automático**
Vantagens	- não necessita controle; - fácil implementação;	- pouco espaço tomado com controle;	- corrige o erro na própria recepção não necessita caracteres para confirmação;
Desvantagens	- possibilidade de erros no retorno; - necessita análise interativa;	- necessário retorno para confirmação (tempo morto);	- muito espaço tomado com controle;

Fonte: [OLI1987]

5 DESENVOLVIMENTO DO TRABALHO

O protótipo apresenta o uso de uma agenda eletrônica interligada a um circuito de transmissão e recepção de dados sem fio através de comunicação serial via RS-232C.

5.1 COMPONENTES DO PROTÓTIPO

Os componentes e o recurso computacional utilizados na montagem do protótipo, estão relacionados a seguir:

a) agenda eletrônica;
b) módulos de RF (rádio-freqüência);
c) circuito auxiliar;
d) *software* para comunicação (Monitor Industrial).

5.1.1 AGENDA ELETRÔNICA

A agenda eletrônica utilizada no protótipo é a Casio SF-5790SY de 256Kb de memória (fig. 38). Este modelo possui um conector que permite interligar a agenda com o PC através da interface RS-232C com um cabo especial.

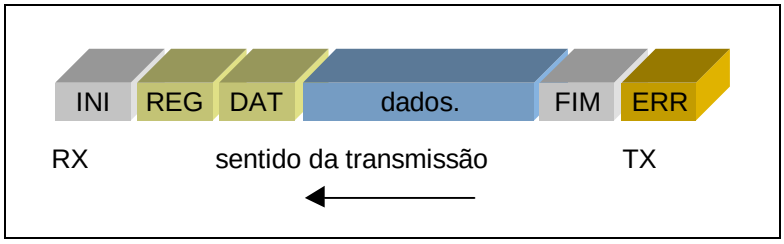
FIGURA 38 – AGENDA ELETRÔNICA CASIO SF-5790SY

A agenda vem acompanhada de um *software* específico para a comunicação com o PC. A finalidade deste programa é de fazer cópias de segurança (*backup*) das informações contidas na agenda. Como é proprietário (seu código não é aberto), não há como extrair as informações referentes a especificação do protocolo. O fabricante ainda declara oficialmente, em [CAS2000], não disponibilizar para terceiros a documentação do protocolo.

Para contornar este problema, foi necessário monitorar a comunicação do programa com a agenda, através de um utilitário denominado PortMon [SYS2000]. Com essa ferramenta é possível monitorar toda a informação que trafega pela porta serial.

Analisando a teoria de protocolos (capítulo 4) com as informações coletadas, foi possível elaborar alguma documentação do pacote (formato do protocolo) (fig. 39).

FIGURA 39 – FORMATO DO PROTOCOLO.



Os caracteres (*bytes*) de controle receberem um nome e suas funções podem ser observadas na tabela abaixo (tab. 4).

TABELA 4 – FUNÇÕES DOS CARACTERES DE CONTROLE.

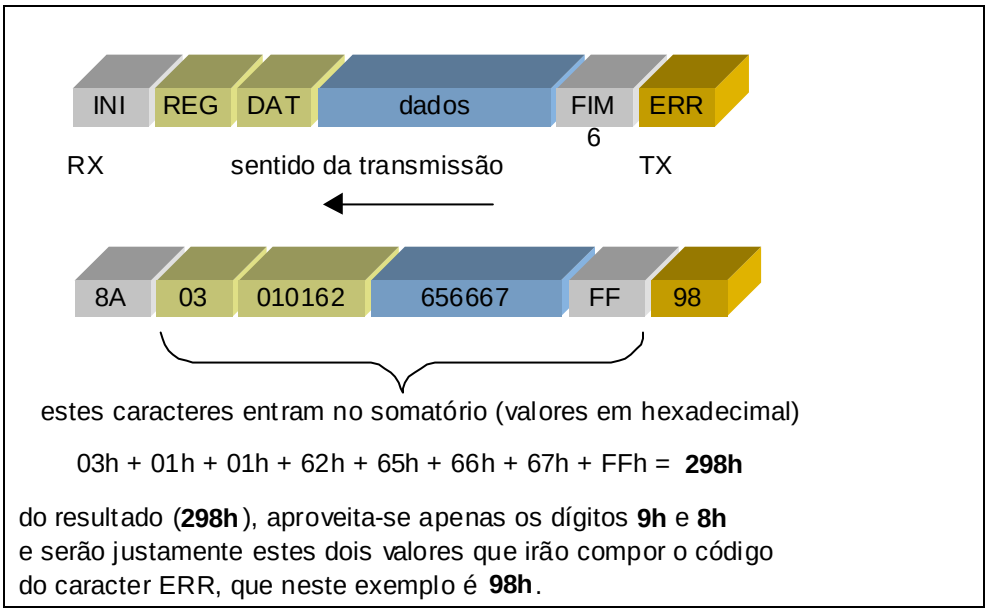
Caracter(es)	**Tamanho**	**Valor** (hexadecimal)	**Função**
INI	1	8A	Indica o início do pacote
REG	1	Variável	Indica o tipo de informação (*memo, phone, schedule,* etc.)
DAT	3	Variável	Data de criação/modificação da informação
FIM	1	FF	Indica o fim do pacote
ERR	1	Variável	Controle de erro

Internamente, a agenda é subdividida em diretórios de informação, como o diretório de telefones (*phone*), diretório de anotações (*memo*), diretório de compromissos (*schedule*), e outros. O caracter **REG** existe para identificar o tipo de informação, ou seja, para indicar de qual diretório o dado pertence. O valor utilizado na aplicação prática do protótipo é 03h, que refere-se ao diretório de anotações (*memo*).

O caracter **DAT** identifica a data de criação ou modificação do dado. É formado por três *bytes*, cujo valor corresponde ao mês, dia e ano respectivamente.

O caracter **ERR** é utilizado no controle de erro. O valor deste é calculado da seguinte forma: Soma-se o valor de todos os *bytes* (exceto do caracter **INI**). Do resultado obtido, extrai-se o valor dos dois dígitos menos significativos (representados em hexadecimal), que respectivamente irão compor o valor do caracter **ERR** (fig. 40)

FIGURA 40 – CÁLCULO DO CARACTER DE CONTROLE DE ERRO.



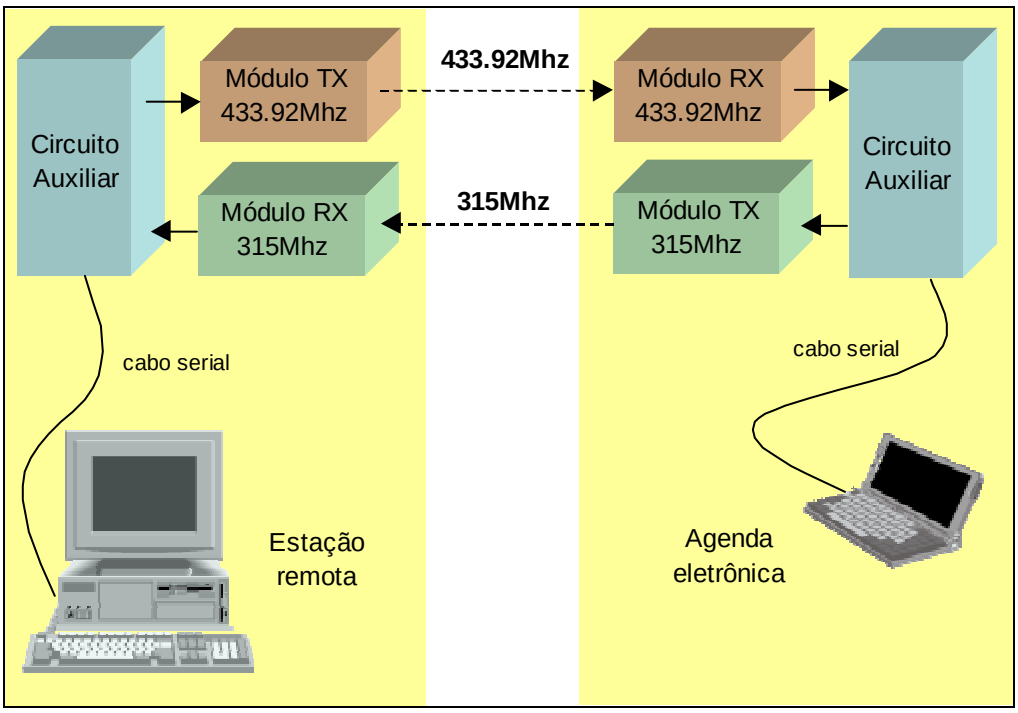
Na comunicação, se a agenda receber algum pacote cujo caracter **ERR** não coincida com o valor calculado, a mesma aborta a comunicação e exibe uma mensagem de erro. Caso contrário, envia um caracter de confirmação de recebimento (**ACK**), que é representado pelo valor BDh (hexadecimal).

5.1.2 MÓDULOS DE TRANSMISSÃO E RECEPÇÃO

Os módulos de RF utilizados no protótipo, são dois transmissores RT4, um operando a 433.92Mhz e outro a 315Mhz, e dois módulos receptores RR3 um operando a 433.92Mhz e outro a 315Mhz, ambos da Telecontrolli [TEL2000].

O sistema é formado por quatro módulos, porque é necessário transmitir e também receber os dados. Isso permite a utilização de dois canais distintos de comunicação, possibilitando assim, uma transmissão *full-duplex* (fig. 41). É importante observar que cada transmissor opera numa freqüência exclusiva, evitando assim a interferência causada pelo módulo que se encontra no sentido oposto, o mesmo vale para os receptores.

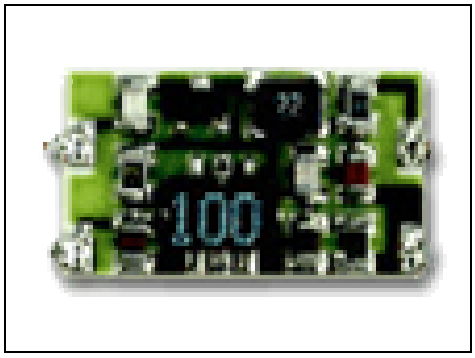
FIGURA 41 – ESQUEMA LÓGICO DA COMUNICAÇÃO.



5.1.2.1 MÓDULOS DE TRANSMISSÃO

Os modelos utilizados no protótipo são: RT4-433.92 e RT4-315 (fig. 42). A dimensão dos módulos é de 17,78mm de comprimento por 10,16mm de largura.

FIGURA 42 – MÓDULO TRANSMISSOR RT4



Fonte: [TEL2000]

Na figura 43, são apresentadas as características elétricas dos módulos de transmissão, bem como sua pinagem e dimensões mecânicas. Maiores informações e detalhes podem ser encontradas na página do fabricante [TEL2000].

FIGURA 43 – CARACTERÍSTICAS DO MÓDULO TRANSMISSOR.

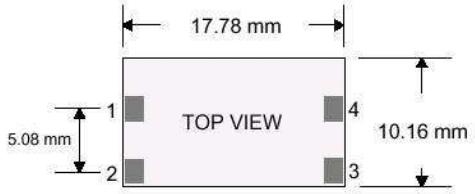
	CHARACTERISTICS	MIN	TYP	MAX	UNIT
V_{CC}	Supply Voltage	2		14	VDC
I_S	Supply Current (Vcc=5V IN=1KHz Square Wave)		4		mA
F_W	Working Frequency	303.8		433.92	MHz
P_O	RF Output Power into 50Ω (Vi=5V, Vcc=12V)		7	10	dBm
	Harmonic Spurious Emission		-30		dBc
V_{IH}	Input High Voltage	2		V_{CC}	V
	Max Data Rate			4	KHz
T_{OP}	Operating Temperature Range	-25		+80	°C

Tipically, equipment utilizing this device requires emissions testing and government approval, wich is the responsibility of the equipment manufacturer.

Pin Description

1	V_{CC}	Supply Voltage
2	GND	Ground
3	IN	Modulation Input
4	EA	External Antenna

Mechanical Dimensions

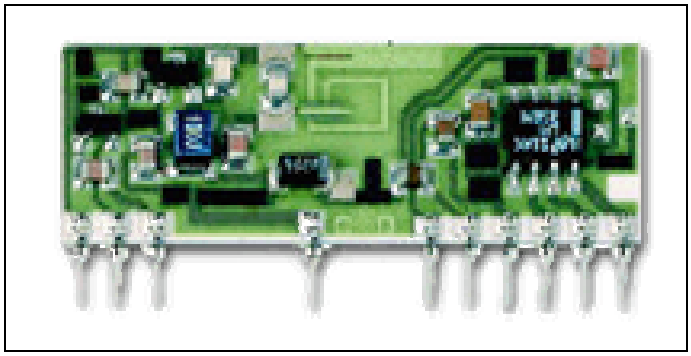


Fonte: [TEL2000]

5.1.2.2 MÓDULOS DE RECEPÇÃO

Os modelos dos módulos de recepção (fig. 44), são RR3-433.92Mhz e RR3-315Mhz. Suas dimensões são 38,1mm de comprimento por 12,7mm de largura.

FIGURA 44 – MÓDULO RECEPTOR RR3



Fonte: [TEL2000]

Abaixo (fig. 45), podemos observar as características elétricas do módulo receptor.

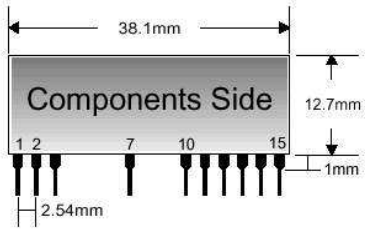
FIGURA 45 – CARACTERÍSTICAS DO MÓDULO RECEPTOR.

	CHARACTERISTICS	MIN	TYP	MAX	UNIT
V_{RF}	RF Supply Voltage	4.5	5	5.5	VDC
V_{AF}	AF Supply Voltage	4.5	5	5.5	VDC
I_S	Supply Current		2.5	3	mA
F_W	Working Frequency	200		450	MHz
	Tuning Tolerance		±0.2	±0.5	MHz
B_W	-3dB Bandwidth		±2	±3	MHz
	Max Data Rate			2	KHz
	RF Sensitivity (100% AM)	-100	-105		dBm
	Level of Emitted Spectrum		-65	-60	dBm
V_{ol}	Low-Level Output Voltage			0.6	V
V_{oh}	High-Level Output Voltage	3.6			V
T_{OP}	Operating Temperature Range	-25		+80	°C

Pin Description

1	RF $+V_{CC}$	9	NC
2	RF GND	10	AF $+V_{CC}$
3	IN	11	AF GND
4	NC	12	AF $+V_{CC}$
5	NC	13	Test Point
6	NC	14	OUT
7	RF GND	15	AF $+V_{CC}$
8	NC		

Mechanical Dimensions



Fonte: [TEL2000]

5.1.3 CIRCUITO AUXILIAR

Devido a características da interface serial RS-232C, não é possível conectar os módulos diretamente na porta serial do PC ou da agenda. Para isso foi necessário elaborar um circuito capaz de compatibilizar a comunicação entre a interface RS-232C e os módulos de RF. Esse circuito foi denominado **circuito auxiliar**.

Para cada módulo de RF existe um circuito auxiliar que é composto basicamente de dois componentes: o circuito integrado MAX232 da Maxim [MAX2000] e o microcontrolador 89C2051 da Atmel [ATM2000].

O MAX232 é responsável pela conversão de tensão do sinal entre a porta serial e o microcontrolador 89C2051 e os módulos de RF.

O sinal elétrico da porta serial opera em +12V e –12V para representar os *bits*, ou seja, +12V para representar o *bit* “1” e –12V para representar o *bit* “0”. Já o circuito auxiliar utiliza outra escala para representar os dígitos binários: +5V para o *bit* “1” e 0V para o *bit* “0”. Para resolver este problema, foi necessário a utilização do MAX232 cuja função é converter o sinal de +12V/-12V para +5V/0V e vice-versa.

No microcontrolador 89C2051 foi gravado um programa cuja finalidade é modular o sinal recebido pelo MAX232 para que o mesmo possa ser transmitido corretamente pelo módulo de RF. Essa modulação é conhecida como PWM (Modulação por Largura de Pulso).

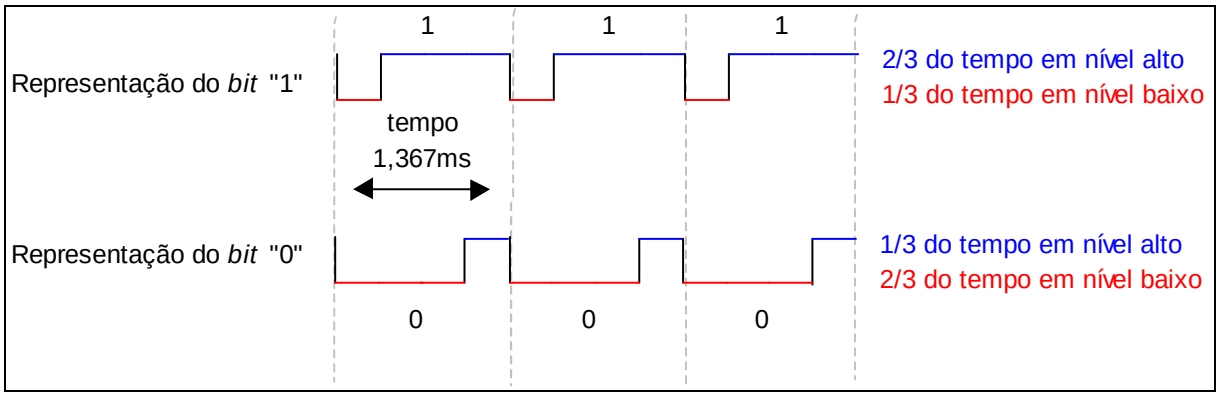
De acordo com [MAR2000], a modulação PWM faz-se necessária porque o módulo interrompe a transmissão se existirem muitos *bits* “1” seguidos. Tal situação ocorre devido a uma limitação do próprio módulo, que não consegue deixar a saída ativa em nível “1” por um tempo superior a 5 microsegundos (2Khz).

O programa do microcontrolador foi baseado originalmente no projeto de Ronaldo Martins [MAR2000], que utilizou a modulação PWM para transmitir dados da porta serial via RF (anexo II e III).

A transmissão de um *byte* via RF é realizada através de PWM, sendo executada *bit* a *bit*. Para representar o *bit* “1”, a duração do tempo em nível lógico “0” será 1/3 do tempo total, e para representar o *bit* “0”, a duração do tempo em nível lógico “0”, será 2/3 do tempo

total (fig. 46). O tempo total de transmissão de um *bit* é de aproximadamente 1,367ms ([MAR2000]).

FIGURA 46 – MODULAÇÃO DOS *BITS* EM PWM.

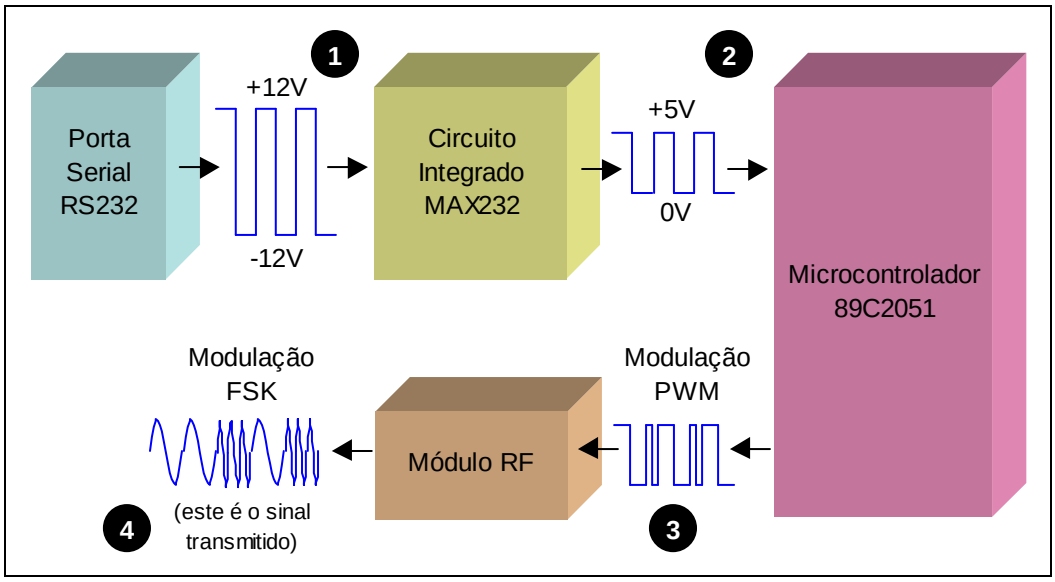


Fonte: [MAR2000]

5.1.3.1 ETAPAS DO PROCESSO DE TRANSMISSÃO/RECEPÇÃO

O processo de transmissão está dividido em quatro etapas, conforme demonstra figura 47. Este esquema também é válido para o processo de recepção, porém o sentido do fluxo do sinal é o inverso.

FIGURA 47 – ETAPAS DO PROCESSO DE TRANSMISSÃO/RECEPÇÃO.



A primeira etapa representa a saída do sinal digital (*bits*) da porta serial até o MAX232. A tensão do sinal que entra no MAX232 oscila entre +12V e –12V.

Na segunda etapa, o sinal é convertido pelo MAX232 e segue em direção ao microcontrolador.

Em seguida, na terceira etapa, o 89C2051 converte o sinal enviado pelo MAX232 em PWM e envia ao módulo transmissor.

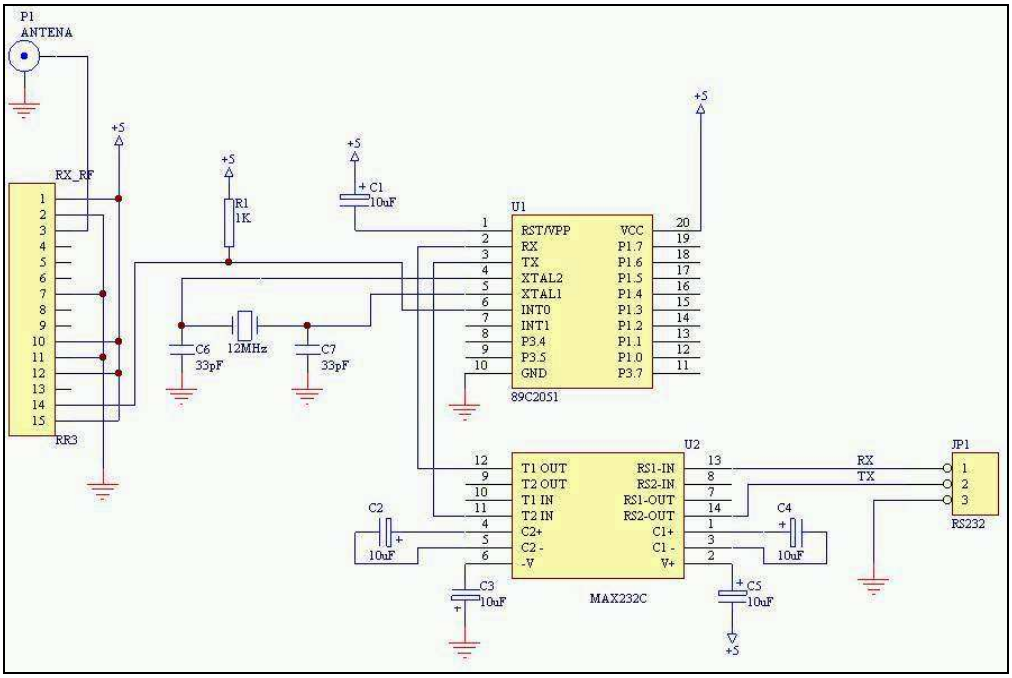
Na quarta e última etapa, o sinal recebido pelo transmissor, é novamente modulado e finalmente transmitido. Segundo o fabricante [TEL2000], a modulação utilizada é a FSK (Modulação por Chaveamento de Freqüência).

5.1.3.2 DIAGRAMA ESQUEMÁTICO DOS CIRCUITOS

Os esquemas apresentados neste trabalho, foram originalmente criados por Ronaldo Martins [MAR2000], e demonstram a montagem do circuito auxiliar juntamente com os módulos. Os esquemas também indicam os valores dos componentes eletrônicos utilizados na construção.

Cada circuito de transmissão/recepção (montado individualmente por módulo) é composto basicamente da junção do circuito auxiliar com o módulo de RF. No esquema abaixo (fig. 48) está representado o circuito de recepção.

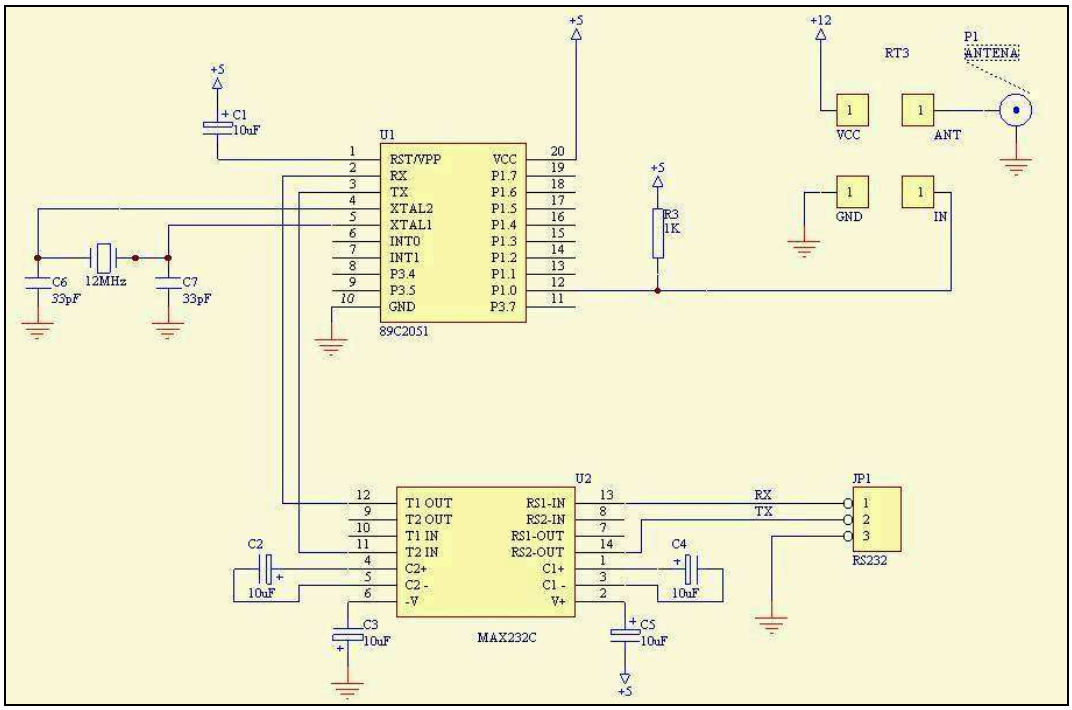
FIGURA 48 – DIAGRAMA ESQUEMÁTICO DO CIRCUITO DE RECEPÇÃO.



Fonte: [MAR2000]

O circuito de transmissão difere do circuito de recepção, por isso foi construído separadamente. A figura 49 demonstra o esquema para montagem do circuito de transmissão.

FIGURA 49 – DIAGRAMA ESQUEMÁTICO DO CIRCUITO DE TRANSMISSÃO.



Fonte: [MAR2000]

5.1.4 *SOFTWARE* PARA COMUNICAÇÃO

Para demonstrar a funcionalidade do sistema, desenvolveu-se um programa, cuja função é tratar a comunicação entre a agenda e o PC. Esse *software* é responsável pelo processo de comunicação, controle do protocolo e interfaceamento com entidades externas (equipamentos).

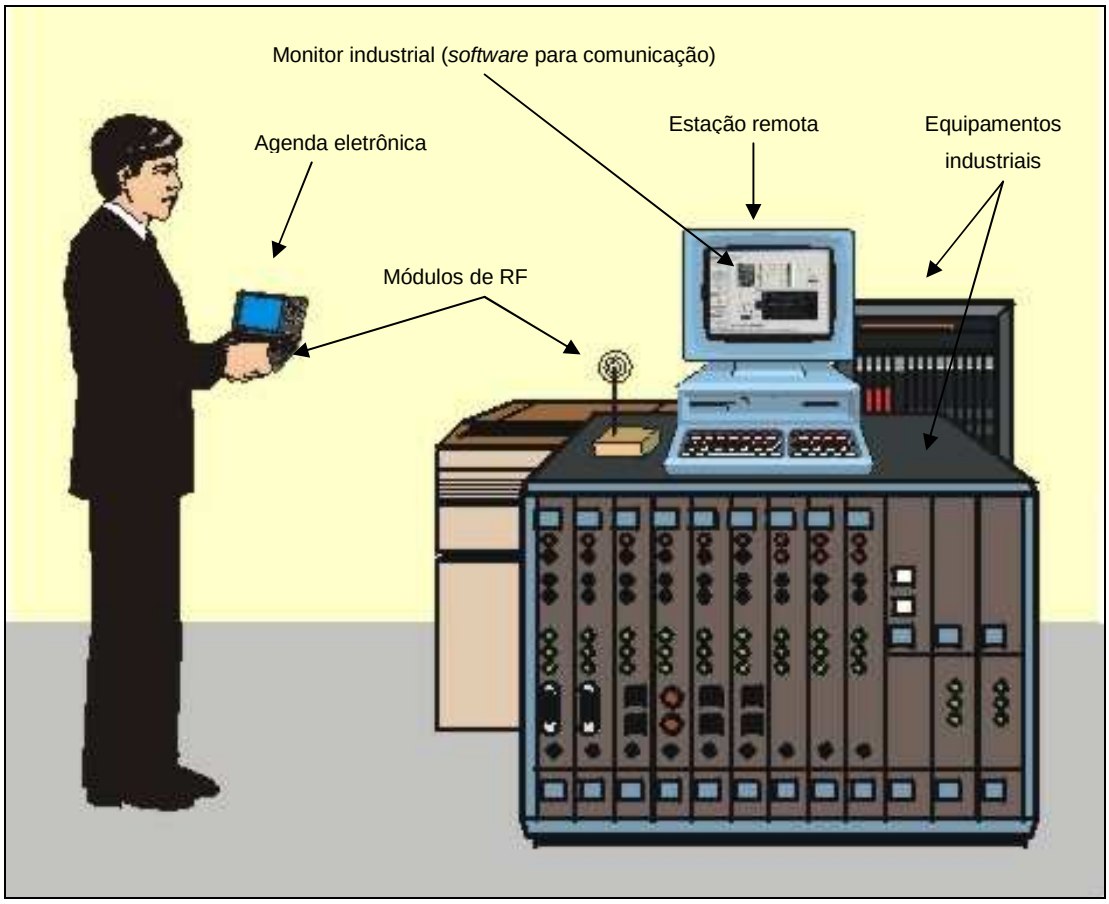
O *software* para comunicação, denominado **Monitor Industrial**, apresenta uma simulação de controle de equipamentos industriais. O papel da agenda eletrônica é de um terminal de controle, que opera e monitora as funções das máquinas à distância.

Sua especificação, detalhes de implementação e documentação podem ser encontradas no item seguinte: 5.2 Especificação.

5.2 ESPECIFICAÇÃO

O objetivo do Monitor Industrial é demonstrar a aplicação prática de um sistema que utiliza a agenda eletrônica como terminal remoto para controle e monitoramento de equipamentos industriais (fig. 50).

FIGURA 50 – APLICAÇÃO PRÁTICA DO MONITOR INDUSTRIAL.



Os equipamentos industriais estão **simulados** no protótipo e supostamente estariam interligados com a estação remota e também seriam controlados pelo Monitor Industrial.

A agenda envia requisições ao Monitor Industrial, que por sua vez interpreta e processa as informações recebidas e, eventualmente, devolve a informação requerida.

As requisições enviadas pela agenda são comandos para ligar/desligar maquinário, monitorar e regular o painel de instrumentação como temperatura, pressão, tensão, corrente, etc.

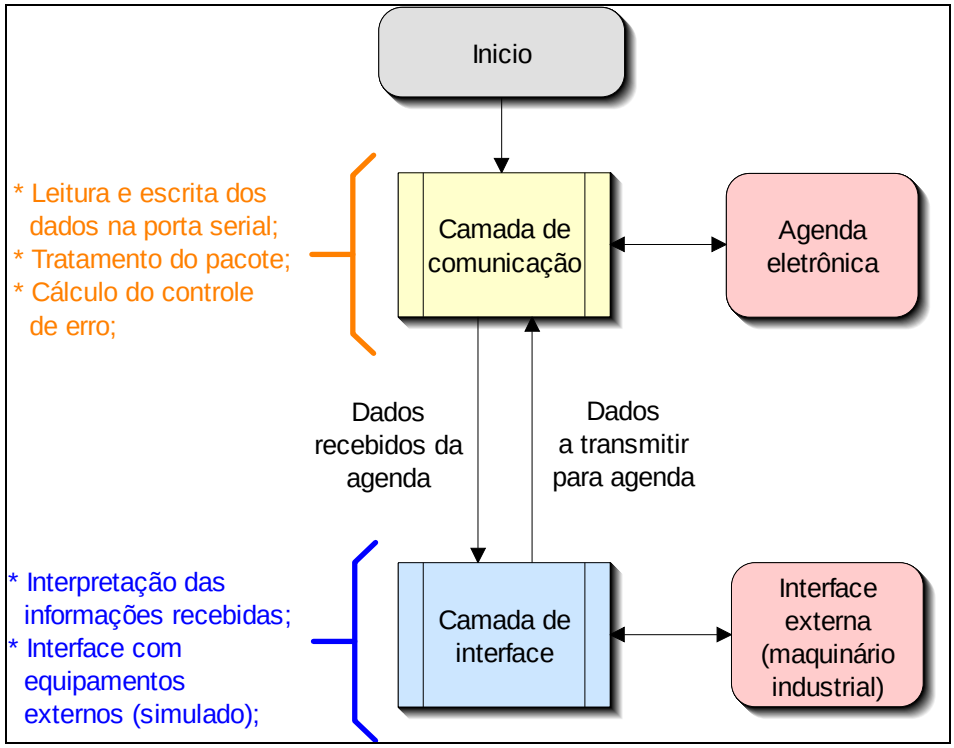
5.2.1 APRESENTAÇÃO DA ESPECIFICAÇÃO

O Monitor Industrial está dividido em duas principais camadas (fig. 51), no qual, cada camada aborda um agregado de funções específicas. São elas:

a) camada de comunicação;

b) camada de interface.

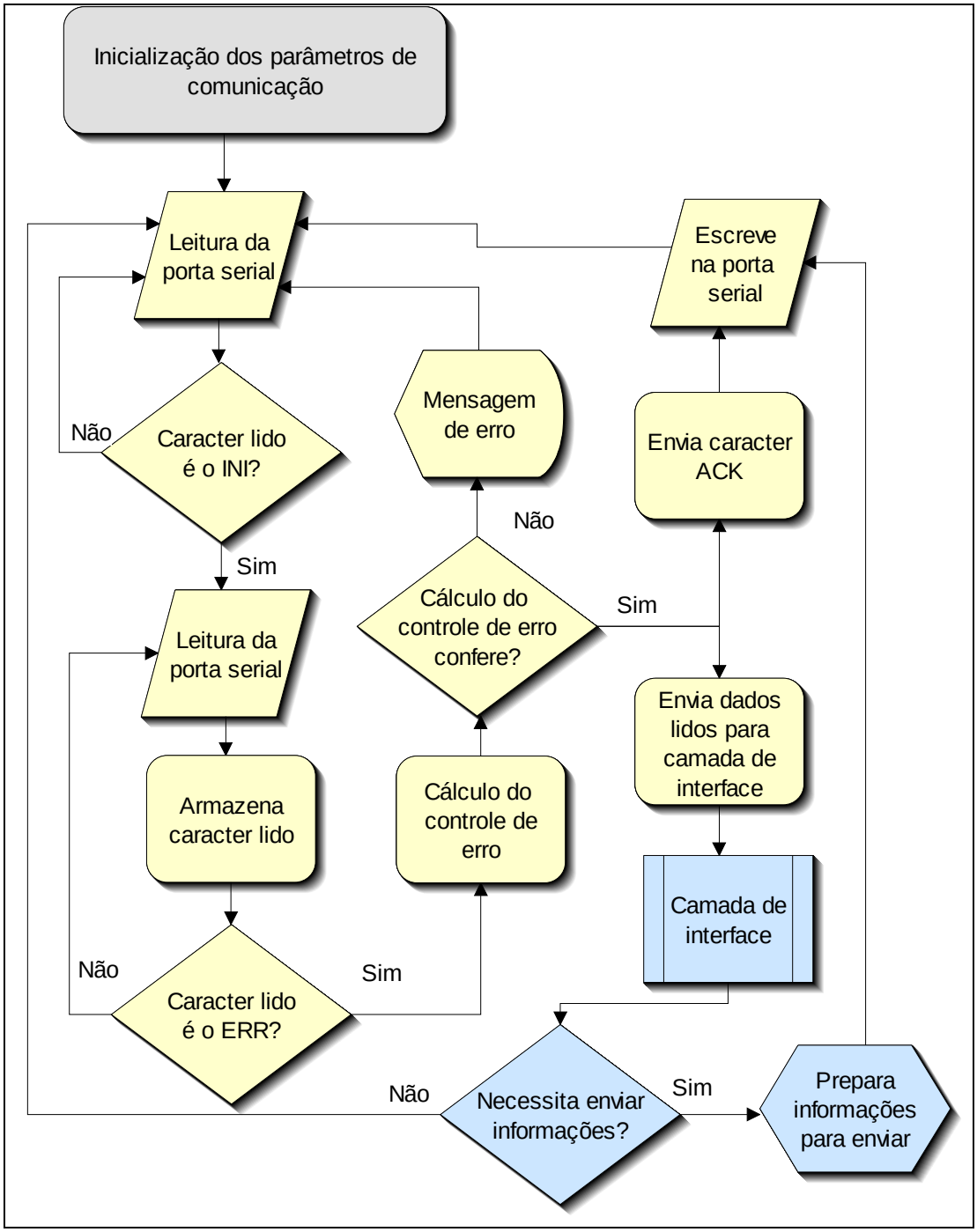
FIGURA 51 – CAMADAS DO MONITOR INDUSTRIAL.



5.2.2 CAMADA DE COMUNICAÇÃO

Nesta camada estão implementadas as rotinas de comunicação com a porta serial, bem como o tratamento e controle do protocolo da agenda. No fluxograma (fig. 52) pode-se visualizar a funcionalidade da camada.

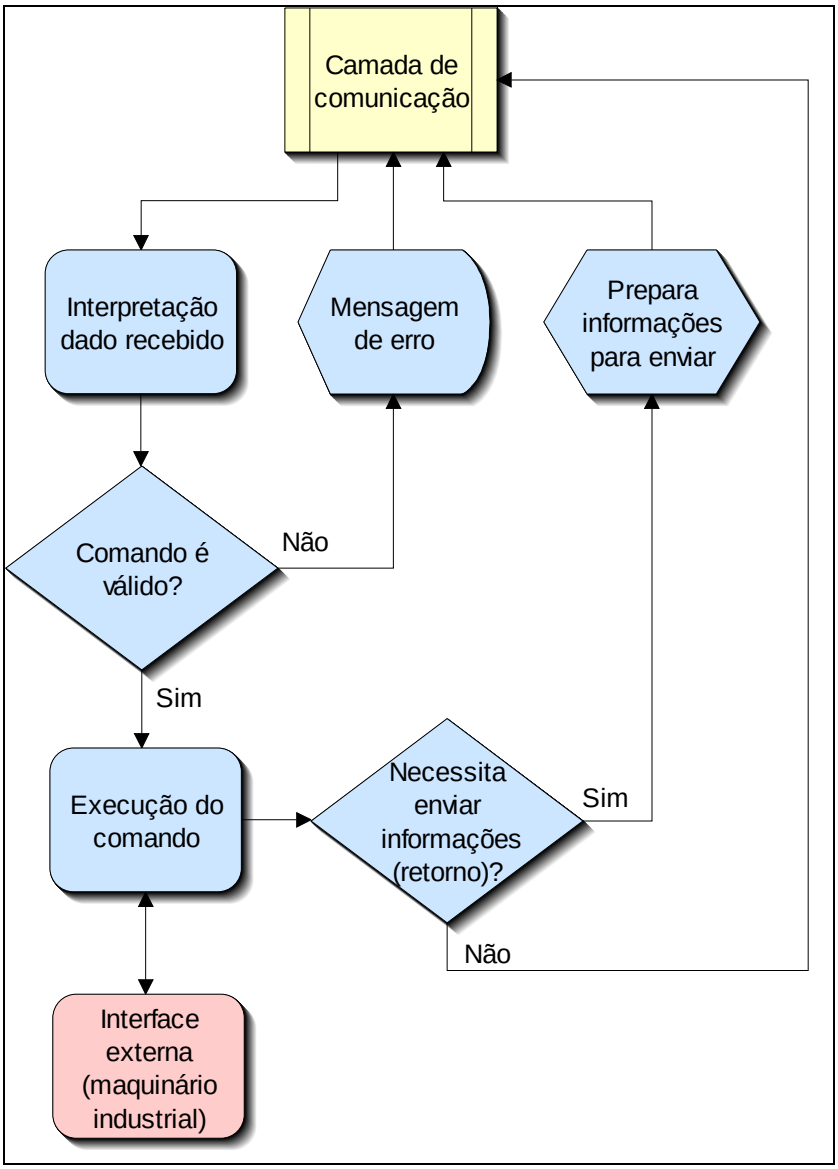
FIGURA 52 – FLUXOGRAMA DA CAMADA DE COMUNICAÇÃO E INTERFACE.



5.2.3 CAMADA DE INTERFACE

A camada de interface possui as seguintes funções: interpretar a informação recebida, processar (executar) e comunicar-se com as entidades externas (fig. 53).

FIGURA 53 – FLUXOGRAMA DA CAMADA DE INTERFACE.



A entrada de comandos pode ser tanto localmente (direto na estação) quando remotamente (pela agenda). Os comandos de execução e suas funções, estão descritas na tabela a seguir (tab. 5):

TABELA 5 – COMANDOS DO MONITOR INDUSTRIAL.

SISTEMA LIG/DES	Liga e desliga o sistema
LIG/DES Z1	Liga e desliga o quadro de energia para a zona 1
LIG/DES Z2	Liga e desliga o quadro de energia para a zona 2
LIG/DES Z3	Liga e desliga o quadro de energia para a zona 3
LIG/DES Z4	Liga e desliga o quadro de energia para a zona 4
ATI/DES PRESSURIZADORES	Ativa e desativa os pressurizadores
ATI/DES AQUECEDORES	Ativa e desativa os aquecedores
INFO ENERGIA	Obtém informações do quadro de energia de cada zona
INFO PRESSAO	Obtém informações da pressurização de cada zona
INFO TEMPERATURA	Obtém informações da temperatura de cada zona
INFO GAS	Obtém informações da taxa de mistura dos gases

5.3 IMPLEMENTAÇÃO

Este subcapítulo aborta considerações sobre técnicas e ferramentas utilizadas no desenvolvimento do protótipo. Também é apresentada a operacionalidade do protótipo.

5.3.1 TÉCNICAS E FERRAMENTAS UTILIZADAS

O *software* de comunicação (Monitor Industrial) foi desenvolvido integralmente na linguagem ObjectPascal através do ambiente de desenvolvimento Delphi 5.0. As rotinas de comunicação utilizam-se de chamadas a biblioteca API (*Application Programming Interface*) do sistema operacional Windows (9x/NT).

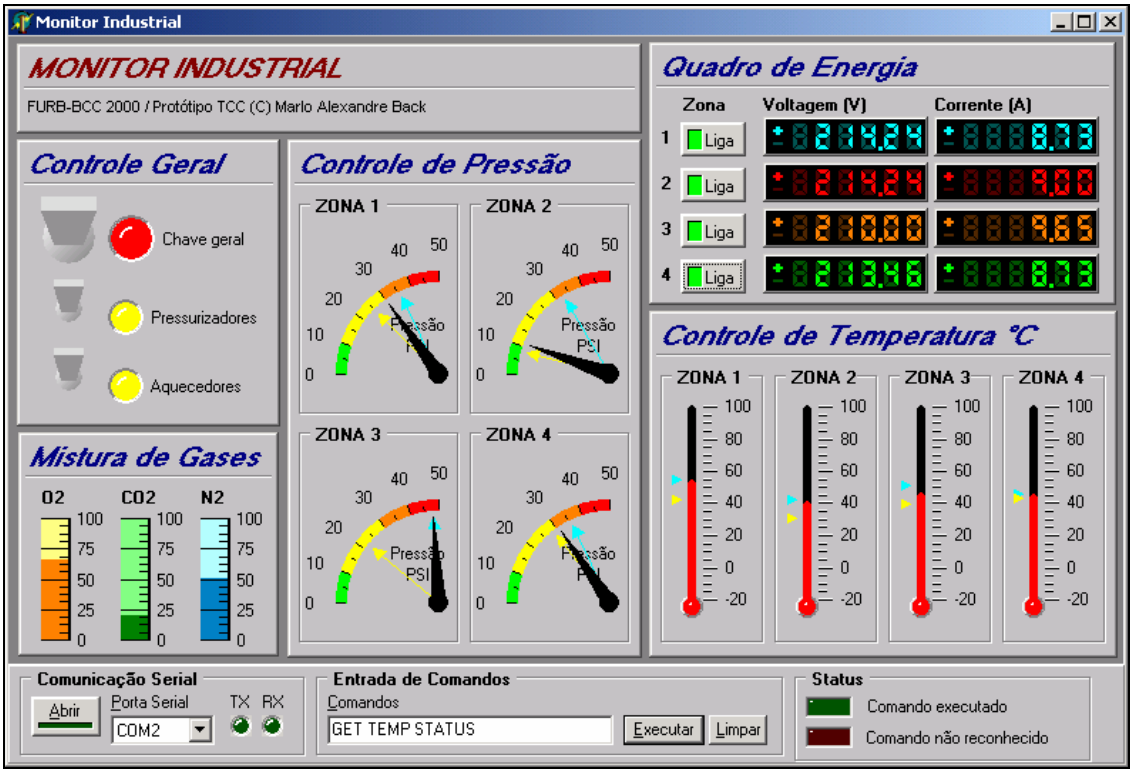
Os componentes visuais de instrumentação (medidores, indicadores e termômetros) utilizados na simulação do protótipo são proprietários da Iocomp Instrumentation Componentes [IOC2000].

5.3.2 OPERACIONALIDADE DA IMPLEMENTAÇÃO

A interface do Monitor Industrial apresentada na figura 54, está dividida em 7 partes:

a) **controle geral**: controla as chaves de ativamento geral do sistema;
b) **quadro de energia**: indicador e controle de energia individual por zona;
c) **controle de pressão**: indicadores individuais de pressão por zona;
d) **controle de temperatura**: indicadores individuais de temperatura por zona;
e) **comunicação serial**: controle da comunicação com a agenda eletrônica;
f) **entrada de comandos**: comandos de controle do sistema;
g) ***status***: indicadores de execução de comandos.

FIGURA 54 – INTERFACE DO MONITOR INDUSTRIAL.



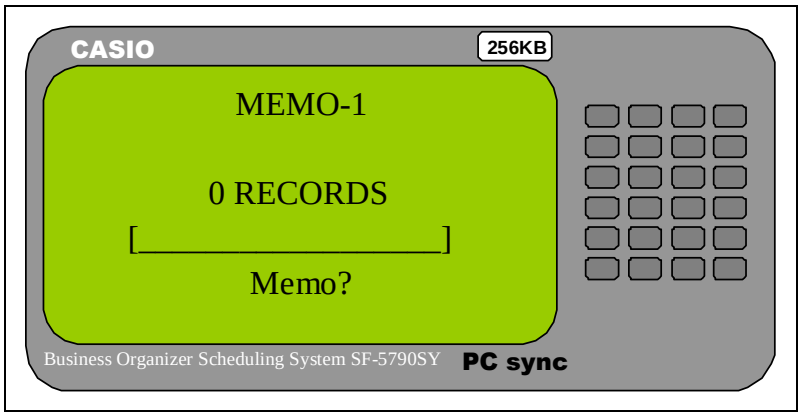
Cada zona indicada no Monitor Industrial, estaria supostamente ligada a um equipamento externo, que enviaria ao *software* informações como, pressão, temperatura,

voltagem, corrente e ainda estaria submetido ao controle do programa. Os valores representados no Monitor Industrial são simulados.

Estando o sistema devidamente montado (módulos de RF, agenda, estação) é necessário configurar a porta serial na qual os módulos estarão ligados, na área de comunicação serial do Monitor Industrial. Em seguida, abre-se a porta para iniciar a comunicação.

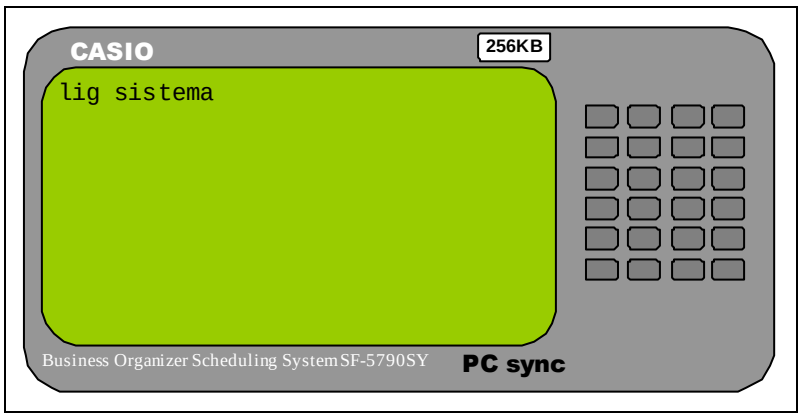
Na agenda eletrônica, seleciona-se a opção *memo* (fig. 55), onde serão digitados e transmitidos os comandos para o Monitor Industrial.

FIGURA 55 – FUNÇÃO *MEMO* DA AGENDA ELETRÔNICA.



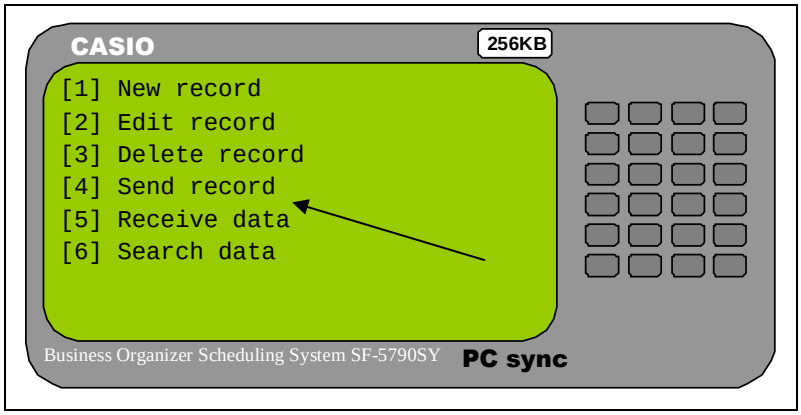
Em seguida digita-se o comando desejado, como por exemplo, o comando “LIG SISTEMA” cuja função é ativar os controles do Monitor Industrial (fig. 56).

FIGURA 56 – DIGITAÇÃO DOS COMANDOS NA AGENDA.



Uma vez concluída esta parte, é necessário enviar o comando para o Monitor Industrial, ou seja, transmitir a informação. Para isso, basta pressionar a tecla *FUNC*, da agenda eletrônica, e selecionar a opção para enviar registro (*send record*) (fig. 57).

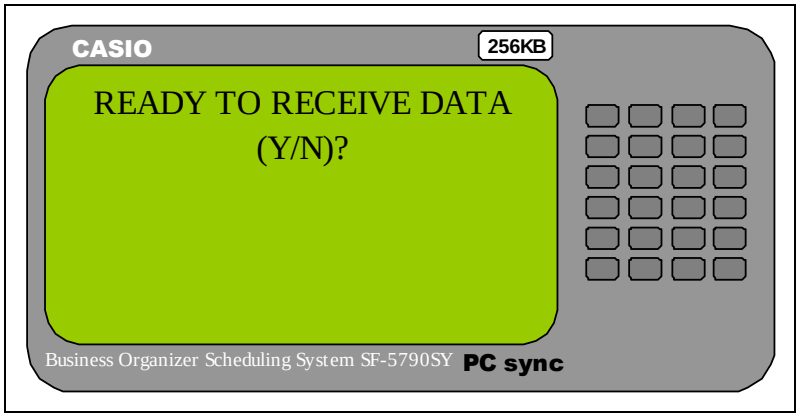
FIGURA 57 – OPÇÃO PARA TRANSMITIR OS DADOS.



Ao concluir esta etapa, o Monitor Industrial já deverá estar processando a informação que lhe foi transmitida.

Existem determinados comandos que requisitam uma informação do Monitor Industrial. Neste caso é necessário enviar as informações, em seguida colocar a agenda em modo de recepção de dados (fig. 58).

FIGURA 58 – MODO DE RECEPÇÃO DE DADOS.



Em seguida, o programa se encarrega de enviar as informações requisitadas para a agenda. É através desta forma que ocorre a interação da agenda eletrônica e a estação remota através da comunicação de dados sem fio.

6 CONCLUSÃO

O desenvolvimento deste trabalho, permitiu demonstrar uma solução simples e prática para um problema que aparenta ser algo muito mais complexo do que a realidade. O uso da tecnologia de comunicação de dados sem fio, juntamente com recursos computacionais, permite o desenvolvimento de sistemas de comunicação eficazes, seguros, e práticos.

A comunicação de dados sem fio é uma forte tendência a ser implementada em muitos dos equipamentos que conhecemos. A mobilidade e praticidade do *wireless*, desperta, em cada vez mais pessoas, o interesse em desenvolver novos projetos baseados nessa tecnologia.

O objetivo do protótipo, estabelecer uma comunicação sem fio entre a agenda e o PC, foi alcançado com sucesso. Esta solução, no decorrer de seu desenvolvimento, abriu um universo de idéias muito maior do que a original. Com a documentação do protocolo nativo da agenda, é possível também, adaptar o protótipo para funcionar como um coletor de dados, um terminal para auxilio de vendas, um terminal para entrada de pedidos, interface com sistemas comerciais e muitos outros.

6.1 CONSIDERAÇÕES FINAIS

Os componentes utilizados na montagem do *hardware* foram adquiridos em São Paulo – SP via sedex, e estão disponíveis na maioria dos estabelecimentos especializados em componentes eletrônicos. A grande vantagem da montagem, é o baixo orçamento, comparado a sistemas prontos (já montados).

A agenda utilizada no protótipo é compatível com todos os sub-modelos de sua série. O *software* para comunicação permite a utilização de outros modelos de agenda, bastando para isso, fazer pequenos ajustes no tratamento do protocolo.

6.1.1 DIFICULDADES ENCONTRADAS

Uma das maiores dificuldades na elaboração do protótipo, foi documentar e testar o protocolo nativo da agenda. Sem nenhuma documentação oficial disponível, o processo de comunicação da agenda com seu programa nativo teve que ser minuciosamente depurado.

Através de muitos testes, conseguiu-se também descobrir o algoritmo de cálculo do caracter de controle de erro do pacote.

Outro grande problema, foi a necessidade de adaptar um microcontrolador programado para modular o sinal em PWM. Sem a utilização deste microcontrolador, seria praticamente impossível transmitir as informações de um módulo para outro. Todos os testes sem a utilização do microcontrolador resultaram em 100% de falha.

6.1.2 LIMITAÇÕES DO SISTEMA

Dentre as limitações que mais se destacam, podemos citar:

a) curto alcance do sistema (máximo 50 metros em condições ideais);
b) baixa taxa de transmissão (o módulo suporta até 2400bps, mas a utilização da modulação PWM reduz ainda mais esta taxa);
c) sensibilidade à ruídos (ambientes com barreiras metálicas produzem alto índice de ruídos);
d) tamanho reduzido do registro da agenda (pode-se enviar/receber um registro que contenha no máximo 255 *bytes* de dados);

6.2 EXTENSÕES

Nesta área estão disponibilizadas algumas idéias e sugestões para trabalhos futuros, focando a utilização da agenda eletrônica e/ou comunicação de dados sem fios.

Uma sugestão interessante, seria adaptar a agenda eletrônica para conectar a um circuito ligado a um modem, que utilizando um microcontrolador programado, poderia discar, enviar e receber e-mails pela Internet.

Outra idéia, como já comentado anteriormente, é modificar o sistema para funcionar como um terminal de consulta, um terminal de entrada de pedidos, coletor de dados e outros.

REFERÊNCIAS BIBLIOGRÁFICAS

[ALD2000] ALDEMARI. **Redes de computadores**: 2000. Endereço Eletrônico: http://www.cnet.com.br/aldemari/rede/redes.htm. Data da consulta: 01/10/2000.

[ATM2000] ATMEL. **Microcontroladores programáveis**: 2000. Endereço eletrônico: http://www.atmel.com/. Data da consulta: 15/09/2000.

[BLA2000] BLACK BOX. **Redes sem fio**: 2000. Endereço Eletrônico: http://www.blackbox.com.br/. Data da consulta: 01/09/2000.

[CAS2000] CASIOWORLD. ***Frequently asked questions (Casio digital diaries)***: 2000. Endereço Eletrônico: http://www.casioworld.com/. Data da consulta: 02/05/2000.

[DAV2000] DAVI, Castro R. S. **Espalhamento espectral**: 2000. Endereço Eletrônico: http://turma-aguia.com/davi/ss/ss_principal.htm. Data da consulta: 01/11/2000.

[FON1998] FONSECA, José Luiz,. **Redes de computadores**: 1998. Endereço Eletrônico: http://www.msb.br/~jluiz. Data da consulta: 01/06/1998.

[IOC2000] IOCOMP. ***Instrumentation Components***: 2000. Endereço Eletrônico: http://www.iocomp.com/. Data da consulta: 08/09/2000.

[JAM1998] JAMUNDÁ, Teobaldo. **Protótipo de um sistema de aquisição de dados utilizando rádio-freqüência**. Blumenau, 1998. Monografia (Bacharelado em Ciências da Computação) Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau.

[JOR1994] JORDAN, Larry; CHURCHILL, Bruce. **Comunicações e redes com o PC**. Trad. de Ernesto Versas. Rio de Janeiro : Axcel Books, 1994.

[MAR2000] MARTINS, Ronaldo. **Gafa *hardware***: 2000. Endereço Eletrônico: http://sites.uol.com.br/tuta. Data da consulta: 15/09/2000.

[MAX2000] MAXIM. ***Maxim Integrated Circuit***: 2000. Endereço Eletrônico: http://www.maxim-ic.com/. Data da consulta: 08/11/2000.

[MIN2000] MINISTÉRIO DAS COMUNICAÇÕES. **Glossário**: 2000. Endereço Eletrônico: http://www.mc.gov.br/. Data da consulta: 11/10/2000.

[OLI1987] OLIVEIRA, Luís Antônio Alves de. **Comunicação de dados e teleprocessamento**. São Paulo : Atlas, 1987.

[PEN1988] PENNA, Rubens M. **Teleprocessamento BSC-3** : conceitos, aplicações e o protocolo. São Paulo : Érica, 1988.

[SMC2000] SMC Networks. ***Glossary RS-232***: 2000. Endereço Eletrônico: http://www.smc.com/smc/glossary/RS232.html. Data da consulta: 09/10/2000.

[SOU1996] SOUSA, Lindeberg Barros de. **Redes: transmissão de dados, voz e imagem**. São Paulo : Érica, 1996.

[SOU1999] SOUSA, Lindeberg Barros de. **Redes de computadores: dados, voz e imagem**. São Paulo : Érica, 1999.

[SPE1997] SENGLER, Tiago; MORO, Mirella Mora. **Meios de transmissão**: 1997. End. Eletr.: http://king.inf.ufrgs.br/aplic/tutoriais/redes972/tiago/meiostr.html. Data da consulta: 05/10/2000.

[SYS2000] SYSINTERNALS. **Utilitários para sistemas *Windows* 95/NT**: 2000. Endereço Eletrônico: http://www.sysinternals.com/. Data da consulta 15/09/2000.

[TAF1996] TAFNER, Malcon Anderson; LOESCH, Claudio; STRINGARI, Sérgio. **Comunicação de dados usando linguagem "C"**. Blumenau : Editora da FURB, 1996.

[TEL2000] TELECONTROLLI; **Módulos transmissores e receptores via RF**. End. Eletrônico: http://www.telecontrolli.com. Data da consulta: 20/08/2000.

[VAR1999] VARGAS, José E. **Princípios de comunicação**: 1999. Endereço Eletrônico: http://polo01.feg.unesp.br/~jvargas/pc/index.html. Data de consulta: 05/10/2000.

ANEXO I

Código fonte do Monitor Industrial

(principal - parte 1/5)

```
unit UMain;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
Dialogs,
  CPort, ExtCtrls, iProfessionalLibrary_TLB, isDigitalLibrary_TLB,
  StdCtrls, OleCtrls, isAnalogLibrary_TLB, AxCtrls;

const
  clOn = clSilver;
  clOff = clGray;

type
  TfmMain = class(TForm)
    ComPort: TComPort;
    Panel1: TPanel;
    Panel2: TPanel;
    ipnCp: TiPanelX;
    ipnQe: TiPanelX;
    iBtQe2: TiSwitchLedX;
    iLdVv1: TiSevenSegmentAnalogX;
    iLdVv2: TiSevenSegmentAnalogX;
    iLdVv3: TiSevenSegmentAnalogX;
    iLdVv4: TiSevenSegmentAnalogX;
    iLdVa1: TiSevenSegmentAnalogX;
    iLdVa2: TiSevenSegmentAnalogX;
    iLdVa3: TiSevenSegmentAnalogX;
    iLdVa4: TiSevenSegmentAnalogX;
    iBtQe3: TiSwitchLedX;
    iBtQe4: TiSwitchLedX;
    ibtQe1: TiSwitchLedX;
    lbQe2: TLabel;
    lbQe3: TLabel;
    lbQe4: TLabel;
    lbQe5: TLabel;
    lbQe6: TLabel;
    lbQe7: TLabel;
    ipnCt: TiPanelX;
    gbP1: TGroupBox;
    iAg1: TiAngularGaugeX;
    gbP2: TGroupBox;
    iAg2: TiAngularGaugeX;
    gbP3: TGroupBox;
    iAg3: TiAngularGaugeX;
    gbP4: TGroupBox;
    iAg4: TiAngularGaugeX;
    gbT1: TGroupBox;
    iT1: TiThermometerX;
    gbT2: TGroupBox;
    iT2: TiThermometerX;
    gbT3: TGroupBox;
    iT3: TiThermometerX;
    gbT4: TGroupBox;
    iT4: TiThermometerX;
    lbQe1: TLabel;
    GroupBox9: TGroupBox;
    ibtOpenPort: TiSwitchLedX;
    cbComPort: TComboBox;
    Label3: TLabel;
    iLdTx: TiLedRoundX;
    iLdRx: TiLedRoundX;
    Label2: TLabel;
    Label1: TLabel;
    GroupBox10: TGroupBox;
    edComandos: TEdit;
    Label11: TLabel;
    iPanelX4: TiPanelX;
    iPanelX5: TiPanelX;
    Label12: TLabel;
    iBtChaveGeral: TiSwitchLeverX;
    ibtChavePressao: TiSwitchLeverX;
    ibtChaveAquecedores: TiSwitchLeverX;
    Label13: TLabel;
    Label14: TLabel;
    Label16: TLabel;
    btExecutar: TButton;
    btLimpar: TButton;
    gbStatus: TGroupBox;
    ildOk: TiLedRectangleX;
    ildNotOk: TiLedRectangleX;
    Label4: TLabel;
    Label5: TLabel;
    TimerRandomT: TTimer;
    TimerDelay: TTimer;
    TimerRandomP: TTimer;
    TimerRandomE: TTimer;
    ildPressao: TiLedRoundX;
    ildAquecedores: TiLedRoundX;
    ildChaveGeral: TiLedRoundX;
    ipnG: TiPanelX;
    Label6: TLabel;
    Label7: TLabel;
    Label8: TLabel;
    ilg3: TiLinearGaugeX;
    ilg2: TiLinearGaugeX;
    ilg1: TiLinearGaugeX;
    TimerRandomG: TTimer;
    procedure FormCreate(Sender: TObject);
    procedure iBtChaveGeralChange(Sender: TObject);
    procedure ibtChavePressaoChange(Sender: TObject);
    procedure ibtChaveAquecedoresChange(Sender: TObject);
    procedure ibtQe1Change(Sender: TObject);
    procedure iBtQe2Change(Sender: TObject);
    procedure iBtQe3Change(Sender: TObject);
    procedure iBtQe4Change(Sender: TObject);
    procedure cbComPortChange(Sender: TObject);
    procedure ibtOpenPortChange(Sender: TObject);
    procedure btExecutarClick(Sender: TObject);
    procedure btLimparClick(Sender: TObject);
    procedure TimerRandomPTimer(Sender: TObject);
    procedure TimerRandomTTimer(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure TimerDelayTimer(Sender: TObject);
    procedure TimerRandomETimer(Sender: TObject);
    procedure TimerRandomGTimer(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
    procedure SetControlsOn;
    procedure SetControlsOff;
    procedure SetPressureControlOn;
    procedure SetPressureControlOff;
    procedure SetTemperatureControlOn;
    procedure SetTemperatureControlOff;
    procedure SetQeOff;
    procedure SetQe1On;
    procedure SetQe1Off;
    procedure SetQe2On;
    procedure SetQe2Off;
    procedure SetQe3On;
    procedure SetQe3Off;
    procedure SetQe4On;
    procedure SetQe4Off;
    procedure DisplayTX;
    procedure DisplayRX;
    procedure GetPowerStatus;
    procedure GetPressureStatus;
    procedure GetTempStatus;
    procedure GetGasStatus;
    procedure Wait(Time: integer);
  end;

var
  fmMain: TfmMain;
  Data: string;

implementation

uses
  UExec, USimul, UComm;

{$R *.DFM}

procedure TfmMain.SetControlsOn;
begin
  // Geral
  ildChaveGeral.Active := True;
  if (not ibtChaveGeral.Active) then
    ibtChaveGeral.Active := True;

  // Energia
  lbQe1.Color := clOn;
  lbQe2.Color := clOn;
  lbQe3.Color := clOn;
  lbQe4.Color := clOn;
  lbQe5.Color := clOn;
  lbQe6.Color := clOn;
  lbQe7.Color := clOn;
  ipnQe.BackGroundColor := clOn;

  // Pressão
  iAg1.BackGroundColor := clOn;
  iAg2.BackGroundColor := clOn;
  iAg3.BackGroundColor := clOn;
  iAg4.BackGroundColor := clOn;
  gbP1.Color := clOn;
  gbP2.Color := clOn;
  gbP3.Color := clOn;
  gbP4.Color := clOn;
  ipnCp.BackGroundColor := clOn;

  // Temperatura
  iT1.BackGroundColor := clOn;
  iT2.BackGroundColor := clOn;
  iT3.BackGroundColor := clOn;
  iT4.BackGroundColor := clOn;
  gbT1.Color := clOn;
  gbT2.Color := clOn;
  gbT3.Color := clOn;
  gbT4.Color := clOn;
  ipnCt.BackGroundColor := clOn;

  // Gas
  ilg1.BackGroundColor := clOn;
  ilg2.BackGroundColor := clOn;
  ilg3.BackGroundColor := clOn;
  ipnG.BackGroundColor := clOn;
  FadeInGas;
end;

procedure TfmMain.SetControlsOff;
begin
  // Geral
  ildChaveGeral.Active := False;
  ibtChaveGeral.Active := False;

  // Energia
  lbQe1.Color := clOff;
  lbQe2.Color := clOff;
  lbQe3.Color := clOff;
  lbQe4.Color := clOff;
  lbQe5.Color := clOff;
```

```
  lbQe6.Color := clOff;
  lbQe7.Color := clOff;
  ipnQe.BackGroundColor := clOff;

  // Pressão
  iAg1.BackGroundColor := clOff;
  iAg2.BackGroundColor := clOff;
  iAg3.BackGroundColor := clOff;
  iAg4.BackGroundColor := clOff;
  gbP1.Color := clOff;
  gbP2.Color := clOff;
  gbP3.Color := clOff;
  gbP4.Color := clOff;
  ipnCp.BackGroundColor := clOff;

  // Temperatura
  iT1.BackGroundColor := clOff;
  iT2.BackGroundColor := clOff;
  iT3.BackGroundColor := clOff;
  iT4.BackGroundColor := clOff;
  gbT1.Color := clOff;
  gbT2.Color := clOff;
  gbT3.Color := clOff;
  gbT4.Color := clOff;
  ipnCt.BackGroundColor := clOff;

  // Gas
  ilg1.BackGroundColor := clOff;
  ilg2.BackGroundColor := clOff;
  ilg3.BackGroundColor := clOff;
  ipnG.BackGroundColor := clOff;
  FadeOutGas;

  SetQeOff;
  SetPressureControlOff;
  SetTemperatureControlOff;

end;

procedure TfmMain.SetPressureControlOn;
begin
  if (not ibtChaveGeral.Active) then
  begin
    ibtchavePressao.Active := False;
    Exit;
  end;

  if (not ibtChavePressao.Active) then
    ibtChavePressao.Active := True;

  ildPressao.Active := True;
  FadeInPressure;
end;

procedure TfmMain.SetPressureControlOff;
begin
  if (ibtChavePressao.Active) then
    ibtChavePressao.Active := False;

  ildPressao.Active := False;
  FadeOutPressure;
end;

procedure TfmMain.SetTemperatureControlOn;
begin
  if (not ibtChaveGeral.Active) then
  begin
    ibtChaveAquecedores.Active := False;
    Exit;
  end;

  if (not ibtChaveAquecedores.Active) then
    ibtChaveAquecedores.Active := True;

  ildAquecedores.Active := True;
  FadeInTemperature;
end;

procedure TfmMain.SetTemperatureControlOff;
begin
  if (ibtChaveAquecedores.Active) then
    ibtChaveAquecedores.Active := False;

  ildAquecedores.Active := False;
  FadeOutTemperature;
end;

procedure TfmMain.SetQeOff;
begin
  SetQe1Off;
  SetQe2Off;
  SetQe3Off;
  SetQe4Off;
end;

procedure TfmMain.SetQe1On;
begin
  if (not ibtChaveGeral.Active) then
  begin
    ibtQe1.Active := False;
    Exit;
  end;

  if (not ibtQe1.Active) then
    ibtQe1.Active := True;

  ildVv1.PowerOff := False;
  ildVa1.PowerOff := False;
  RandomQe1;
end;

procedure TfmMain.SetQe1Off;
begin
  if (ibtQe1.Active) then
    ibtQe1.Active := False;

  ildVv1.PowerOff := True;
  ildVa1.PowerOff := True;
end;

procedure TfmMain.SetQe2On;
begin
  if (not ibtChaveGeral.Active) then
  begin
    ibtQe2.Active := False;
    Exit;
  end;

  if (not ibtQe2.Active) then
    ibtQe2.Active := True;

  ildVv2.PowerOff := False;
  ildVa2.PowerOff := False;
  RandomQe2;
end;

procedure TfmMain.SetQe2Off;
begin
  if (ibtQe2.Active) then
    ibtQe2.Active := False;

  ildVv2.PowerOff := True;
  ildVa2.PowerOff := True;
end;

procedure TfmMain.SetQe3On;
begin
  if (not ibtChaveGeral.Active) then
  begin
    ibtQe3.Active := False;
    Exit;
  end;

  if (not ibtQe3.Active) then
    ibtQe3.Active := True;

  ildVv3.PowerOff := False;
  ildVa3.PowerOff := False;
  RandomQe3;
end;

procedure TfmMain.SetQe3Off;
begin
  if (ibtQe3.Active) then
    ibtQe3.Active := False;

  ildVv3.PowerOff := True;
  ildVa3.PowerOff := True;
end;

procedure TfmMain.SetQe4On;
begin
  if (not ibtChaveGeral.Active) then
  begin
    ibtQe4.Active := False;
    Exit;
  end;

  if (not ibtQe4.Active) then
    ibtQe4.Active := True;

  ildVv4.PowerOff := False;
  ildVa4.PowerOff := False;
  RandomQe4;
end;

procedure TfmMain.SetQe4Off;
begin
  if (ibtQe4.Active) then
    ibtQe4.Active := False;

  ildVv4.PowerOff := True;
  ildVa4.PowerOff := True;
end;

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

procedure TfmMain.FormCreate(Sender: TObject);
begin
  SetControlsOff;
  cbComPort.ItemIndex := 1;
  cbComPortChange(Sender);
end;

procedure TfmMain.iBtChaveGeralChange(Sender: TObject);
begin
  if ibtChaveGeral.Active then
    SetControlsOn
  else
    SetControlsOff;
end;

procedure TfmMain.ibtChavePressaoChange(Sender: TObject);
begin
  if ibtChavePressao.Active then
    SetPressureControlOn
  else
    SetPressureControlOff;
end;

procedure TfmMain.ibtChaveAquecedoresChange(Sender: TObject);
begin
  if ibtChaveAquecedores.Active then
    SetTemperatureControlOn
  else
    SetTemperatureControlOff;
end;

procedure TfmMain.ibtQe1Change(Sender: TObject);
begin
  if ibtQe1.Active then
    SetQe1On
  else
    SetQe1Off;
end;
```

```
procedure TfmMain.iBtQe2Change(Sender: TObject);
begin
  if ibtQe2.Active then
    SetQe2On
  else
    SetQe2Off;
end;

procedure TfmMain.iBtQe3Change(Sender: TObject);
begin
  if ibtQe3.Active then
    SetQe3On
  else
    SetQe3Off;
end;

procedure TfmMain.iBtQe4Change(Sender: TObject);
begin
  if ibtQe4.Active then
    SetQe4On
  else
    SetQe4Off;
end;

procedure TfmMain.cbComPortChange(Sender: TObject);
begin
  if ComPort.Connected then
    Exit;

  case cbComPort.ItemIndex of
    0: ComPort.Port := COM1;
    1: ComPort.Port := COM2;
    2: ComPort.Port := COM3;
    4: ComPort.Port := COM4;
  end;
end;

procedure TfmMain.ibtOpenPortChange(Sender: TObject);
begin
  try
    if (ibtOpenPort.Active) then
    begin
      ComPort.Open;
      ReceiveData;
    end
    else
      ComPort.Close;
  finally
    ibtOpenPort.Active := ComPort.Connected;
  end;
end;

procedure TfmMain.btExecutarClick(Sender: TObject);
begin
  if  (Executar(edComandos.Text)) then
  begin
    ildOk.Active := True;
    Wait(1000);
    ildOk.Active := False;
  end
  else
  begin
    ildNotOk.Active := True;
    Wait(1000);
    ildNotOk.Active := False;
  end;
end;

procedure TfmMain.btLimparClick(Sender: TObject);
begin
  edComandos.Text := '';
  edComandos.SetFocus;
end;

procedure TfmMain.TimerRandomPTimer(Sender: TObject);
begin
  RandomPressure;
end;

procedure TfmMain.TimerRandomTTimer(Sender: TObject);
begin
  RandomTemperature;
end;

procedure TfmMain.TimerRandomETimer(Sender: TObject);
begin
  RandomPower;
end;

procedure TfmMain.DisplayTX;
begin
  iLdTx.Active := True;
  Wait(10);
  iLdTx.Active := False;
end;

procedure TfmMain.DisplayRX;
begin
  iLdRx.Active := True;
  ildRx.Active := False;
end;

procedure TfmMain.FormClose(Sender: TObject; var Action:
TCloseAction);
begin
  ComPort.Close;
end;

procedure TfmMain.GetPowerStatus;
begin
  Data := 'QUADRO DE ENERGIA' + Chr($0B) +
          'ZONA 1 = ' + FloatToStr(Round(iLdVv1.Value)) + ' V - ' +
FloatToStr(Round(iLdVa1.Value)) + ' A' + Chr($0B) +
          'ZONA 2 = ' + FloatToStr(Round(iLdVv2.Value)) + ' V - ' +
FloatToStr(Round(iLdVa2.Value)) + ' A' + Chr($0B) +
          'ZONA 3 = ' + FloatToStr(Round(iLdVv3.Value)) + ' V - ' +
FloatToStr(Round(iLdVa3.Value)) + ' A' + Chr($0B) +
          'ZONA 4 = ' + FloatToStr(Round(iLdVv4.Value)) + ' V - ' +
FloatToStr(Round(iLdVa4.Value)) + ' A';
  edComandos.Text := 'READY TO SEND';
  Wait(15000);
  edComandos.Text := 'SENDING...';
  SendData(Data);
  edComandos.Text := 'GET POWER STATUS';
end;

procedure TfmMain.GetPressureStatus;
begin
  Data := 'CONTROLE DE PRESSAO' + Chr($0B) +
          'ZONA 1 = ' + FloatToStr(Round(iAg1.Position)) + ' PSI' +
Chr($0B) +
          'ZONA 2 = ' + FloatToStr(Round(iAg2.Position)) + ' PSI' +
Chr($0B) +
          'ZONA 3 = ' + FloatToStr(Round(iAg3.Position)) + ' PSI' +
Chr($0B) +
          'ZONA 4 = ' + FloatToStr(Round(iAg4.Position)) + ' PSI';
  edComandos.Text := 'READY TO SEND';
  Wait(15000);
  edComandos.Text := 'SENDING...';
  SendData(Data);
  edComandos.Text := 'GET PRESSURE STATUS';
end;

procedure TfmMain.GetTempStatus;
begin
  Data := 'CONTROLE DE TEMPERATURA' + Chr($0B) +
          'ZONA 1 = ' + FloatToStr(Round(iT1.Position)) + ' C' +
Chr($0B) +
          'ZONA 2 = ' + FloatToStr(Round(iT2.Position)) + ' C' +
Chr($0B) +
          'ZONA 3 = ' + FloatToStr(Round(iT3.Position)) + ' C' +
Chr($0B) +
          'ZONA 4 = ' + FloatToStr(Round(iT4.Position)) + ' C';
  edComandos.Text := 'READY TO SEND';
  Wait(15000);
  edComandos.Text := 'SENDING...';
  SendData(Data);
  edComandos.Text := 'GET TEMP STATUS';
end;

procedure TfmMain.GetGasStatus;
begin
  Data := 'MISTURA DE GASES' + Chr($0B) +
          'ZONA 1 = ' + FloatToStr(Round(ilg1.Position)) + ' C' +
Chr($0B) +
          'ZONA 2 = ' + FloatToStr(Round(ilg2.Position)) + ' C' +
Chr($0B) +
          'ZONA 3 = ' + FloatToStr(Round(ilg3.Position)) + ' C';
  edComandos.Text := 'READY TO SEND';
  Wait(15000);
  edComandos.Text := 'SENDING...';
  SendData(Data);
  edComandos.Text := 'GET GAS STATUS';
end;


procedure TfmMain.TimerDelayTimer(Sender: TObject);
begin
  TimerDelay.Enabled := False;
end;

procedure TfmMain.Wait(Time: Integer);
begin
  TimerDelay.Enabled := False;
  Application.ProcessMessages;

  TimerDelay.Interval := Time;
  TimerDelay.Enabled := True;
  while (TimerDelay.Enabled) do
    Application.ProcessMessages;
end;

procedure TfmMain.TimerRandomGTimer(Sender: TObject);
begin
  RandomGas;
end;

end.
```

Código fonte do Monitor Industrial

(interpretação comandos – parte 2/5)

```
unit UExec;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
Dialogs, StdCtrls,
  UMain;

function Executar(Comando: string): Boolean;

implementation

function Executar(Comando: string): Boolean;
begin
  Comando := Trim(UpperCase(Comando));
  Result := True;

  // Chave geral
  if (Comando = 'LIG SISTEMA') then
    fmMain.SetControlsOn
  else
  if (Comando = 'DES SISTEMA') then
    fmMain.SetControlsOff
  else
```

```
  // Quadro energia
  if (Comando = 'LIG Z1') then
    fmMain.SetQe1On
  else
  if (Comando = 'DES Z1') then
    fmMain.SetQe1Off
  else
  if (Comando = 'LIG Z2') then
    fmMain.SetQe2On
  else
  if (Comando = 'DES Z2') then
    fmMain.SetQe2Off
  else
  if (Comando = 'LIG Z3') then
    fmMain.SetQe3On
  else
  if (Comando = 'DES Z3') then
    fmMain.SetQe3Off
  else
  if (Comando = 'LIG Z4') then
    fmMain.SetQe4On
  else
  if (Comando = 'DES Z4') then
    fmMain.SetQe4Off
  else

  // Pressurizadores
  if (Comando = 'ATI PRESSURIZADORES') then
    fmMain.SetPressureControlOn
  else
  if (Comando = 'DES PRESSURIZADORES') then
    fmMain.SetPressureControlOff
  else

  // Aquecedores
  if (Comando = 'ATI AQUECEDORES') then
    fmMain.SetTemperatureControlOn
  else
  if (Comando = 'DES AQUECEDORES') then
    fmMain.SetTemperatureControlOff
  else

  // Genérico
  if (Comando = 'INFO ENERGIA') then
    fmMain.GetPowerStatus
  else
  if (Comando = 'INFO PRESSAO') then
    fmMain.GetPressureStatus
  else
  if (Comando = 'INFO TEMPERATURA') then
    fmMain.GetTempStatus
  else
  if (Comando = 'INFO GAS') then
    fmMain.GetGasStatus

  // Nenhum comando válido
  else
    Result := False;
end;

end.
```

Código fonte do Monitor Industrial

(simulação instrumentação – parte 3/5)

```
unit USimul;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
Dialogs, StdCtrls,
  UMain;

procedure RandomPressure;
procedure RandomTemperature;
procedure RandomPower;
procedure RandomGas;
procedure RandomQe1;
procedure RandomQe2;
procedure RandomQe3;
procedure RandomQe4;
procedure FadeInPressure;
procedure FadeOutPressure;
procedure FadeInTemperature;
procedure FadeOutTemperature;
procedure FadeInGas;
procedure FadeOutGas;

implementation

procedure RandomPressure;
var
  Valor: Integer;
begin
  Valor := Random(50);
  if  (Abs(Valor - fmMain.iAg1.Position) <= 5) then
  begin
    fmMain.iAg1.Position := Valor;
  end;

  Valor := Random(50);
  if  (Abs(Valor - fmMain.iAg2.Position) <= 5) then
  begin
    fmMain.iAg2.Position := Valor;
  end;

  Valor := Random(50);
  if  (Abs(Valor - fmMain.iAg3.Position) <= 5) then
  begin
    fmMain.iAg3.Position := Valor;
  end;

  Valor := Random(50);
  if  (Abs(Valor - fmMain.iAg4.Position) <= 5) then
  begin
    fmMain.iAg4.Position := Valor;
  end;

end;

procedure RandomTemperature;
var
  Valor: Integer;
begin
  Valor := Random(100);
  if  (Abs(Valor - fmMain.iT1.Position) <= 5) then
  begin
    fmMain.iT1.Position := Valor;
  end;

  Valor := Random(100);
  if  (Abs(Valor - fmMain.iT2.Position) <= 5) then
  begin
    fmMain.iT2.Position := Valor;
  end;

  Valor := Random(100);
  if  (Abs(Valor - fmMain.iT3.Position) <= 5) then
  begin
    fmMain.iT3.Position := Valor;
  end;

  Valor := Random(100);
  if  (Abs(Valor - fmMain.iT4.Position) <= 5) then
  begin
    fmMain.iT4.Position := Valor;
  end;
end;

procedure RandomPower;
begin
  if not fmMain.iLdVv1.PowerOff then
    RandomQe1;
  if not fmMain.iLdVv1.PowerOff then
    RandomQe2;
  if not fmMain.iLdVv1.PowerOff then
    RandomQe3;
  if not fmMain.iLdVv1.PowerOff then
    RandomQe4;
end;

procedure RandomGas;
var
  Valor: Integer;
begin
  Valor := Random(100);
  if  (Abs(Valor - fmMain.ilg1.Position) <= 5) then
  begin
    fmMain.ilg1.Position := Valor;
  end;

  Valor := Random(100);
  if  (Abs(Valor - fmMain.ilg2.Position) <= 5) then
  begin
    fmMain.ilg2.Position := Valor;
  end;

  Valor := Random(100);
  if  (Abs(Valor - fmMain.ilg3.Position) <= 5) then
  begin
    fmMain.ilg3.Position := Valor;
  end;
end;

procedure RandomQe1;
begin
  fmMain.iLdVv1.Value := 210 + Sqrt(Random(21));
  fmMain.iLdVa1.Value := 7 + Sqrt(Random(8));
end;

procedure RandomQe2;
begin
  fmMain.iLdVv2.Value := 210 + Sqrt(Random(21));
  fmMain.iLdVa2.Value := 7 + Sqrt(Random(8));
end;

procedure RandomQe3;
begin
  fmMain.iLdVv3.Value := 210 + Sqrt(Random(21));
  fmMain.iLdVa3.Value := 7 + Sqrt(Random(8));
end;

procedure RandomQe4;
begin
  fmMain.iLdVv4.Value := 210 + Sqrt(Random(21));
  fmMain.iLdVa4.Value := 7 + Sqrt(Random(8));
end;

procedure FadeInPressure;
begin
  while (fmMain.iAg1.Position < fmMain.iAg1.Tag) or
   (fmMain.iAg2.Position < fmMain.iAg2.Tag) or
   (fmMain.iAg3.Position < fmMain.iAg3.Tag) or
   (fmMain.iAg4.Position < fmMain.iAg4.Tag) do
  begin
    if (fmMain.iAg1.Position < fmMain.iAg1.Tag) then
      fmMain.iAg1.Position := fmMain.iAg1.Position + 1;
    if (fmMain.iAg2.Position < fmMain.iAg2.Tag) then
      fmMain.iAg2.Position := fmMain.iAg2.Position + 1;
    if (fmMain.iAg3.Position < fmMain.iAg3.Tag) then
      fmMain.iAg3.Position := fmMain.iAg3.Position + 1;
    if (fmMain.iAg4.Position < fmMain.iAg4.Tag) then
      fmMain.iAg4.Position := fmMain.iAg4.Position + 1;

    fmMain.Wait(50);
  end;
```

```
  fmMain.iAg1.ResetMinMax;
  fmMain.iAg2.ResetMinMax;
  fmMain.iAg3.ResetMinMax;
  fmMain.iAg4.ResetMinMax;

  fmMain.TimerRandomP.Enabled := True;
end;

procedure FadeOutPressure;
begin
  fmMain.TimerRandomP.Enabled := False;

  while (fmMain.iAg1.Position > 0) or
   (fmMain.iAg2.Position > 0) or
   (fmMain.iAg3.Position > 0) or
   (fmMain.iAg4.Position > 0) do
  begin
    if (fmMain.iAg1.Position > 0) then
      fmMain.iAg1.Position := fmMain.iAg1.Position  - 1;
    if (fmMain.iAg2.Position > 0) then
      fmMain.iAg2.Position := fmMain.iAg2.Position  - 1;
    if (fmMain.iAg3.Position > 0) then
      fmMain.iAg3.Position := fmMain.iAg3.Position  - 1;
    if (fmMain.iAg4.Position > 0) then
      fmMain.iAg4.Position := fmMain.iAg4.Position  - 1;

    fmMain.Wait(50);
  end;
end;

procedure FadeInTemperature;
begin
  while (fmMain.iT1.Position < fmMain.iT1.Tag) or
   (fmMain.iT2.Position < fmMain.iT2.Tag) or
   (fmMain.iT3.Position < fmMain.iT3.Tag) or
   (fmMain.iT4.Position < fmMain.iT4.Tag) do
  begin
    if (fmMain.iT1.Position < fmMain.iT1.Tag) then
      fmMain.iT1.Position := fmMain.iT1.Position + 1;
    if (fmMain.iT2.Position < fmMain.iT2.Tag) then
      fmMain.iT2.Position := fmMain.iT2.Position + 1;
    if (fmMain.iT3.Position < fmMain.iT3.Tag) then
      fmMain.iT3.Position := fmMain.iT3.Position + 1;
    if (fmMain.iT4.Position < fmMain.iT4.Tag) then
      fmMain.iT4.Position := fmMain.iT4.Position + 1;

    fmMain.Wait(50);
  end;

  fmMain.iT1.ResetMinMax;
  fmMain.iT2.ResetMinMax;
  fmMain.iT3.ResetMinMax;
  fmMain.iT4.ResetMinMax;

  fmMain.TimerRandomT.Enabled := True;
end;

procedure FadeOutTemperature;
begin
  fmMain.TimerRandomT.Enabled := False;

  while (fmMain.iT1.Position > 0) or
   (fmMain.iT2.Position > 0) or
   (fmMain.iT3.Position > 0) or
   (fmMain.iT4.Position > 0) do
  begin
    if (fmMain.iT1.Position > 0) then
      fmMain.iT1.Position := fmMain.iT1.Position  - 1;
    if (fmMain.iT2.Position > 0) then
      fmMain.iT2.Position := fmMain.iT2.Position  - 1;
    if (fmMain.iT3.Position > 0) then
      fmMain.iT3.Position := fmMain.iT3.Position  - 1;
    if (fmMain.iT4.Position > 0) then
      fmMain.iT4.Position := fmMain.iT4.Position  - 1;

    fmMain.Wait(50);
   end;
end;

procedure FadeInGas;
begin
  while (fmMain.ilg1.Position < fmMain.ilg1.Tag) or
   (fmMain.ilg2.Position < fmMain.ilg2.Tag) or
   (fmMain.ilg3.Position < fmMain.ilg3.Tag) do
  begin
    if (fmMain.ilg1.Position < fmMain.ilg1.Tag) then
      fmMain.ilg1.Position := fmMain.ilg1.Position + 1;
    if (fmMain.ilg2.Position < fmMain.ilg2.Tag) then
      fmMain.ilg2.Position := fmMain.ilg2.Position + 1;
    if (fmMain.ilg3.Position < fmMain.ilg3.Tag) then
      fmMain.ilg3.Position := fmMain.ilg3.Position + 1;
    fmMain.Wait(50);
  end;

  fmMain.TimerRandomG.Enabled := True;
end;

procedure FadeOutGas;
begin
  fmMain.TimerRandomG.Enabled := False;

  while (fmMain.ilg1.Position > 0) or
   (fmMain.ilg2.Position > 0) or
   (fmMain.ilg3.Position > 0) do
  begin
    if (fmMain.ilg1.Position > 0) then
      fmMain.ilg1.Position := fmMain.ilg1.Position  - 1;
    if (fmMain.ilg2.Position > 0) then
      fmMain.ilg2.Position := fmMain.ilg2.Position  - 1;
    if (fmMain.ilg3.Position > 0) then
      fmMain.ilg3.Position := fmMain.ilg3.Position  - 1;
    fmMain.Wait(50);
  end;
end;

end.
```

Código fonte do Monitor Industrial

(comunicação serial – parte 4/5)

```
unit UComm;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
Dialogs, StdCtrls,
  UMain;

const
  cHexValues: array[0..15] of Char =
('0','1','2','3','4','5','6','7','8','9','A','B','C','D','E','F');

function HexToAsc(Hex: string): Byte;
function AscToHex(Asc: Byte): string;
procedure ReceiveData;
procedure SendData(BufferData: string);
function CheckSum(Data: string): Byte;

implementation

function HexToAsc(Hex: string): Byte;
var
  A,B,Ind: Byte;
begin
  try
    Hex := UpperCase(Hex);
    A := 0;
    B := 0;

    for Ind := 0 to SizeOf(cHexValues) - 1 do
    begin
      if (cHexValues[Ind] = Hex[1]) then
        A := Ind * 16;
      if (cHexValues[Ind] = Hex[2]) then
        B := Ind;
    end;
    Result := A + B;
  except
    Result := 0;
  end;
end;

function AscToHex(Asc: Byte): string;
begin
  try
    Result := cHexValues[Asc div 16] + cHexValues[Asc mod 16];
  except
    Result := '00';
  end;
end;

procedure ReceiveData;
var
  Buffer: Char;
  CheckSumOK: Char;
  CheckSum: Boolean;
  BufferData: string;
begin
  CheckSumOK := Chr($BD);

  fmMain.ComPort.ClearBuffer(True, True);
  CheckSum := False;
  BufferData := '';

  while (fmMain.ComPort.Connected) do
  begin
    Buffer := Chr(0);
    if (fmMain.ComPort.Read(Buffer, 1) > 0) then
    begin
      fmMain.DisplayRx;
      if (Buffer = Chr($8A)) or (BufferData = 'GAFA'#13#10) then
        BufferData := '';

      if (CheckSum) then
      begin
        fmMain.ComPort.Write(CheckSumOK, 1);
        Delete(BufferData, 1, 6);
        Delete(BufferData, Length(BufferData), 1);
        fmMain.edComandos.Text := BufferData;
        fmMain.btExecutarClick(nil);
        CheckSum := False;
      end;

      BufferData := BufferData + Buffer;

      if (Buffer = Chr($FF)) then
        CheckSum := True;
    end;
    Application.ProcessMessages;
  end;
end;

procedure SendData(BufferData: string);
var
  Buffer: Char;
  Ind: Integer;
begin
  BufferData := Chr($05) + Chr($01) + Chr($01) + Chr($62) + Chr($00)
+ BufferData + Chr($FF);
  BufferData := Chr($8A) + BufferData + Chr(CheckSum(BufferData));

  for Ind := 1 to Length(BufferData) do
  begin
    Buffer := BufferData[Ind];
    fmMain.ComPort.Write(Buffer, 1);
    fmMain.DisplayTx;
  end;
end;
```

```
function CheckSum(Data: string): Byte;
var
  Ind: Integer;
  ByteSum: Integer;
  A, B: Byte;
begin
  ByteSum := 0;

  for Ind := 1 to Length(Data) do
    ByteSum := ByteSum + Ord(Data[Ind]);

  A := (ByteSum div 16) mod 16;
  B := (ByteSum mod 16);

  Result := HexToAsc(cHexValues[A] + cHexValues[B]);
end;


end.
```

Código fonte do Monitor Industrial

(rotinas de comunicação – parte 5/5)

```
unit CPort;

interface

uses
  Windows, Messages, Classes, SysUtils;

type
  TPortType = (COM1, COM2, COM3, COM4, COM5, COM6, COM7, COM8);
  TBaudRate = (br110, br300, br600, br1200, br2400, br4800, br9600,
               br14400, br19200, br38400, br56000, br57600,
br115200);
  TStopBits = (sbOneStopBit, sbOne5StopBits, sbTwoStopBits);
  TDataBits = (dbFive, dbSix, dbSeven, dbEight);
  TParityBits = (prNone, prOdd, prEven, prMark, prSpace);
  TDTRFlowControl = (dtrDisable, dtrEnable, dtrHandshake);
  TRTSFlowControl = (rtsDisable, rtsEnable, rtsHandshake, rtsToggle);
  TEvent = (evRxChar, evTxEmpty, evRxFlag, evRing, evBreak, evCTS,
            evDSR, evError, evRLSD, evRx80Full);
  TEvents = set of TEvent;
  TModemSignal = (msCTS, msDSR, msRing, msRLSD);
  TModemSignals = set of TModemSignal;
  TComError = (ceFrame, ceRxParity, ceOverrun, ceBreak, ceIO, ceMode,
               ceRxOver, ceTxFull);
  TComErrors = set of TComError;
  TSyncMethod = (smThreadSync, smWindowSync, smNone);
  TRxCharEvent = procedure(Sender: TObject; Count: Integer) of
object;
  TErrorEvent = procedure(Sender: TObject; Errors: TComErrors) of
object;
  TSignalEvent = procedure(Sender: TObject; OnOff: Boolean) of
object;

  TOperationKind = (okWrite, okRead);
  TAsync = record
    Overlapped: TOverlapped;
    Kind: TOperationKind;
  end;
  PAsync = ^TAsync;

  TComPort = class;

  TComThread = class(TThread)
  private
    FComPort: TComPort;
    FMask: DWORD;
    FStopEvent: THandle;
  protected
    procedure Execute; override;
    procedure DoEvents;
    procedure SendEvents;
    procedure DispatchComMsg;
    procedure Stop;
  public
    constructor Create(AComPort: TComPort);
    destructor Destroy; override;
  end;

  TComTimeouts = class(TPersistent)
  private
    FComPort: TComPort;
    FReadInterval: Integer;
    FReadTotalM: Integer;
    FReadTotalC: Integer;
    FWriteTotalM: Integer;
    FWriteTotalC: Integer;
    procedure SetReadInterval(Value: Integer);
    procedure SetReadTotalM(Value: Integer);
    procedure SetReadTotalC(Value: Integer);
    procedure SetWriteTotalM(Value: Integer);
    procedure SetWriteTotalC(Value: Integer);
  protected
    procedure AssignTo(Dest: TPersistent); override;
  public
    constructor Create;
  published
    property ReadInterval: Integer read FReadInterval write
SetReadInterval;
    property ReadTotalMultiplier: Integer read FReadTotalM write
SetReadTotalM;
    property ReadTotalConstant: Integer read FReadTotalC write
SetReadTotalC;
    property WriteTotalMultiplier: Integer read FWriteTotalM write
SetWriteTotalM;
    property WriteTotalConstant: Integer read FWriteTotalC write
SetWriteTotalC;
  end;
```

```
  TComFlowControl = class(TPersistent)
  private
    FComPort: TComPort;
    FOutCTSFlow: Boolean;
    FOutDSRFlow: Boolean;
    FControlDTR: TDTRFlowControl;
    FControlRTS: TRTSFlowControl;
    FXonXoffOut: Boolean;
    FXonXoffIn:  Boolean;
    FDSRSensitivity: Boolean;
    FTxContinueOnXoff: Boolean;
    FXonChar: Byte;
    FXoffChar: Byte;
    procedure SetOutCTSFlow(Value: Boolean);
    procedure SetOutDSRFlow(Value: Boolean);
    procedure SetControlDTR(Value: TDTRFlowControl);
    procedure SetControlRTS(Value: TRTSFlowControl);
    procedure SetXonXoffOut(Value: Boolean);
    procedure SetXonXoffIn(Value: Boolean);
    procedure SetDSRSensitivity(Value: Boolean);
    procedure SetTxContinueOnXoff(Value: Boolean);
    procedure SetXonChar(Value: Byte);
    procedure SetXoffChar(Value: Byte);
  protected
    procedure AssignTo(Dest: TPersistent); override;
  public
    constructor Create;
  published
    property OutCTSFlow: Boolean read FOutCTSFlow write
SetOutCTSFlow;
    property OutDSRFlow: Boolean read FOutDSRFlow write
SetOutDSRFlow;
    property ControlDTR: TDTRFlowControl read FControlDTR write
SetControlDTR;
    property ControlRTS: TRTSFlowControl read FControlRTS write
SetControlRTS;
    property XonXoffOut: Boolean read FXonXoffOut write
SetXonXoffOut;
    property XonXoffIn:  Boolean read FXonXoffIn write SetXonXoffIn;
    property DSRSensitivity: Boolean
      read FDSRSensitivity write SetDSRSensitivity;
    property TxContinueOnXoff: Boolean
      read FTxContinueOnXoff write SetTxContinueOnXoff;
    property XonChar: Byte read FXonChar write SetXonChar;
    property XoffChar: Byte read FXoffChar write SetXoffChar;
  end;

  TComParity = class(TPersistent)
  private
    FComPort: TComPort;
    FBits: TParityBits;
    FCheck: Boolean;
    FReplace: Boolean;
    FReplaceChar: Byte;
    procedure SetBits(Value: TParityBits);
    procedure SetCheck(Value: Boolean);
    procedure SetReplace(Value: Boolean);
    procedure SetReplaceChar(Value: Byte);
  protected
    procedure AssignTo(Dest: TPersistent); override;
  public
    constructor Create;
  published
    property Bits: TParityBits read FBits write SetBits;
    property Check: Boolean read FCheck write SetCheck;
    property Replace: Boolean read FReplace write SetReplace;
    property ReplaceChar: Byte read FReplaceChar write
SetReplaceChar;
  end;

  TComBuffer = class(TPersistent)
  private
    FComPort: TComPort;
    FInputSize: DWORD;
    FOutputSize: DWORD;
    procedure SetInputSize(Value: DWORD);
    procedure SetOutputSize(Value: DWORD);
  protected
    procedure AssignTo(Dest: TPersistent); override;
  public
    constructor Create;
  published
    property InputSize: DWORD read FInputSize write SetInputSize;
    property OutputSize: DWORD read FOutputSize write SetOutputSize;
  end;

  TComPort = class(TComponent)
  private
    FEventThread: TComThread;
    FThreadCreated: Boolean;
    FHandle: THandle;
    FWindow: THandle;
    FConnected: Boolean;
    FBaudRate: TBaudRate;
    FPort: TPortType;
    FStopBits: TStopBits;
    FDataBits: TDataBits;
    FDiscardNull: Boolean;
    FEventChar: Byte;
    FEvents: TEvents;
    FBuffer: TComBuffer;
    FParity: TComParity;
    FTimeouts: TComTimeouts;
    FFlowControl: TComFlowControl;
    FSyncMethod: TSyncMethod;
    FOnRxChar: TRxCharEvent;
    FOnTxEmpty: TNotifyEvent;
    FOnBreak: TNotifyEvent;
    FOnRing: TNotifyEvent;
    FOnCTSChange: TSignalEvent;
    FOnDSRChange: TSignalEvent;
    FOnRLSDChange: TSignalEvent;
    FOnError: TErrorEvent;
    FOnRxFlag: TNotifyEvent;
    FOnOpen: TNotifyEvent;
    FOnClose: TNotifyEvent;
    FOnRx80Full: TNotifyEvent;
    procedure SetConnected(Value: Boolean);
```

```
    procedure SetBaudRate(Value: TBaudRate);
    procedure SetPort(Value: TPortType);
    procedure SetStopBits(Value: TStopBits);
    procedure SetDataBits(Value: TDataBits);
    procedure SetDiscardNull(Value: Boolean);
    procedure SetEventChar(Value: Byte);
    procedure SetSyncMethod(Value: TSyncMethod);
    procedure SetParity(Value: TComParity);
    procedure SetTimeouts(Value: TComTimeouts);
    procedure SetBuffer(Value: TComBuffer);
    procedure SetFlowControl(Value: TComFlowControl);
    procedure DoOnRxChar;
    procedure DoOnTxEmpty;
    procedure DoOnBreak;
    procedure DoOnRingChange;
    procedure DoOnRxFlag;
    procedure DoOnCTSChange;
    procedure DoOnDSRChange;
    procedure DoOnError;
    procedure DoOnRLSDChange;
    procedure DoOnRx80Full;
    function ErrorCode(AsyncPtr: PAsync): Integer;
    function ComString: String;
    procedure WindowMethod(var Message: TMessage);
  protected
    procedure CreateHandle; virtual;
    procedure DestroyHandle; virtual;
    procedure ApplyDCB;
    procedure ApplyTimeouts;
    procedure ApplyBuffer;
    procedure SetupComPort; virtual;
  public
    property Connected: Boolean read FConnected write SetConnected;
    property Handle: THandle read FHandle;

    procedure Open;
    procedure Close;
    procedure ShowSetupDialog;

    function InputCount: DWORD;
    function OutputCount: DWORD;
    function ModemSignals: TModemSignals;
    function StateFlags: TComStateFlags;
    procedure SetDTR(OnOff: Boolean);
    procedure SetRTS(OnOff: Boolean);
    procedure SetXonXoff(OnOff: Boolean);
    procedure SetBreak(OnOff: Boolean);
    procedure ClearBuffer(Input, Output: Boolean);
    function LastErrors: TComErrors;

    function Write(const Buffer; Count: DWORD): DWORD;
    function WriteStr(Str: String): DWORD;
    function Read(var Buffer; Count: DWORD): DWORD;
    function ReadStr(var Str: String; Count: DWORD): DWORD;
    function WriteAsync(const Buffer; Count: DWORD; var AsyncPtr:
PAsync): DWORD;
    function WriteStrAsync(Str: String; var AsyncPtr: PAsync): DWORD;
    function ReadAsync(var Buffer; Count: DWORD; var AsyncPtr:
PAsync): DWORD;
    function ReadStrAsync(var Str: String; Count: DWORD; var
AsyncPtr: PAsync): DWORD;
    function WaitForAsync(var AsyncPtr: PAsync): DWORD;
    function IsAsyncCompleted(AsyncPtr: PAsync): Boolean;
    procedure AbortAllAsync;
    procedure TransmitChar(Ch: Char);

    constructor Create(AOwner: TComponent); override;
    destructor Destroy; override;
  published
    property BaudRate: TBaudRate read FBaudRate write SetBaudRate;
    property Port: TPortType read FPort write SetPort;
    property Parity: TComParity read FParity write SetParity;
    property StopBits: TStopBits read FStopBits write SetStopBits;
    property DataBits: TDataBits read FDataBits write SetDataBits;
    property DiscardNull: Boolean read FDiscardNull write
SetDiscardNull;
    property EventChar: Byte read FEventChar write SetEventChar;
    property Events: TEvents read FEvents write FEvents;
    property Buffer: TComBuffer read FBuffer write SetBuffer;
    property FlowControl: TComFlowControl read FFlowControl write
SetFlowControl;
    property Timeouts: TComTimeouts read FTimeouts write SetTimeouts;
    property SyncMethod: TSyncMethod read FSyncMethod write
SetSyncMethod;
    property OnRxChar: TRxCharEvent read FOnRxChar write FOnRxChar;
    property OnTxEmpty: TNotifyEvent read FOnTxEmpty write
FOnTxEmpty;
    property OnBreak: TNotifyEvent read FOnBreak write FOnBreak;
    property OnRing: TNotifyEvent read FOnRing write FOnRing;
    property OnCTSChange: TSignalEvent read FOnCTSChange write
FOnCTSChange;
    property OnDSRChange: TSignalEvent read FOnDSRChange write
FOnDSRChange;
    property OnRLSDChange: TSignalEvent read FOnRLSDChange write
FOnRLSDChange;
    property OnRxFlag: TNotifyEvent read FOnRxFlag write FOnRxFlag;
    property OnError: TErrorEvent read FOnError write FOnError;
    property OnOpen: TNotifyEvent read FOnOpen write FOnOpen;
    property OnClose: TNotifyEvent read FOnClose write FOnClose;
    property OnRx80Full: TNotifyEvent read FOnRx80Full
                                      write FOnRx80Full;
  end;

  EComPort = class(Exception)
  private
    FWinCode: Integer;
    FCode: Integer;
  public
    property WinCode: Integer read FWinCode write FWinCode;
    property Code: Integer read FCode write FCode;

    constructor Create(ACode: Integer; AWinCode: Integer);
    constructor CreateNoWinCode(ACode: Integer);
  end;

procedure InitAsync(var AsyncPtr: PAsync);
procedure DoneAsync(var AsyncPtr: PAsync);
```

```
const
  dcb_Binary           = $00000001;
  dcb_Parity           = $00000002;
  dcb_OutxCTSFlow      = $00000004;
  dcb_OutxDSRFlow      = $00000008;
  dcb_DTRControl       = $00000030;
  dcb_DSRSensivity     = $00000040;
  dcb_TxContinueOnXoff = $00000080;
  dcb_OutX             = $00000100;
  dcb_InX              = $00000200;
  dcb_ErrorChar        = $00000400;
  dcb_Null             = $00000800;
  dcb_RTSControl       = $00003000;
  dcb_AbortOnError     = $00004000;

  CM_COMPORT           = WM_USER + 1;

// error codes
  CError_OpenFailed      = 1;
  CError_WriteFailed     = 2;
  CError_ReadFailed      = 3;
  CError_InvalidAsync    = 4;
  CError_PurgeFailed     = 5;
  CError_AsyncCheck      = 6;
  CError_SetStateFailed  = 7;
  CError_TimeoutsFailed  = 8;
  CError_SetupComFailed  = 9;
  CError_ClearComFailed  = 10;
  CError_ModemStatFailed = 11;
  CError_EscapeComFailed = 12;
  CError_TransmitFailed  = 13;
  CError_SyncMeth        = 14;

// error messages
  ComErrorMessages: array[1..14] of String =
    ('Unable to open com port',
     'WriteFile function failed',
     'ReadFile function failed',
     'Invalid Async parameter',
     'PurgeComm function failed',
     'Unable to get async status',
     'SetCommState function failed',
     'SetCommTimeouts failed',
     'SetupComm function failed',
     'ClearCommError function failed',
     'GetCommModemStatus function failed',
     'EscapeCommFunction function failed',
     'TransmitCommChar function failed',
     'Cannot set SyncMethod while connected');

procedure Register;

implementation

{$R CPortImg.res}

uses
 { DsgnIntf,} CPortFrm, Controls, Forms;

//type
//  TComPortEditor = class(TComponentEditor)
//  public
//    procedure ExecuteVerb(Index: Integer); override;
//    function GetVerb(Index: Integer): string; override;
//    function GetVerbCount: Integer; override;
//  end;

function GetTOValue(Value: Integer): DWORD;
begin
  if Value = -1 then
    Result := MAXDWORD
  else
    Result := Value;
end;

procedure InitAsync(var AsyncPtr: PAsync);
begin
  New(AsyncPtr);
  with AsyncPtr^ do begin
    FillChar(Overlapped, SizeOf(TOverlapped), 0);
    Overlapped.hEvent := CreateEvent(nil, True, True, nil);
  end;
end;

procedure DoneAsync(var AsyncPtr: PAsync);
begin
  with AsyncPtr^ do
    CloseHandle(Overlapped.hEvent);
  Dispose(AsyncPtr);
  AsyncPtr := nil;
end;

// TComThread

constructor TComThread.Create(AComPort: TComPort);
var
  AMask: DWORD;
begin
  inherited Create(True);
  FStopEvent := CreateEvent(nil, True, False, nil);
  FComPort := AComPort;
  AMask := 0;
  if evRxChar in FComPort.FEvents then AMask := AMask or EV_RXCHAR;
  if evRxFlag in FComPort.FEvents then AMask := AMask or EV_RXFLAG;
  if evTxEmpty in FComPort.FEvents then AMask := AMask or EV_TXEMPTY;
  if evRing in FComPort.FEvents then AMask := AMask or EV_RING;
  if evCTS in FComPort.FEvents then AMask := AMask or EV_CTS;
  if evDSR in FComPort.FEvents then AMask := AMask or EV_DSR;
  if evRLSD in FComPort.FEvents then AMask := AMask or EV_RLSD;
  if evError in FComPort.FEvents then AMask := AMask or EV_ERR;
  if evBreak in FComPort.FEvents then AMask := AMask or EV_BREAK;
  if evRx80Full in FComPort.FEvents then AMask := AMask or
EV_RX80FULL;
  SetCommMask(FComPort.Handle, AMask);
  Resume;
end;

procedure TComThread.Execute;
```

```
var
  EventHandles: Array[0..1] of THandle;
  Overlapped: TOverlapped;
  Signaled, BytesTrans: DWORD;
begin
  FillChar(Overlapped, SizeOf(Overlapped), 0);
  Overlapped.hEvent := CreateEvent(nil, True, True, nil);
  EventHandles[0] := FStopEvent;
  EventHandles[1] := Overlapped.hEvent;
  repeat
    WaitCommEvent(FComPort.Handle, FMask, @Overlapped);
    Signaled := WaitForMultipleObjects(2, @EventHandles, False,
INFINITE);
    case Signaled of
      WAIT_OBJECT_0: Break;
      WAIT_OBJECT_0 + 1:
        if GetOverlappedResult(FComPort.Handle, Overlapped,
BytesTrans, False)
          then DispatchComMsg;
      else Break;
    end;
  until False;
  PurgeComm(FComPort.Handle, PURGE_TXABORT or PURGE_RXABORT or
     PURGE_TXCLEAR or PURGE_RXCLEAR);
  CloseHandle(Overlapped.hEvent);
  CloseHandle(FStopEvent);
end;

procedure TComThread.Stop;
begin
  SetEvent(FStopEvent);
end;

destructor TComThread.Destroy;
begin
  Stop;
  inherited Destroy;
end;

procedure TComThread.DispatchComMsg;
begin
  case FComPort.SyncMethod of
    smThreadSync: Synchronize(DoEvents);
    smWindowSync: SendEvents;
    smNone:       DoEvents;
  end;
end;

procedure TComThread.SendEvents;
begin
  if (EV_RXCHAR and FMask) <> 0 then
    SendMessage(FComPort.FWindow, CM_COMPORT, EV_RXCHAR, 0);
  if (EV_TXEMPTY and FMask) <> 0 then
    SendMessage(FComPort.FWindow, CM_COMPORT, EV_TXEMPTY, 0);
  if (EV_BREAK and FMask) <> 0 then
    SendMessage(FComPort.FWindow, CM_COMPORT, EV_BREAK, 0);
  if (EV_RING and FMask) <> 0 then
    SendMessage(FComPort.FWindow, CM_COMPORT, EV_RING, 0);
  if (EV_CTS and FMask) <> 0 then
    SendMessage(FComPort.FWindow, CM_COMPORT, EV_CTS, 0);
  if (EV_DSR and FMask) <> 0 then
    SendMessage(FComPort.FWindow, CM_COMPORT, EV_DSR, 0);
  if (EV_RXFLAG and FMask) <> 0 then
    SendMessage(FComPort.FWindow, CM_COMPORT, EV_RXFLAG, 0);
  if (EV_RLSD and FMask) <> 0 then
    SendMessage(FComPort.FWindow, CM_COMPORT, EV_RLSD, 0);
  if (EV_ERR and FMask) <> 0 then
    SendMessage(FComPort.FWindow, CM_COMPORT, EV_ERR, 0);
  if (EV_RX80FULL and FMask) <> 0 then
    SendMessage(FComPort.FWindow, CM_COMPORT, EV_RX80FULL, 0);
end;

procedure TComThread.DoEvents;
begin
  if (EV_RXCHAR and FMask) > 0 then FComPort.DoOnRxChar;
  if (EV_TXEMPTY and FMask) > 0 then FComPort.DoOnTxEmpty;
  if (EV_BREAK and FMask) > 0 then FComPort.DoOnBreak;
  if (EV_RING and FMask) > 0 then FComPort.DoOnRingChange;
  if (EV_CTS and FMask) > 0 then FComPort.DoOnCTSChange;
  if (EV_DSR and FMask) > 0 then FComPort.DoOnDSRChange;
  if (EV_RXFLAG and FMask) > 0 then FComPort.DoOnRxFlag;
  if (EV_RLSD and FMask) > 0 then FComPort.DoOnRLSDChange;
  if (EV_ERR and FMask) > 0 then FComPort.DoOnError;
  if (EV_RX80FULL and FMask) > 0 then FComPort.DoOnRx80Full;
end;

// TComTimeouts

constructor TComTimeouts.Create;
begin
  inherited Create;
  FReadInterval := -1;
  FWriteTotalM := 100;
  FWriteTotalC := 1000;
end;

procedure TComTimeouts.AssignTo(Dest: TPersistent);
begin
  if Dest is TComTimeouts then begin
    with TComTimeouts(Dest) do begin
      FReadInterval := Self.FReadInterval;
      FReadTotalM   := Self.FReadTotalM;
      FReadTotalC   := Self.FReadTotalC;
      FWriteTotalM  := Self.FWriteTotalM;
      FWriteTotalC  := Self.FWriteTotalC;
    end
  end
  else
    inherited AssignTo(Dest);
end;

procedure TComTimeouts.SetReadInterval(Value: Integer);
begin
  if Value <> FReadInterval then begin
    FReadInterval := Value;
    if FComPort <> nil then
      FComPort.ApplyTimeouts;
  end;
end;

procedure TComTimeouts.SetReadTotalC(Value: Integer);
begin
  if Value <> FReadTotalC then begin
    FReadTotalC := Value;
    if FComPort <> nil then
      FComPort.ApplyTimeouts;
  end;
end;

procedure TComTimeouts.SetReadTotalM(Value: Integer);
begin
  if Value <> FReadTotalM then begin
    FReadTotalM := Value;
    if FComPort <> nil then
      FComPort.ApplyTimeouts;
  end;
end;

procedure TComTimeouts.SetWriteTotalC(Value: Integer);
begin
  if Value <> FWriteTotalC then begin
    FWriteTotalC := Value;
    if FComPort <> nil then
      FComPort.ApplyTimeouts;
  end;
end;

procedure TComTimeouts.SetWriteTotalM(Value: Integer);
begin
  if Value <> FWriteTotalM then begin
    FWriteTotalM := Value;
    if FComPort <> nil then
      FComPort.ApplyTimeouts;
  end;
end;

// TComFlowControl

constructor TComFlowControl.Create;
begin
  inherited Create;
  FXonChar := 17;
  FXoffChar := 19;
end;

procedure TComFlowControl.AssignTo(Dest: TPersistent);
begin
  if Dest is TComFlowControl then begin
    with TComFlowControl(Dest) do begin
      FOutCTSFlow       := Self.FOutCTSFlow;
      FOutDSRFlow       := Self.FOutDSRFlow;
      FControlDTR       := Self.FControlDTR;
      FControlRTS       := Self.FControlRTS;
      FXonXoffOut       := Self.FXonXoffOut;
      FXonXoffIn        := Self.FXonXoffIn;
      FTxContinueOnXoff := Self.FTxContinueOnXoff;
      FDSRSensitivity   := Self.FDSRSensitivity;
      FXonChar          := Self.FXonChar;
      FXoffChar         := Self.FXoffChar;
    end
  end
  else
    inherited AssignTo(Dest);
end;

procedure TComFlowControl.SetControlDTR(Value: TDTRFlowControl);
begin
  if Value <> FControlDTR then begin
    FControlDTR := Value;
    if FComPort <> nil then
      FComPort.ApplyDCB;
  end;
end;

procedure TComFlowControl.SetControlRTS(Value: TRTSFlowControl);
begin
  if Value <> FControlRTS then begin
    FControlRTS := Value;
    if FComPort <> nil then
      FComPort.ApplyDCB;
  end;
end;

procedure TComFlowControl.SetOutCTSFlow(Value: Boolean);
begin
  if Value <> FOutCTSFlow then begin
    FOutCTSFlow := Value;
    if FComPort <> nil then
      FComPort.ApplyDCB;
  end;
end;

procedure TComFlowControl.SetOutDSRFlow(Value: Boolean);
begin
  if Value <> FOutDSRFlow then begin
    FOutDSRFlow := Value;
    if FComPort <> nil then
      FComPort.ApplyDCB;
  end;
end;

procedure TComFlowControl.SetXonXoffIn(Value: Boolean);
begin
  if Value <> FXonXoffIn then begin
    FXonXoffIn := Value;
    if FComPort <> nil then
      FComPort.ApplyDCB;
  end;
end;

procedure TComFlowControl.SetXonXoffOut(Value: Boolean);
begin
  if Value <> FXonXoffOut then begin
    FXonXoffOut := Value;
    if FComPort <> nil then
      FComPort.ApplyDCB;
```

```
  end;
end;

procedure TComFlowControl.SetDSRSensitivity(Value: Boolean);
begin
  if Value <> FDSRSensitivity then begin
    FDSRSensitivity := Value;
    if FComPort <> nil then
      FComPort.ApplyDCB;
  end;
end;

procedure TComFlowControl.SetTxContinueOnXoff(Value: Boolean);
begin
  if Value <> FTxContinueOnXoff then begin
    FTxContinueOnXoff := Value;
    if FComPort <> nil then
      FComPort.ApplyDCB;
  end;
end;

procedure TComFlowControl.SetXonChar(Value: Byte);
begin
  if Value <> FXonChar then begin
    FXonChar := Value;
    if FComPort <> nil then
      FComPort.ApplyDCB;
  end;
end;

procedure TComFlowControl.SetXoffChar(Value: Byte);
begin
  if Value <> FXoffChar then begin
    FXoffChar := Value;
    if FComPort <> nil then
      FComPort.ApplyDCB;
  end;
end;

// TComParity

constructor TComParity.Create;
begin
  inherited Create;
end;

procedure TComParity.AssignTo(Dest: TPersistent);
begin
  if Dest is TComParity then begin
    with TComParity(Dest) do begin
      FBits       := Self.FBits;
      FCheck      := Self.FCheck;
      FReplace    := Self.FReplace;
      FReplaceChar := Self.FReplaceChar;
    end
  end
  else
    inherited AssignTo(Dest);
end;

procedure TComParity.SetBits(Value: TParityBits);
begin
  if Value <> FBits then begin
    FBits := Value;
    if FComPort <> nil then
      FComPort.ApplyDCB;
  end;
end;

procedure TComParity.SetCheck(Value: Boolean);
begin
  if Value <> FCheck then begin
    FCheck := Value;
    if FComPort <> nil then
      FComPort.ApplyDCB;
  end;
end;

procedure TComParity.SetReplace(Value: Boolean);
begin
  if Value <> FReplace then begin
    FReplace := Value;
    if FComPort <> nil then
      FComPort.ApplyDCB;
  end;
end;

procedure TComParity.SetReplaceChar(Value: Byte);
begin
  if Value <> FReplaceChar then begin
    FReplaceChar := Value;
    if FComPort <> nil then
      FComPort.ApplyDCB;
  end;
end;

// TComBuffer

constructor TComBuffer.Create;
begin
  inherited Create;
  FInputSize := 1024;
  FOutputSize := 1024;
end;

procedure TComBuffer.AssignTo(Dest: TPersistent);
begin
  if Dest is TComBuffer then begin
    with TComBuffer(Dest) do begin
      FOutputSize  := Self.FOutputSize;
      FInputSize   := Self.FInputSize;
    end
  end
  else
    inherited AssignTo(Dest);
end;

procedure TComBuffer.SetInputSize(Value: DWORD);
begin
  if Value <> FInputSize then begin
    FInputSize := Value;
    if FComPort <> nil then
      FComPort.ApplyBuffer;
  end;
end;

procedure TComBuffer.SetOutputSize(Value: DWORD);
begin
  if Value <> FOutputSize then begin
    FOutputSize := Value;
    if FComPort <> nil then
      FComPort.ApplyBuffer;
  end;
end;

// TComPort

constructor TComPort.Create(AOwner: TComponent);
begin
  inherited Create(AOwner);
  FConnected := False;
  FBaudRate := br9600;
  FPort := COM1;
  FStopBits := sbOneStopBit;
  FDataBits := dbEight;
  FEvents := [evRxChar, evTxEmpty, evRxFlag, evRing, evBreak,
              evCTS, evDSR, evError, evRLSD, evRx80Full];
  FHandle := INVALID_HANDLE_VALUE;

  FParity := TComParity.Create;
  FParity.FComPort := Self;
  FFlowControl := TComFlowControl.Create;
  FFlowControl.FComPort := Self;
  FTimeouts := TComTimeouts.Create;
  FTimeouts.FComPort := Self;
  FBuffer := TComBuffer.Create;
  FBuffer.FComPort := Self;
end;

destructor TComPort.Destroy;
begin
  Close;
  FBuffer.Free;
  FFlowControl.Free;
  FTimeouts.Free;
  FParity.Free;
  inherited Destroy;
end;

procedure TComPort.CreateHandle;
begin
  FHandle := CreateFile(
    PChar(ComString),
    GENERIC_READ or GENERIC_WRITE,
    0,
    nil,
    OPEN_EXISTING,
    FILE_FLAG_OVERLAPPED,
    0);

  if FHandle = INVALID_HANDLE_VALUE then
    raise EComPort.Create(CError_OpenFailed, GetLastError);
end;

procedure TComPort.DestroyHandle;
begin
  if FHandle <> INVALID_HANDLE_VALUE then
    CloseHandle(FHandle);
end;

procedure TComPort.WindowMethod(var Message: TMessage);
begin
  with Message do
    if Msg = CM_COMPORT then begin
      case wParam of
        EV_RXCHAR:   DoOnRxChar;
        EV_TXEMPTY:  DoOnTxEmpty;
        EV_BREAK:    DoOnBreak;
        EV_RING:     DoOnRingChange;
        EV_CTS:      DoOnCTSChange;
        EV_DSR:      DoOnDSRChange;
        EV_RXFLAG:   DoOnRxFlag;
        EV_RLSD:     DoOnRLSDChange;
        EV_ERR:      DoOnError;
        EV_RX80FULL: DoOnRx80Full;
      end
    end else
      Result := DefWindowProc(FWindow, Msg, wParam, lParam);
end;

procedure TComPort.Open;
begin
  if not FConnected then begin
    CreateHandle;
    FConnected := True;
    try
      SetupComPort;
    except
      DestroyHandle;
      FConnected := False;
      raise;
    end;
    if (FEvents = []) then
      FThreadCreated := False
    else begin
      if (FSyncMethod = smWindowSync) then
        FWindow := AllocateHWnd(WindowMethod);
      FEventThread := TComThread.Create(Self);
      FThreadCreated := True;
    end;
    if Assigned(FOnOpen) then
      FOnOpen(Self);
  end;
end;

procedure TComPort.Close;
```

```
begin
  if FConnected then begin
    AbortAllAsync;
    if FThreadCreated then begin
      FEventThread.Free;
      if FSyncMethod = smWindowSync then
        DeallocateHWnd(FWindow);
    end;
    DestroyHandle;
    FConnected := False;
    if Assigned(FOnClose) then
      FOnClose(Self);
  end;
end;

procedure TComPort.ApplyDCB;
const
  CParityBits: array[TParityBits] of DWORD =
    (NOPARITY, ODDPARITY, EVENPARITY, MARKPARITY, SPACEPARITY);
  CStopBits: array[TStopBits] of DWORD =
    (ONESTOPBIT, ONE5STOPBITS, TWOSTOPBITS);
  CBaudRate: array[TBaudRate] of DWORD =
    (CBR_110, CBR_300, CBR_600, CBR_1200, CBR_2400, CBR_4800,
CBR_9600,
     CBR_14400, CBR_19200, CBR_38400, CBR_56000, CBR_57600,
CBR_115200);
  CDataBits: array[TDataBits] of DWORD = (5, 6, 7, 8);
  CControlRTS: array[TRTSFlowControl] of Integer =
    (RTS_CONTROL_DISABLE shl 12,
     RTS_CONTROL_ENABLE shl 12,
     RTS_CONTROL_HANDSHAKE shl 12,
     RTS_CONTROL_TOGGLE shl 12);
  CControlDTR: array[TDTRFlowControl] of Integer =
    (DTR_CONTROL_DISABLE shl 4,
     DTR_CONTROL_ENABLE shl 4,
     DTR_CONTROL_HANDSHAKE shl 4);

var
  DCB: TDCB;

begin
  if FConnected then begin
    DCB.XonLim := FBuffer.InputSize div 4;
    DCB.XoffLim := DCB.XonLim;
    DCB.EvtChar := Char(FEventChar);

    DCB.Flags := dcb_Binary;
    if FDiscardNull then
      DCB.Flags := DCB.Flags or dcb_Null;

    with FFlowControl do begin
      DCB.XonChar := Chr(XonChar);
      DCB.XoffChar := Chr(XoffChar);
      if OutCTSFlow then
        DCB.Flags := DCB.Flags or dcb_OutxCTSFlow;
      if OutDSRFlow then
        DCB.Flags := DCB.Flags or dcb_OutxDSRFlow;
      DCB.Flags := DCB.Flags or CControlDTR[ControlDTR]
        or CControlRTS[ControlRTS];
      if XonXoffOut then
        DCB.Flags := DCB.Flags or dcb_OutX;
      if XonXoffIn then
        DCB.Flags := DCB.Flags or dcb_InX;
      if DSRSensitivity then
        DCB.Flags := DCB.Flags or dcb_DSRSensivity;
      if TxContinueOnXoff then
        DCB.Flags := DCB.Flags or dcb_TxContinueOnXoff;
    end;

    DCB.Parity := CParityBits[FParity.Bits];
    DCB.StopBits := CStopBits[FStopBits];
    DCB.BaudRate := CBaudRate[FBaudRate];
    DCB.ByteSize := CDataBits[FDataBits];

    if FParity.Check then begin
      DCB.Flags := DCB.Flags or dcb_Parity;
      if FParity.Replace then begin
        DCB.Flags := DCB.Flags or dcb_ErrorChar;
        DCB.ErrorChar := Char(FParity.ReplaceChar);
      end;
    end;

    if not SetCommState(FHandle, DCB) then
      raise EComPort.Create(CError_SetStateFailed, GetLastError);
  end;
end;

procedure TComPort.ApplyTimeouts;
var
  Timeouts: TCommTimeouts;
begin
  if FConnected then begin
    Timeouts.ReadIntervalTimeout :=
GetTOValue(FTimeouts.ReadInterval);
    Timeouts.ReadTotalTimeoutMultiplier :=
GetTOValue(FTimeouts.ReadTotalMultiplier);
    Timeouts.ReadTotalTimeoutConstant :=
GetTOValue(FTimeouts.ReadTotalConstant);
    Timeouts.WriteTotalTimeoutMultiplier :=
GetTOValue(FTimeouts.WriteTotalMultiplier);
    Timeouts.WriteTotalTimeoutConstant :=
GetTOValue(FTimeouts.WriteTotalConstant);

    if not SetCommTimeouts(FHandle, Timeouts) then
      raise EComPort.Create(CError_TimeoutsFailed, GetLastError);
  end;
end;

procedure TComPort.ApplyBuffer;
begin
  if FConnected then
    if not SetupComm(FHandle, FBuffer.InputSize, FBuffer.OutputSize)
then
      raise EComPort.Create(CError_SetupComFailed, GetLastError);
end;

procedure TComPort.SetupComPort;
begin
  ApplyDCB;
  ApplyTimeouts;
  ApplyBuffer;
end;

function TComPort.InputCount: DWORD;
var
  Errors: DWORD;
  ComStat: TComStat;
begin
  if not ClearCommError(FHandle, Errors, @ComStat) then
    raise EComPort.Create(CError_ClearComFailed, GetLastError);
  Result := ComStat.cbInQue;
end;

function TComPort.OutputCount: DWORD;
var
  Errors: DWORD;
  ComStat: TComStat;
begin
  if not ClearCommError(FHandle, Errors, @ComStat) then
    raise EComPort.Create(CError_ClearComFailed, GetLastError);
  Result := ComStat.cbOutQue;
end;

function TComPort.ModemSignals: TModemSignals;
var
  Status: DWORD;
begin
  if not GetCommModemStatus(FHandle, Status) then
    raise EComPort.Create(CError_ModemStatFailed, GetLastError);
  Result := [];

  if (MS_CTS_ON and Status) <> 0 then
    Result := Result + [msCTS];
  if (MS_DSR_ON and Status) <> 0 then
    Result := Result + [msDSR];
  if (MS_RING_ON and Status) <> 0 then
    Result := Result + [msRing];
  if (MS_RLSD_ON and Status) <> 0 then
    Result := Result + [msRLSD];
end;

function TComPort.StateFlags: TComStateFlags;
var
  Errors: DWORD;
  ComStat: TComStat;
begin
  if not ClearCommError(FHandle, Errors, @ComStat) then
    raise EComPort.Create(CError_ClearComFailed, GetLastError);
  Result := ComStat.Flags;
end;

procedure TComPort.SetBreak(OnOff: Boolean);
var
  Act: DWORD;
begin
  if OnOff then
    Act := Windows.SETBREAK
  else
    Act := Windows.CLRBREAK;

  if not EscapeCommFunction(FHandle, Act) then
    raise EComPort.Create(CError_EscapeComFailed, GetLastError);
end;

procedure TComPort.SetDTR(OnOff: Boolean);
var
  Act: DWORD;
begin
  if OnOff then
    Act := Windows.SETDTR
  else
    Act := Windows.CLRDTR;

  if not EscapeCommFunction(FHandle, Act) then
    raise EComPort.Create(CError_EscapeComFailed, GetLastError);
end;

procedure TComPort.SetRTS(OnOff: Boolean);
var
  Act: DWORD;
begin
  if OnOff then
    Act := Windows.SETRTS
  else
    Act := Windows.CLRRTS;

  if not EscapeCommFunction(FHandle, Act) then
    raise EComPort.Create(CError_EscapeComFailed, GetLastError);
end;

procedure TComPort.SetXonXoff(OnOff: Boolean);
var
  Act: DWORD;
begin
  if OnOff then
    Act := Windows.SETXON
  else
    Act := Windows.SETXOFF;

  if not EscapeCommFunction(FHandle, Act) then
    raise EComPort.Create(CError_EscapeComFailed, GetLastError);
end;

procedure TComPort.ClearBuffer(Input, Output: Boolean);
var
  Flag: DWORD;
begin
  Flag := 0;
  if Input then
    Flag := PURGE_RXCLEAR;
  if Output then
    Flag := Flag or PURGE_TXCLEAR;

  if not PurgeComm(FHandle, Flag) then
    raise EComPort.Create(CError_PurgeFailed, GetLastError);
end;
```

```
function TComPort.LastErrors: TComErrors;
var
  Errors: DWORD;
  ComStat: TComStat;
begin
  if not ClearCommError(FHandle, Errors, @ComStat) then
    raise EComPort.Create(CError_ClearComFailed, GetLastError);
  Result := [];

  if (CE_FRAME and Errors) <> 0 then
    Result := Result + [ceFrame];
  if ((CE_RXPARITY and Errors) <> 0) and FParity.Check then // get
around a bug
    Result := Result + [ceRxParity];
  if (CE_OVERRUN and Errors) <> 0 then
    Result := Result + [ceOverrun];
  if (CE_RXOVER and Errors) <> 0 then
    Result := Result + [ceRxOver];
  if (CE_TXFULL and Errors) <> 0 then
    Result := Result + [ceTxFull];
  if (CE_BREAK and Errors) <> 0 then
    Result := Result + [ceBreak];
  if (CE_IOE and Errors) <> 0 then
    Result := Result + [ceIO];
  if (CE_MODE and Errors) <> 0 then
    Result := Result + [ceMode];
end;

function TComPort.WriteAsync(const Buffer; Count: DWORD; var
AsyncPtr: PAsync): DWORD;
var
  Success: Boolean;
  BytesTrans: DWORD;
begin
  AsyncPtr^.Kind := okWrite;

  Success := WriteFile(FHandle, Buffer, Count, BytesTrans,
@AsyncPtr^.Overlapped)
    or (GetLastError = ERROR_IO_PENDING);

  if not Success then
    raise EComPort.Create(CError_WriteFailed, GetLastError);

  Result := BytesTrans;
end;

function TComPort.Write(const Buffer; Count: DWORD): DWORD;
var
  AsyncPtr: PAsync;
begin
  InitAsync(AsyncPtr);
  try
    WriteAsync(Buffer, Count, AsyncPtr);
    Result := WaitForAsync(AsyncPtr);
  finally
    DoneAsync(AsyncPtr);
  end;
end;

function TComPort.WriteStrAsync(Str: String; var AsyncPtr: PAsync):
DWORD;
var
  Success: Boolean;
  BytesTrans: DWORD;
begin
  AsyncPtr^.Kind := okWrite;

  Success := WriteFile(FHandle, Str[1], Length(Str), BytesTrans,
@AsyncPtr^.Overlapped)
    or (GetLastError = ERROR_IO_PENDING);

  if not Success then
    raise EComPort.Create(CError_WriteFailed, GetLastError);

  Result := BytesTrans;
end;

function TComPort.WriteStr(Str: String): DWORD;
var
  AsyncPtr: PAsync;
begin
  InitAsync(AsyncPtr);
  try
    WriteStrAsync(Str, AsyncPtr);
    Result := WaitForAsync(AsyncPtr);
  finally
    DoneAsync(AsyncPtr);
  end;
end;

function TComPort.ReadAsync(var Buffer; Count: DWORD; var AsyncPtr:
PAsync): DWORD;
var
  Success: Boolean;
  BytesTrans: DWORD;
begin
  AsyncPtr^.Kind := okRead;

  Success := ReadFile(FHandle, Buffer, Count, BytesTrans,
@AsyncPtr^.Overlapped)
    or (GetLastError = ERROR_IO_PENDING);

  if not Success then
    raise EComPort.Create(CError_ReadFailed, GetLastError);

  Result := BytesTrans;
end;

function TComPort.Read(var Buffer; Count: DWORD): DWORD;
var
  AsyncPtr: PAsync;
begin
  InitAsync(AsyncPtr);
  try
    ReadAsync(Buffer, Count, AsyncPtr);
    Result := WaitForAsync(AsyncPtr);
  finally
    DoneAsync(AsyncPtr);
  end;
end;

function TComPort.ReadStrAsync(var Str: String; Count: DWORD; var
AsyncPtr: PAsync): DWORD;
var
  Success: Boolean;
  BytesTrans: DWORD;
begin
  AsyncPtr^.Kind := okRead;
  SetLength(Str, Count);

  Success := ReadFile(FHandle, Str[1], Count, BytesTrans,
@AsyncPtr^.Overlapped)
    or (GetLastError = ERROR_IO_PENDING);

  if not Success then
    raise EComPort.Create(CError_ReadFailed, GetLastError);

  Result := BytesTrans;
end;

function TComPort.ReadStr(var Str: String; Count: DWORD): DWORD;
var
  AsyncPtr: PAsync;
begin
  InitAsync(AsyncPtr);
  try
    ReadStrAsync(Str, Count, AsyncPtr);
    Result := WaitForAsync(AsyncPtr);
    SetLength(Str, Result);
  finally
    DoneAsync(AsyncPtr);
  end;
end;

function TComPort.WaitForAsync(var AsyncPtr: PAsync): DWORD;
var
  BytesTrans, Signaled: DWORD;
  Success: Boolean;
begin
  if AsyncPtr = nil then
    raise EComPort.CreateNoWinCode(CError_InvalidAsync);

  with AsyncPtr^ do begin
    Signaled := WaitForSingleObject(Overlapped.hEvent, INFINITE);
    Success := (Signaled = WAIT_OBJECT_0) and
      (GetOverlappedResult(FHandle, Overlapped, BytesTrans, False));
  end;

  if not Success then
    raise EComPort.Create(ErrorCode(AsyncPtr), GetLastError);

  Result := BytesTrans;
end;

function TComPort.ErrorCode(AsyncPtr: PAsync): Integer;
begin
  Result := 0;
  case AsyncPtr^.Kind of
    okWrite: Result := CError_WriteFailed;
    okRead:  Result := CError_ReadFailed;
  end;
end;

procedure TComPort.AbortAllAsync;
begin
  if not PurgeComm(FHandle, PURGE_TXABORT or PURGE_RXABORT) then
    raise EComPort.Create(CError_PurgeFailed, GetLastError);
end;

function TComPort.IsAsyncCompleted(AsyncPtr: PAsync): Boolean;
var
  BytesTrans: DWORD;
begin
  if AsyncPtr = nil then
    raise EComPort.CreateNoWinCode(CError_InvalidAsync);

  Result := GetOverlappedResult(FHandle, AsyncPtr^.Overlapped,
BytesTrans, False);
  if not Result then
    if GetLastError <> ERROR_IO_INCOMPLETE then
      raise EComPort.Create(CError_AsyncCheck, GetLastError);
end;

procedure TComPort.TransmitChar(Ch: Char);
begin
  if not TransmitCommChar(FHandle, Ch) then
    raise EComPort.Create(CError_TransmitFailed, GetLastError);
end;

procedure TComPort.ShowSetupDialog;
var
  Temp: TComFlowControl;
begin
  with TSerialFrm.Create(nil) do begin
    Temp := TComFlowControl.Create;
    Temp.Assign(FFlowControl);
    ComboBox1.ItemIndex := Integer(Port);
    ComboBox2.ItemIndex := Integer(BaudRate);
    ComboBox3.ItemIndex := Integer(StopBits);
    ComboBox4.ItemIndex := Integer(DataBits);
    ComboBox5.ItemIndex := Integer(Parity.Bits);
    CheckBox1.Checked := Temp.OutCTSFlow;
    CheckBox2.Checked := Temp.OutDSRFlow;
    CheckBox3.Checked := Temp.XonXoffOut;
    CheckBox4.Checked := Temp.XonXoffIn;
    RadioGroup1.ItemIndex := Integer(Temp.ControlRTS);
    RadioGroup2.ItemIndex := Integer(Temp.ControlDTR);
    if ShowModal = mrOK then begin
      Port := TPortType(ComboBox1.ItemIndex);
      BaudRate := TBaudRate(ComboBox2.ItemIndex);
      StopBits := TStopBits(ComboBox3.ItemIndex);
      DataBits := TDataBits(ComboBox4.ItemIndex);
      Parity.FBits := TParityBits(ComboBox5.ItemIndex);
      Temp.OutCTSFlow := CheckBox1.Checked;
      Temp.OutDSRFlow := CheckBox2.Checked;
```

```
      Temp.XonXoffOut := CheckBox3.Checked;
      Temp.XonXoffIn := CheckBox4.Checked;
      Temp.ControlRTS := TRTSFlowControl(RadioGroup1.ItemIndex);
      Temp.ControlDTR := TDTRFlowControl(RadioGroup2.ItemIndex);
      FlowControl := Temp;
    end;
    Temp.Free;
    Free;
  end;
end;

procedure TComPort.SetConnected(Value: Boolean);
begin
  if Value then
    Open
  else
    Close;
end;

procedure TComPort.SetBaudRate(Value: TBaudRate);
begin
  if Value <> FBaudRate then begin
    FBaudRate := Value;
    ApplyDCB;
  end;
end;

procedure TComPort.SetDataBits(Value: TDataBits);
begin
  if Value <> FDataBits then begin
    FDataBits := Value;
    ApplyDCB;
  end;
end;

procedure TComPort.SetDiscardNull(Value: Boolean);
begin
  if Value <> FDiscardNull then begin
    FDiscardNull := Value;
    ApplyDCB;
  end;
end;

procedure TComPort.SetEventChar(Value: Byte);
begin
  if Value <> FEventChar then begin
    FEventChar := Value;
    ApplyDCB;
  end;
end;

procedure TComPort.SetPort(Value: TPortType);
begin
  if Value <> FPort then begin
    FPort := Value;
    if FConnected then begin
      Close;
      Open;
    end;
  end;
end;

procedure TComPort.SetStopBits(Value: TStopBits);
begin
  if Value <> FStopBits then begin
    FStopBits := Value;
    ApplyDCB;
  end;
end;

procedure TComPort.SetSyncMethod(Value: TSyncMethod);
begin
  if Value <> FSyncMethod then begin
    if FConnected then
      raise EComPort.CreateNoWinCode(CError_SyncMeth)
    else
      FSyncMethod := Value;
  end;
end;

procedure TComPort.SetFlowControl(Value: TComFlowControl);
begin
  FFlowControl.Assign(Value);
  ApplyDCB;
end;

procedure TComPort.SetParity(Value: TComParity);
begin
  FParity.Assign(Value);
  ApplyDCB;
end;

procedure TComPort.SetTimeouts(Value: TComTimeouts);
begin
  FTimeouts.Assign(Value);
  ApplyTimeouts;
end;

procedure TComPort.SetBuffer(Value: TComBuffer);
begin
  FBuffer.AssignTo(Value);
  ApplyBuffer;
end;

procedure TComPort.DoOnRxChar;
begin
  if Assigned(FOnRxChar) then
    FOnRxChar(Self, Integer(InputCount));
end;

procedure TComPort.DoOnBreak;
begin
  if Assigned(FOnBreak) then
    FOnBreak(Self);
end;

procedure TComPort.DoOnTxEmpty;
begin
  if Assigned(FOnTxEmpty)
    then FOnTxEmpty(Self);
end;

procedure TComPort.DoOnRingChange;
begin
  if Assigned(FOnRing) then
    FOnRing(Self);
end;

procedure TComPort.DoOnCTSChange;
begin
  if Assigned(FOnCTSChange) then
    FOnCTSChange(Self, msCTS in ModemSignals);
end;

procedure TComPort.DoOnDSRChange;
begin
  if Assigned(FOnDSRChange) then
    FOnDSRChange(Self, msDSR in ModemSignals);
end;

procedure TComPort.DoOnRLSDChange;
begin
  if Assigned(FOnRLSDChange) then
    FOnRLSDChange(Self, msRLSD in ModemSignals);
end;

procedure TComPort.DoOnError;
begin
  if Assigned(FOnError) then
    FOnError(Self, LastErrors);
end;

procedure TComPort.DoOnRxFlag;
begin
  if Assigned(FOnRxFlag) then
    FOnRxFlag(Self);
end;

procedure TComPort.DoOnRx80Full;
begin
  if Assigned(FOnRx80Full) then
    FOnRx80Full(Self);
end;

function TComPort.ComString: String;
begin
  case FPort of
    COM1: Result := 'COM1';
    COM2: Result := 'COM2';
    COM3: Result := 'COM3';
    COM4: Result := 'COM4';
    COM5: Result := 'COM5';
    COM6: Result := 'COM6';
    COM7: Result := 'COM7';
    COM8: Result := 'COM8';
  end;
end;

// EComPort

constructor EComPort.Create(ACode: Integer; AWinCode: Integer);
begin
  FWinCode := AWinCode;
  FCode := ACode;
  inherited CreateFmt(ComErrorMessages[ACode] + ' (win error code:
%d)', [AWinCode]);
end;

constructor EComPort.CreateNoWinCode(ACode: Integer);
begin
  FWinCode := -1;
  FCode := ACode;
  inherited Create(ComErrorMessages[ACode]);
end;

// TComPortEditor

//procedure TComPortEditor.ExecuteVerb(Index: Integer);
//begin
//  if Index = 0 then begin
//    (Component as TComPort).ShowSetupDialog;
//    Designer.Modified;
//  end;
//end;

//function TComPortEditor.GetVerb(Index: Integer): string;
//begin
//  case Index of
//    0: Result := 'Edit Properties';
//  end;
//end;

//function TComPortEditor.GetVerbCount: Integer;
//begin
//  Result := 1;
//end;

procedure Register;
begin
  RegisterComponents('e-comp', [TComPort]);
//  RegisterComponentEditor(TComPort, TComPortEditor);
end;
end
```

ANEXO II

Código fonte do microcontrolador 89C2051 para o circuito de recepção.

```
; SISTEMA DE RECEPCAO DE DADOS VIA RF E REENVIA DADO VIA SERIAL
              ; INTERRUPCAO EXTERNA UTILIZADA
            ; MODIFICADO PARA 4F BUFFER DE RX , STACK 55H NA INT
SERIAL
            ; COLOCADO SO RETI
            ; 11/05/ ACRESCIDO RESET DE R0 1 R1 AO FIM DE RECEPCAO

$MOD51
$TITLE(BYTE SIGNED MULTIPLY)
$PAGEWIDTH(132)
$DEBUG
$OBJECT
$NOPAGING

            LED                 EQU     P1.0
            LED2                EQU     P1.1
            RX                  EQU     P3.2

            MSB_H               EQU     10H     ; CONTADOR HIGH TEMPO
HIGH
            MSB_L               EQU     11H     ; CONTADOR HIGH TEMPO
LOW
                CONT_INT                        EQU     12H ;
CONTADOR DE INTERRUPCAO
            START_BIT               EQU     13H ; START BIT
            CONT_BIT                EQU     14H ; CONTADOR DE BITS RX
            BIT_RX              EQU     16H
                BYTE_SAIDA              EQU         20H
            ; BYTE RECEBIDO VIA RF E ENVIADO VIA RS232
            CONTADOR            EQU     15H     ; CONTADOR BYTES
RECEBIDOS

                ORG 0H
                LJMP        MAIN

                ORG 03H
            LJMP        RECEBE  ;RETI

                ORG 0BH
            LJMP INTERRUPCAO    ; TIMER

                ORG 13H
            RETI ;LJMP        TRAT_IN

                ORG 23H
            ;CLR ES
            ;CLR TI
            RETI

            ORG 100H

       MAIN:

            ;*******************************
            ; REGISTROS DO MICROCONTROLADOR


                CLR  EA             ; INIBE INTERRUPCOES
            MOV         SP,#55H         ; INICIALIZA STACK
                LCALL       INICIALIZA
                MOV                 CONT_INT,#0 ; ZERA CONTADOR DE
INTERRUPCAO
                MOV                 START_BIT,#0            ; ZERA
START BIT
            MOV                     CONT_BIT,#0 ; ZERA CONTADOR DE
BITS RX
            MOV         CONTADOR,#0
            MOV         R0,#30H
            MOV         R1,#30H
            MOV         A,#'G'
            LCALL       TX_1
            MOV         A,#'A'
            LCALL       TX_1
            MOV         A,#'F'
            LCALL       TX_1
            MOV         A,#'A'
            LCALL       TX_1
            MOV         A,#0DH
            LCALL       TX_1
            MOV         A,#0AH
            LCALL       TX_1
            SETB        EA

 VOLTA_2:
            MOV         A,CONTADOR
            JZ          VOLTA_2
            ;JMP         VOLTA_2

 VOLTA_1:
            MOV         A,@R1   ; LE DADO DA MEMORIA
            LCALL       TX_1
            DEC         CONTADOR
            INC         R1
            MOV         A,R1
            CJNE        A,#4FH,VOLTA_22
            MOV         R1,#30H
            JMP         VOLTA_2

 VOLTA_22:  MOV         A,CONTADOR
            JNZ         VOLTA_1
```

```
            MOV         R1,#30H
            MOV         R0,#30H

            JMP         VOLTA_2

          ;**************************************
              ; TRATAMENTO DE INTERRUPCAO, DADO RX VIA RF
          ; RECEBE E SALVA O TAMANHO DOS PULSOS
          ; RECEBIDOS EM NIVEL LOGICO LOW
              ; TEMPO TOTAL  => 546H
              ; TEMPO BYTE 0 => 168H
              ; TEMPO BYTE 1 => 348H


    RECEBE:
          CPL LED2

          ; SALVA TEMPO TOTAL ENTRE INTERRUPCAO

          PUSH          ACC
          PUSH          PSW
          MOV           A,TH0
          MOV                       MSB_H,A                 ; SALVAR
CONTADOR MSB TEMPO HIGH

          MOV     TH0,#00H
          MOV     TL0,#00H          ;  CLOCK

          JNB           RX,$        ; ESPERA INTERRUPCAO SUBIR


          MOV           A,TH0                   ; SALVAR CONTADOR MSB
TEMPO LOW
          MOV           MSB_L,A


              ; VERIFICA TEMPO TOTAL DA INTERRUPCAO

          MOV                       A,MSB_H                 ;
VERIFICA TEMPO TOTAL
              CJNE                  A,#05H,RECE_A           ; TEMPO
TOTAL MSB= 05, NAO RET E LIMPA

          ; VERIFICA SE BITE "0"

          MOV                       A,MSB_L
          CJNE          A,#01H,RECE_B11  ; NAO E BIT O, VERIFICA BIT
=1
          SETB          C                ; SETA CARRY COM "1"
          MOV           08H,C
          CALL          TRATA_BIT
          POP           PSW
          POP           ACC
          RETI

           ; VERIFICA SE E BIT "1"
 RECE_B11:
          CJNE          A,#02H,RECE_B1  ; NAO E BIT O, VERIFICA BIT
=1
          SETB          C               ; SETA CARRY COM "1"
          MOV           08H,C
          CALL          TRATA_BIT
          POP           PSW
          POP           ACC
          RETI

   RECE_B1:

          CJNE                      A,#03H,RECE_FIM         ; NAO E
BIT 1,
          CLR           C               ; LIMPA CARRY
          MOV           08H,C
          CALL          TRATA_BIT
          POP           PSW
          POP           ACC
          RETI

          ; LIMPA FLAGS DA INTERRUPCAO E RETORNA

   RECE_A:
                MOV                 START_BIT,#0            ; LIMPA
START BIT
                MOV                 CONT_INT,#0 ; LIMPA CONTADOR DE
ITERRUPCAO
                MOV                 CONT_BIT,#0 ; LIMPA CONTADOR DE
BITS
 RECE_FIM:
          POP           PSW
          POP           ACC
          RETI


           ;########################################
           ; TRATA O BYTE RECEBIDO E MONTA A PALAVRA
           ; PARA SER ENVIADA PARA A SAIDA SERIAL

  TRATA_BIT:

           MOV          A,START_BIT
           CJNE         A,#0AAH,TRATA_D0        ; VERIFICA SE TEM
START BIT
```

```
            JMP         BIT_0

  TRATA_D0:

            ; VERIFICA SE BIT HE 0 E SETA START BIT

            MOV         C,08H
            JC          TRATA_F           ; BIT 0? SE NAO SAI
            MOV         START_BIT,#0AAH   ; SIM SETA START BIT

 TRATA_F:    RET

            ; MONTA BYTE 20H

     BIT_0:
            MOV         A,CONT_BIT              ; SIM, MONTA BIT EM
20H
            CJNE        A,#0,BIT_1              ; SE CONTADOR = 0
LOAD D0 DADO RX
            MOV         C,08H
            MOV         00H,C                   ; SETA D0 DE 20H
            INC         CONT_BIT
            RET

   BIT_1:
            CJNE        A,#1,BIT_2              ; SE CONTADOR = 0
LOAD D0 DADO RX
            MOV         C,08H
            MOV         01H,C                   ; SETA D1 DE 20H
            INC         CONT_BIT
            RET

   BIT_2:
            CJNE        A,#2,BIT_3              ; SE CONTADOR = 0
LOAD D0 DADO RX
            MOV         C,08H
            MOV         02H,C                   ; SETA D2 DE 20H
            INC         CONT_BIT
            RET

   BIT_3:
            CJNE        A,#3,BIT_4              ; SE CONTADOR = 0
LOAD D0 DADO RX
            MOV         C,08H
            MOV         03H,C                   ; SETA D3 DE 20H
            INC         CONT_BIT
            RET

   BIT_4:
            CJNE        A,#4,BIT_5              ; SE CONTADOR = 0
LOAD D0 DADO RX
            MOV         C,08H
            MOV         04H,C                   ; SETA D4 DE 20H
            INC         CONT_BIT
            RET

   BIT_5:
            CJNE        A,#5,BIT_6              ; SE CONTADOR = 0
LOAD D0 DADO RX
            MOV         C,08H
            MOV         05H,C                   ; SETA D5 DE 20H
            INC         CONT_BIT
            RET

   BIT_6:
            CJNE        A,#6,BIT_7              ; SE CONTADOR = 0
LOAD D0 DADO RX
            MOV         C,08H
            MOV         06H,C                   ; SETA D6 DE 20H
            INC         CONT_BIT
            RET

   BIT_7:
            CJNE        A,#7,BIT_9              ; SE CONTADOR = 0
LOAD D0 DADO RX
            MOV         C,08H
            MOV         07H,C                   ; SETA D7 DE 20H
            INC         CONT_BIT
            RET
   BIT_9:
            CJNE        A,#8,BIT_8              ; SE CONTADOR = 0
LOAD D0 DADO RX
            MOV         C,08H
            JNC         XXX

            MOV          START_BIT,#0    ; LIMPA START BIT
            MOV          CONT_INT,#0     ; LIMPA CONTADOR DE
ITERRUPCAO
            MOV          CONT_BIT,#0     ; LIMPA CONTADOR DE BITS

            ; ENVIA BYTE CONVERTIDO PARA PORTA SERIAL


            MOV         A,20H

  ;     TRANSFERE DADO PARA BUFFER DE SAIDA

            MOV         @R0,A           ; TRANSFERE DADO PARA MEMORIA
            INC         CONTADOR        ; CONTADOR +1
            INC         R0
            MOV         A,R0            ;
            CJNE        A,#4FH,BIT_8    ; VERIFICA SE MEMORIA CHEGOU
AO FIM
            MOV         R0,#30H

            ;LCALL       TX_1
```

```
   BIT_8:

            RET

      XXX:
            MOV          START_BIT,#0    ; LIMPA START BIT
            MOV          CONT_INT,#0     ; LIMPA CONTADOR DE
ITERRUPCAO
            MOV          CONT_BIT,#0     ; LIMPA CONTADOR DE BITS
            RET


               ; ******************************
            ; * ROTINA DE INICIALIZACAO    *
               ; ******************************

               INICIALIZA:

               MOV     TMOD,#0A1H       ; CONT 0 16 BITS, CONT 1
C/RELOAD
                       MOV     SCON,#11011000B ; SERIAL MODO 3,RX
HAB,SM20,TB8=1,TI/RI=0
               MOV     TH1,#0FDH       ; 9600 BPS 11.0592MHz
               MOV     TH0,#00H
               MOV     TL0,#00H        ;  CLOCK
                       SETB    TR1             ; LIGA CONT.1
                       SETB    TF1             ;
                       CLR     ES              ;  "   CANAL SERIAL
               SETB    ET0             ; LIMPA INT CONT 0
               SETB    TR0             ; LIGA CONTADOR 0
               ;SETB    PX0              ; PRIORIDADE INT SERIAL
               SETB    EX0              ; HABILITA INT 0 EXT
               SETB    IT0              ; INTERRUPCAO NA DESCIDA ;WF

                       ;****SETB PS             ; PRIORIDADE INT.
SERIAL

                       MOV     PSW,#00H
               ;SETB    EA
                       RET


      TX_1:
                MOV    SBUF,A  ; TRANS. DATA
                CLR        TI
                JNB        TI,$
                CLR        TI
                RET

             ; *************************
                ; *  RECEPCAO  DE UM BYTE  *
                ; *  VIA CANAL SERIAL      *
                ; *************************

             RX_1:

                       JNB     RI,$
                       CLR     RI
                       MOV     A,SBUF          ; LE DADO
                       RET



               ;*************************
               ; TRATA INTERRUPCAO INTERNA
               ; DO TIMER 0 A CADA 10MS
               ;*************************

  INTERRUPCAO:
            MOV         TH0,#0H         ; contador 0
            MOV         TL0,#0H         ; contador 0
            CPL         LED
            RETI

       ;*****************************************
       ; CONVERTE UM BYTE HEXA EM DOIS ASC
       ; B CONTEM BYTE HEX E

    HEXASC:
              MOV       A,#0F0H
              ANL       A,B
              SWAP      A
              CALL      CVERT
              MOV       A,#0FH
              ANL       A,B
              CALL      CVERT
              ret
    CVERT:
              PUSH      ACC
              SUBB      A,#10
              JC        J1
              POP       ACC
              ADD       A,#37H
              lcall     TX_1
              RET

      J1:     POP       ACC
              ADD       A,#30H
              lcall     TX_1
              RET


  DB 'RONALDO 28/11/96'
              END
```

ANEXO III

Código fonte do microcontrolador 89C2051 para o circuito de transmissão.

```
                ; ESTA ROTINA RECEBE DADOS VIA RS-232 E REENVIA VIA RF
                ; INICIALMENTE OS DADOS RECEBIDOS SAO ARMAZENADOS NA
AREA
                ; DE MEMORIA 30H A 3FH UTILIZANDO ESTE BANCO ROTATIVO
                ; OU SEJA QDO CHEGAR NA ULTIMA POSICAO ENDERECO, OS
DADOS SERAO
                ; GRAVADOS NO INICIO. CADA BYTE RECEBIDO INCREMENTA O
CONTADOR
                ; DE CARACTERES RECEBIDOS.
                ; A TRANSMISSAO RF INICIA QDO O CONTADOR  > 0,
ENVIANDO O
                ; PRIMEIRO BYTE ALOCADO EM 30H.
                ; SERAO UTLIZADOS DOIS PONTEIROS SEPARADOS PARA
ARMAZENAR E ENVIAR
                ; OS DADOS.
                ; O PONTEIRO "ENTRADA" => "R0"
                ; O PONTEIRO "SAIDA"   => "R1"
                ; O CONTADOR "CONTADOR" => 20H
            ; OS BITS SERAO ENVIADOS NO FORMAT PWM COM TOTAL DE 542H
PULSOS
                ;          ________
                ; |____|         |__  BIT "0" O TEMPO TOTAL EM NIVEL 0
= 360H
                ;          _____
                ; |________|    |__  BIT "1" O TEMPO TOTAL EM NIVEL 0
= 1F0 A 220H
                ;
                ; CADA BYTE E PRECEDIDO POR 3 BITS DE SINCRONISMO =1
                ; 1 START BIT =0  E OITO BITS DE DADOS
                ;
                ; | S | S | S | ST | D0 | D1 | D2 | D3 | D4 | D5 | D6
| D7 |
                ; 09/05/0000 AUMENTADO O BUFFER DE RECEPCAO DE
CARACTERES.
                ; 09/05/0000 MODIFICADO STACK PARA 58H
                ; ACRESCENTA MAIS DOIS BITS DA SAIDA DE DADOS
                ; 18/05 MODIFICADO O BUFFER DE ENTRADA DE DADOS


$MOD51
$TITLE(BYTE SIGNED MULTIPLY)
$PAGEWIDTH(132)
$DEBUG
$OBJECT
$NOPAGING



                LED                 EQU     P1.1
                RF                  EQU     P1.0
                INTE                EQU     10H
                DELAY               EQU     11H
                CONTADOR            EQU     12H
                SEGUNDO             EQU     13H
                ENTRADA             EQU     30H
                SAIDA               EQU     30H

                ORG 0H
                LJMP        MAIN

                ORG 03H
                RETI

                ORG 0BH
                LJMP INTERRUPCAO    ; TIMER

                ORG 13H
                RETI ;LJMP        TRAT_IN

                ORG 23H

                ;CLR ES
                ;RETI
                LJMP        RX_DADO

           ORG 100H


       MAIN:

                CLR  EA             ; INIBE INTERRUPCOES
                MOV         SP,#58H         ; INICIALIZA STACK
                LCALL       INICIALIZA

                MOV         R1,#ENTRADA     ; CARREGA PONTEIRO
INICIAL DE ENTRADA
                MOV         R0,#SAIDA        ; CARREGA PONTEIRO SAIDA
DE DADOS
                MOV         CONTADOR,#0
                MOV         SEGUNDO,#0

                ; VERIFICA SE TEM DADO PARA SER ENVIADO VIA RF

  VOLTA_1:  MOV         A,CONTADOR     ; VERIFICA SE TEM DADO PARA TX
RF
                JZ          VOLTA_1


  TRATA_RF:
                MOV         A,@R1
                MOV         B,A
                INC         R1       ; APONTA PROXIMO DADO
                MOV         A,R1
                CJNE        A,#4FH,TT
                MOV         R1,#30H
            TT:
                DEC         CONTADOR
                MOV         R3,#8   ;NUMERO DE BITS PARA RODAR
                SETB        C
                CALL        BIT_1   ; ENVIA SICRONISMO
                SETB        C
                CALL        BIT_1   ; ENVIA SICRONISMO
                SETB        C
                CALL        BIT_1   ; ENVIA SICRONISMO
                SETB        C
                CALL        BIT_1   ; ENVIA SICRONISMO
                SETB        C
                CALL        BIT_1   ; ENVIA SICRONISMO
                SETB        C
                CALL        BIT_1   ; ENVIA SICRONISMO
                CLR         C
                CALL        BIT_1   ;   ENVIA STAR_BIT

     T_RF:
                MOV         A,B
                RRC         A       ; ENVIA BIT PARA CARRY
                MOV         B,A
                CALL        BIT_1
                DJNZ        R3,T_RF
                SETB        C
                CALL        BIT_1   ; ENVIA SICRONISMO
                SETB        C
                CALL        BIT_1   ; ENVIA SICRONISMO

                MOV         A,CONTADOR

            ;incluir rotina de delay

                JNZ         TRATA_RF

                MOV         R1,#30H
                MOV         R0,#30H

                JMP         VOLTA_1


                ; ###################################
                ; ENVIA OS TEMPO EQUIVALENTE EM "0"
                ; E EM "1"
                ; INICIALMENTE VERIFICA SE CARRY = 0
                ; CONTA ATE 20 INT COM SAIDA EM ZERO
                ; SE CARRY = 1, CONTA ATE 10 INTERRUPCOES
                ; EM SAIDA A ZERO

      BIT_1:
                JNC         BIT_0           ; SE BIT = 0 ENVIA

                ; ENVIA TEMPO DE NIVEL 1

                CLR         RF              ; NIVEL BAIXO EM RF
                MOV         DELAY,#0        ; LIMPA CONTADORM DE
INTERRUPCAO
                MOV         A,#10
                CJNE        A,DELAY,$       ; ESPERA NIVEL ALTO EM RF
                SETB        RF

    V1:
                MOV         A,#30
                CJNE        A,DELAY,$
                CLR         RF
                RET

      BIT_0:

                ; ENVIA TEMPO DE NIVEL 0

                CLR         RF              ; NIVEL BAIXO EM RF
                MOV         DELAY,#0        ; LIMPA CONTADORM DE
INTERRUPCAO
                MOV         A,#20
                CJNE        A,DELAY,$       ; ESPERA NIVEL ALTO EM RF
                SETB        RF
                JMP         V1

                ; ******************************
                ; * ROTINA DE INICIALIZACAO    *
                ; ******************************

                INICIALIZA:

                MOV     TMOD,#021H       ; CONT 0 16 BITS, CONT 1
C/RELOAD
                        MOV     SCON,#11011000B ; SERIAL MODO 3,RX
HAB,SM20,TB8=1,TI/RI=0
                MOV     TH1,#0FDH       ; 9600 BPS 11.0592MHz
                MOV     TH0,#0FFH
                MOV     TL0,#0E3H       ;  CLOCK
                        SETB    TR1             ; LIGA CONT.1
                        SETB    TF1             ;
                        SETB    ES              ;  "   CANAL SERIAL
                        SETB    ET0
                        SETB    TR0

                        SETB    PS              ; PRIORIDADE INT.
SERIAL
```

```
                    MOV     PSW,#00H
                    SETB    EA
                    MOV     DELAY,#0
                    RET


     TX_1:
               MOV    SBUF,A  ; TRANS. DATA
               CLR    TI
               JNB    TI,$
               CLR    TI
               RET


              ;***************************
              ; TRATA INTERRUPCAO INTERNA
              ; DO TIMER 0 A CADA 10MS
              ;***************************
INTERRUPCAO:

              PUSH        PSW
              PUSH        ACC
              MOV         TH0,#0FFH          ; contador 0
          MOV         TL0,#0E0H; #0E3H          ; contador 0
              INC         DELAY
              INC         SEGUNDO
              CLR         A
              CJNE        A,SEGUNDO,INTE_01
              INC         R5
              CPL         LED
INTE_01:
              POP         ACC
              POP         PSW
              RETI
```

```
                    ;*********************************
                    ; TRATAMENTO DE INTERRUPCAO SERIAL
                    ; R1 DADO LIDO CANAL SERIAL
                    ;*********************************

       RX_DADO: ;CLR     EA              ; INIBE INT SERIAL
                    PUSH    ACC             ; SALVA ACUMULADOR
                    PUSH    PSW             ;   "   FLAG

                    JNB     RI,$
                    CLR     RI
                    MOV     A,SBUF          ; LE DADO

                    MOV     @R0,A           ; ENVIA DADO P/
MEMORIA
                    INC     R0              ; PROXIMA POSICAO
                    INC     CONTADOR        ; CONTADOR DE BYTES
+1
                    MOV     A,R0
                    CJNE    A,#4FH,SAI      ; VERIFICA SE FIM
MEMORIA
                    MOV     R0,#30H

              SAI:
                    POP     PSW              ; RESTAURA FLAG
                    POP     ACC              ;    "
ACUMULADOR
                    CLR     RI
                    RETI



   DB 'RONALDO 28/11/96'
              END
```