UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO

(Bacharelado)

PROTÓTIPO DE SOFTWARE PARA O INTERFACEAMENTO E AQUISIÇÃO DE DADOS DE UMA BALANÇA ATRAVÉS DA RS-232

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

MARCIO MARCOS MACHADO JUNIOR

BLUMENAU, JUNHO/1999

1999/1-99

PROTÓTIPO DE SOFTWARE PARA O INTERFACEAMENTO E AQUISIÇÃO DE DADOS DE UMA BALANÇA ATRAVÉS DA RS-232

MARCIO MARCOS MACHADO JUNIOR

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. Sérgio Stringari — Orientador na FURB

_____

Prof. José Roque Voltolini da Silva — Coordenador do TCC

BANCA EXAMINADORA

_____

Prof. Sérgio Stringari

_____

Prof. Antônio Carlos Tavares

_____

Prof. Miguel Wisintainer

DEDICATÓRIA

À meus pais e avós, os quais me motivaram e apoiaram muito em minha vida.

AGRADECIMENTOS

Ao professor Sérgio Stringari pela atenção prestada durante a execução do trabalho.

SUMÁRIO

LISTA DE ABREVIATURAS

UART – Universal Asynchronous Receiver Transmitter

THR – Transmitter Holding Register

RDR – Receiver Data Register

IER – Interrupt Enable Register

IIR – Interrupt Identifier Register

LCR – Line Control Register

MCR – Modem Control Register

LSR – Line Status Register

MSR – Modem Status Register

DTR – Data Terminal Ready

RTS – Request to Send

DSR – Clear To Send

DCD – Data Carrier Detected

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

A interface padrão RS-232 é uma grande ferramenta para a integração entre dispositivos independentes. Com o propósito de criar uma interface que adquira dados adquiridos de uma balança através de uma RS-232 e enviar a saída à uma impressora de código de barras através de uma outra RS-232, foram levantadas e estudadas técnicas de interface com RS-232 assim como os protocolos de comunicação envolvidos no processo. Em função disto foi desenvolvido um protótipo de um software para o interfaceamento de uma balança e uma impressora através de suas interfaces RS-232.

ABSTRACT

The standard interface RS-232 is a great tool for the integration between independent devices. With the objective to interface data acquired from a balance through one RS-232 and send an output to a bar code printer through another RS-232, were colected and studied interfacing technics and the comunication protocols involved in the process. With all these was developed a prototype of a software that interfaces one balance and one bar code printer througth two RS-232.

1 INTRODUÇÃO

O processo de uma célula de pesagem de componentes/peças em uma empresa, consiste em alguns passos, tais como, pesar uma amostra contada; pesar a quantidade total, obter o número total de peças do lote; dar entrada no sistema de controle; desenhar e gerar uma etiqueta de código de barras para a identificação do lote. Esta operação, quando executada manualmente é passível de muitos apontamentos errados, indo de encontro aos objetivos da empresa que se utiliza deste processo.

Tendo em vista este problema, considera-se que a automatização deste processo garantiria uma maior segurança dos dados obtidos. Para isto seria necessário desenvolver um software que opere como interface entre a balança, o sistema que administra os dados e a impressora. É a informatização deste processo que este TCC visa alcançar. Para isto foi necessário estudar e compreender o protocolo de comunicação da balança, recebido via uma interface serial padrão RS-232, e o protocolo da impressora, recebido também através de uma interface padrão RS-232, para o processamento da informação, formulação e impressão de uma etiqueta de código de barras.

Para a especificação e implementação do processo proposto foi necessário estudar e entender a forma que estes dados se comunicam através da interface RS-232 e interpretá-los corretamente, ou seja, estudar o interfaceamento da RS-232 e o protocolo proprietário da balança para posterior tratamento das informações.

Posteriormente, foi especificado e implementado o protocolo da impressora de código de barras, para a geração de etiquetas identificadoras do lote, pelo processo proposto .

1.1 OBJETIVOS

O objetivo principal do trabalho proposto é especificar e implementar um protótipo de software que faça o interfaceamento entre uma balança e uma impressora de código de barras através da porta RS-232, com os respectivos controles das informações:

O objetivo secundário do trabalho desenvolvido foi informatizar/automatizar o controle de componentes (peças) para um estoque de peças em uma empresa.

1.2 ORGANIZAÇÃO TEXTO

No capítulo 1 é apresentada uma introdução ao trabalho desenvolvido, os objetivos e a organização do mesmo.

No capítulo 2 são apresentados fundamentos sobre comunicação de dados assim como alguns métodos para o interfaceamento com uma RS-232.

No capítulo 3 são apresentadas as tecnologias envolvidas no desenvolvimento do protótipo.

No capítulo 4 são descritos o desenvolvimento, a especificação e a implementação do protótipo.

O capítulo 5 relata as conclusões, assim como a sugestão de continuidade do protótipo.

No capítulo 6 são apresentadas as referências bibliográficas citadas no trabalho.

2 COMUNICAÇÃO DE DADOS

O “ato” de comunicação, não necessita de computadores. Duas pessoas conversando, utilizam o mesmo processo. O processo de uma conversa envolve uma série de processos que para as pessoas passam como despercebidos. Enquanto uma pessoa fala a outra escuta, e vice-versa. Com isto temos a alternância entre transmissor e receptor. Utilizam-se de um canal comum a eles, e ambos trocam mensagens entre si, utilizando o mesmo canal. As duas pessoas seguem regras induzidas pelo comportamento, afim de tornar a conversação ordenada e segura. Por exemplo, não é possível que uma pessoa fale com outra ao mesmo tempo que tenta ouvi-la.

A comunicação é formada de meios e regras, pelos quais, a mensagem, o elemento causador da comunicação seja o item principal. Uma mensagem só é válida se a mensagem chegar ao seu destino com a mesma qualidade que sai de sua origem. Os aspectos que devem ser observados em uma comunicação são:

a) fonte da transmissão;
b) mensagem a ser recebida;
c) canal ou meio de transmissão;
d) destino da informação a ser transmitida.

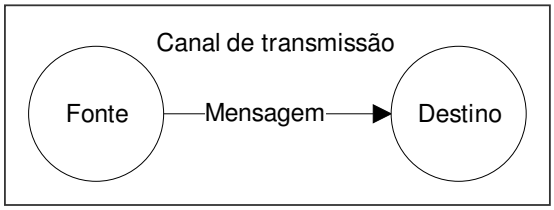


Figura 2-1 – Elementos básicos da comunicação

2.1 FORMAÇÃO DE CÓDIGO

Samuel Morse inventou o código *Morse*, um sistema binário de tons largos e curtos associados a cada letra do alfabeto, para comunicação a longa distância por linhas telegráficas ou rádio.

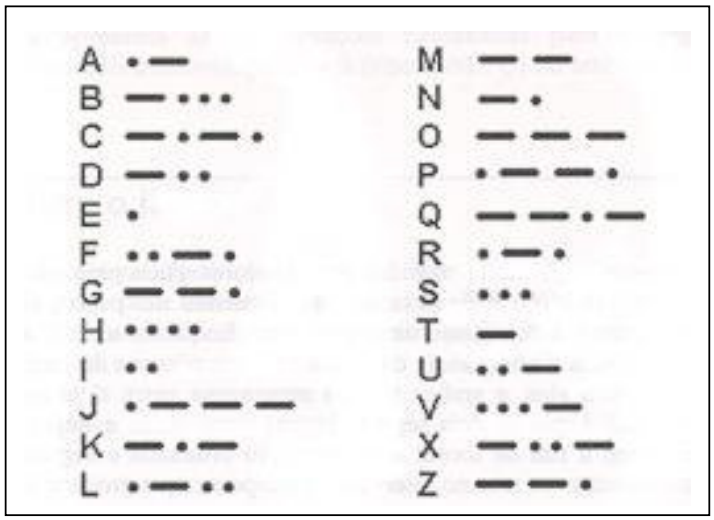


Figura 2-2 – Codificação do Código Morse

Como descobriu Samuel Morse, a aplicação do código não apenas facilita, mais é fundamental para uma comunicação confiável e segura. Este processo, também aplicado à computadores, recebeu o nome de processo digital. Os números, as letras, os caracteres e os símbolos especiais são representados, em processamento de dados, pela condição (ligado/ desligado), ou seja, uma base numérica 2 ou binária.

Base 2 ➔ 2 estados (0 e 1)

Base- conjunto de elementos básicos para a formação de valores.

Bit- binary digit- menor unidade de informação

A essência dos códigos em computadores é sempre base 2, então, pode-se afirmar que a quantidade de bits utilizados determinará o número de combinações possíveis. A representação matemática para esta fórmula é:

$2^n = Q$

Onde:

-n = ao número de bits utilizados

-Q = número de combinações possíveis

-2 = base numérica

Utilizando o sistema de bits para representar letras ou números, criou-se o conceito de *byte* que é um agrupamento de bits que define uma informação. Exemplo: um *byte* nos micro computadores geralmente é formado por oito bits. Então :

1 Byte = 8 bits

2^8=256 combinações possíveis

No código *ASCII* (*Americam Standard Code for Information Interchange*), dentro das 128 primeiras possibilidades, é possível englobar todos os dígitos decimais, letras maiúsculas, minúsculas e os símbolos especiais.

No código ASCII a letra maiúscula “E” é representada pela seqüência: “01000101”.

2.2 TRANSFERÊNCIA DE DADOS

As transferências de dados podem ser realizadas de duas formas:

a) transferência serial;
b) transferência paralela.

2.2.1 TRANSFERÊNCIA SERIAL

Os bits que representam uma informação, são transmitidos seqüencialmente, um a um, por um único suporte físico (figura 2-3). Esta é a forma mais utilizada na transmissão de dados entre computadores.

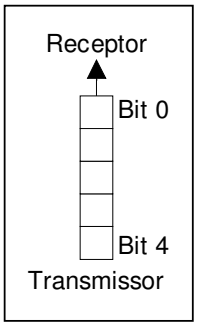


Figura 2-3 – Transferência serial

2.2.2 TRANSFERÊNCIA PARALELA

Os bits que representam uma informação são transmitidos simultaneamente, através de diversos suportes físicos em paralelo (figura 2-4). Esta forma de transferência é utilizada para curtas distâncias entre os computadores ou, ainda operações internas no computador e periféricos.

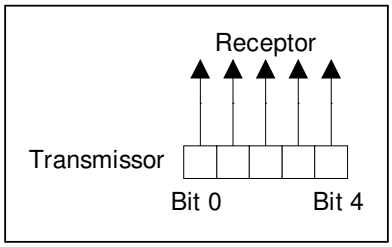


Figura 2-4 – Transferência paralela

2.2.3 SENTIDO DA TRANSMISSÃO

Na comunicação de dados, considerando a conexão entre dois computadores, a transferência dos dados é classificada de acordo com o sentido da transmissão. O sentido da transmissão especificado particularmente para cada conexão, determina claramente a figura de transmissor (emissor) e do receptor. Em relação ao sistema da transmissão, a comunicação pode ser:

Simplex: segundo [TAF96], o sentido simplex é uma transmissão de dados em um único sentido, como é ilustrado na figura 2-5. Este modo é pouco utilizado, pois impossibilita a resposta do receptor, confirmando a chegada da informação.

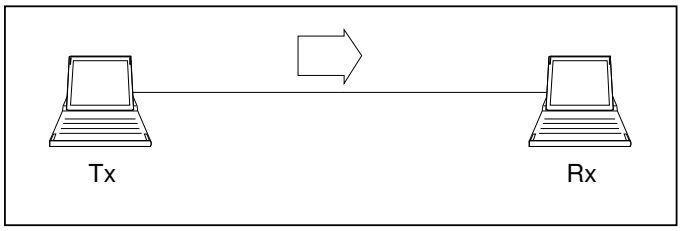


Figura 2-5 – Transmissão simplex

Half-duplex: segundo [TAF96] o sentido half-duplex é uma transmissão de dados em ambos os sentidos, porém não simultâneas, ilustrado da figura 2-6.

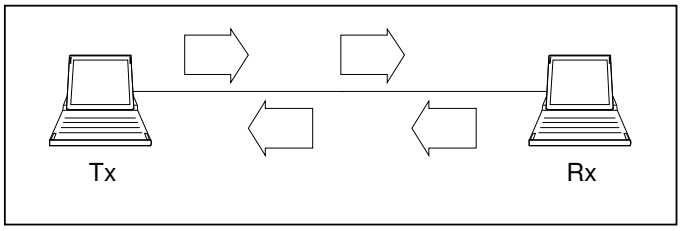


Figura 2-6 – Transmissão half-duplex

Full-duplex: segundo [TAF96] o sentido full-duplex é uma transmissão de dados em ambos os sentidos, simultaneamente, como é ilustrado na figura 2-7.

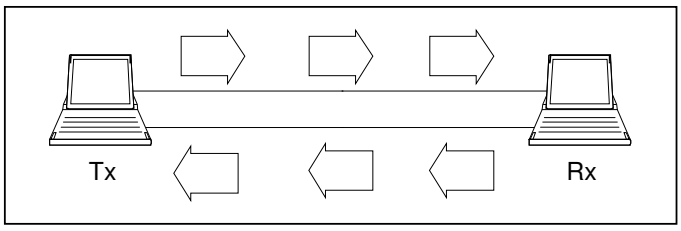


Figura 2-7 – Transmissão full-duplex

2.3 MODOS DE TRANSMISSÃO

Existem dois modos de transmissão, assíncrona e síncrona.

Assíncrona:

Neste modo, cada caracter recebe bits adicionais que indicam o início e fim dos mesmos, não necessitando uma pré-fixação de cadência entre o transmissor e o receptor. A principal característica deste modo é de poder ser feito a qualquer hora, sem limite de tamanho de mensagem.

Síncrona:

Este processo é possível pois se baseia no estabelecimento de uma cadência (clock) fixa para a transmissão dos bits. Esta cadência deve ser compatibilizada pelo transmissor e pelo receptor. Sua principal característica é a possibilidade de transmitir blocos de dados com a adição de controles apenas no começo e no fim de cada bloco.

Velocidade da transmissão

A velocidade significa "a quantidade de informações transferidas do transmissor para o receptor em um determinado intervalo de tempo". Estas grandezas são geralmente expressas em bps e baud.

- bps – bits por segundo
- baud – velocidade de sinalização, é a velocidade do sinal em um meio de transmissão.

Relacionando as mediadas podemos dizer que:

Se cada sinal representa uma informação 1 bps=1 baud.

2.4 ERROS NA TRANSMISSÃO

Os erros de transmissão ocorrem devido a interferências causadas pelo ambiente externo sobre o canal de comunicação.

Para precaver alguma deformação da mensagem, foram desenvolvidas técnicas para a detecção de erros como VRC, CRC e Hamming, descritos abaixo:

- VRC – Vertical Redundance Check – consistem em somar o número de bits 1 de um caracter que totalizará uma número par ou ímpar. Desta forma pode-se optar pela paridade par ou ímpar;

- CRC – Ciclical Redundance Check – procedimento que emprega cálculos complexos para gerar um número baseado nos dados transmitidos e transmite ao receptor ao resultado. O receptor por sua vez também efetua os calculos sobre os dados recebidos e compara com o resultado enviado pelo transmissor;
- Hamming – este método acrescenta 3 bits de verificação no final de cada 4 bits de dados que, quando recalculados pelo receptor são utilizados para verificar a presença de erros e, alguns casos, até podem ser utilizados para corrigir a mensagem.

2.5 A INTERFACE RS-232

Segundo [TAN94], a interface entre um computador ou um terminal e um modem é um exemplo de um protocolo da camada física, pois ele deve especificar em detalhe a interface mecânica, elétrica, funcional e procedimental. A interface RS-232, foi elaborada pela EIA (Eletronic Industries Association), e estabelece um padrão para o interfaceamento serial com um computador. Os sinais trocados entre dois equipamentos são normalizados pela CCITT através da recomendação V24.

Estes sinais são especificados por um conector de 25 pinos tipo “D”. A conexão mecânica é padronizada pela ISO (International Standart Organization), através de um conector padrão denominado “DB-25P”

2.5.1 DESCRIÇÃO DOS SINAIS DA INTERFACE

Conforme [TAF96] dos 25 pinos especificados no pradrão RS-232C, os sinais mais utilizados, estão descritos na tabela abaixo na Tabela 2-1.

Pino conector	Circuito (CCITT)	Denominação CCITT-V.24	Origem	Função dos sinais
01	101	Protective ground (PG)		Terra de proteção
02	103	Transmited data (TXD)	Terminal	Este é o fio por onde os dados são transmitidos.
03	104	Received data (RXD)	Modem	Este é o fio por onde os dados são recebidos.
04	105	Request to send (RST)	Terminal	Este é o fio pelo qual o terminal faz uma solicitação para transmitir.
05	106	Clear to send (CTS)	Modem	O terminal faz solicitação para transmitir.
06	107	Data set ready (DSR)	Modem	Resposta de um CTS do terminal.
07	102	Signal grund		Sinal base de zero volts.
08	109	Data carrier detected (DCD) ou Received line signal detector (RLSD)	Modem	Sinal de resposta informando a recepção de algum dado.
15	114	Transmiter clock (TXCLK)	Modem	Clock de transmissão.
17	115	Data derived clock (RXCLK)	Modem	Clock de recepção.
20	108	Data terminal ready (DTR)	Terminal	Sinal do terminal que indica que está pronto para comunicação.
22		Ring indicator	Modem	Sinal de um modem indicando o sinal de tom.
24	113	External tranmitter clock (EXT TXCLK)	Terminal	Clock externo para transmissão.

Tab2-1 – Descrição dos sinais da interface

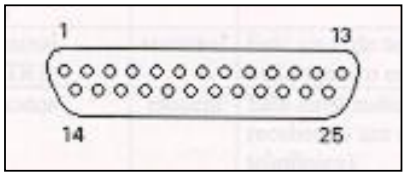


Fig2-8 – Conector padrão D-25P fêmea

2.5.2 CARACTERÍSTICAS ELÉTRICAS

Segundo [TAN94] as questões de projeto são concernentes à garantia de que, quando um lado transmite um bit 1, este seja recebido como um bit 1 do outro lado. As questões típicas são quantos *volts* devem ser utilizados para representar 1 e 0.

A interface padrão RS-232 define, atualmente, quatro níveis lógicos. A entradas têm definições diferentes das saídas e o controles têm definições diferentes dos dados. Para as saídas em condição de marca (*mark*), ou estado "1", quando a tensão no circuito de transferência, medida no ponto da interface é menor de –5V e maior que –15V, com relação ao circuito *signal ground* (terra). O sinal é considerado na condição de espaço (*space*), ou

estado “0”, quando a tensão for maior que 5V e menor que 15V, também com relação ao circuito terra. A região compreendida entre –5V e +5V é definida como região de transição.

Figura 2-9 – Saída do sinal

Para entradas o sinal é considerado em condição de marca ou estado “1”, quando a tensão do circuito de transferência, medida no ponto de interface, é menor que –3V e maior que –15V, com relação ao circuito terra. O sinal é considerado na condição de espaço ou estado “0”, quando a tensão for maior que +3V e menor que mais +15V, também com relação ao circuito terra. A região compreendida entre –3V e +3V –e definida como região de transição.

Figura 2-10 – Entrada do sinal

Durante a transição de dados a condição de marca é usada para descriminar o estado binário “1” e em condição de espaço é usada para descriminar o estado “0”.

2.6 COMUNICANDO COM UM IBM-PC

A linha de computadores IBM-PC, produzidos inicialmente pela empresa IBM utilizava na arquitetura da máquina o chip 8250 (Western Digital), hoje em dia e utilizado o 16550, para a efetivação da comunicação. Este chip também chamado Universal Asynchronous Receiver Trasmitter) – UART, é usado para a conversão do protocolo serial.

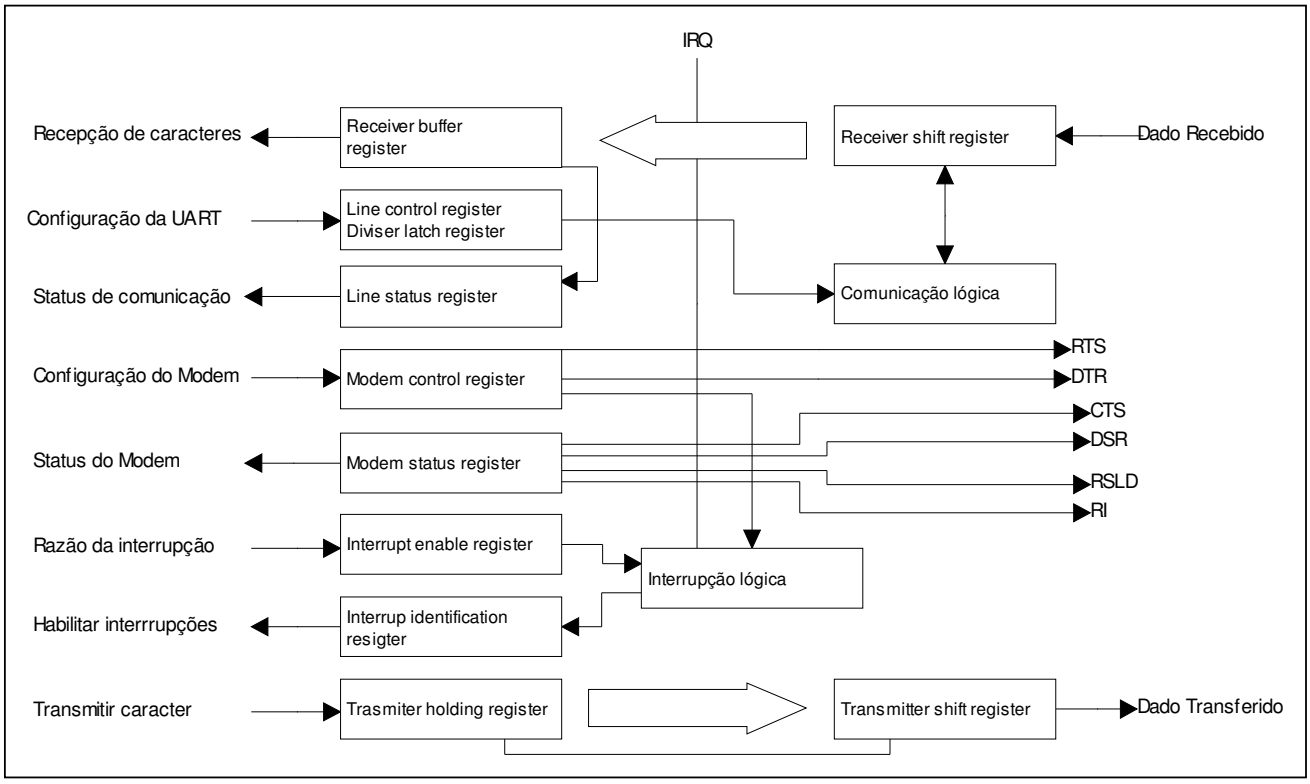


Figura 2-11 – Diagrama da UART

A UART poupa o programa a tarefa de realizar, através do processador, a transmissão e a recepção de caracteres, pelo canal serial. Para receber e enviar dados, o programa, simplesmente, lê e escreve dados na UART que se mostra aos "olhos" do processador, apenas como uma porta de E/S (endereço).

Este chip utiliza, a nível de bit, o modo de entrada/saída assíncrono serial para a comunicação. O principal conceito deste protocolo é que todos os dados e informações de controle, necessários para transmitir e receber um caracter de informação, precisam serem movidos sobre um linha simples, um bit por vez.

Quando estão sendo transmitidos dados sobre uma linha, o estado lógico da linha é 1, ou *marking state (mark)*. Quando é transmitido um caracter de dado, o primeiro bit a ser transmitido é o *start bit*. O *start bit* é representado pelo sinal lógico 0 da linha. O receptor sabe que está esperando um caracter de dado quando o estado da linha muda de *mark* para *mark*, de 1 para 0.

Os bits de dados, após a configuração prévia da UART, são transmitidos imediatamente após o *start bit*. O número de bits pode variar de 5 a 8 bits. Após os bit de dados, segue, opcionalmente o bit de paridade. Este bit deve ser constante durante toda a transmissão. Por exemplo: se a paridade escolhida for par, então o número de bits lógicos de dados precisa ser par até o final da transmissão.

Este bit de paridade pode permitir ao receptor detectar possíveis erros que venham ocorrer durante a transmissão dos dados.

Após o bit de paridade segue um 1, 1.5, ou 2 bits de *mark* (1 lógico). Estes bits são chamados de *stop bits*. O *stop bit* representa o tempo mínimo que a linha precisa para voltar para a condição de *mark* antes do próximo *mark* aparecer. O número de *stop bits* também precisa ser constante durante toda a transmissão.

2.6.1 UART (UNIVERSAL ASYNCHRONOUS TRANSMITTER RECEIVER)

A UART, para efetuar a comunicação, necessita ser configurado com dados sobre o número de bits de dados, tipo de paridade e número de *stop bits*. Uma vez inicializada, tudo que precisa-se fazer é passar um byte qualquer de dado para a porta de E/S e o caracter é automaticamente transmitido, serialmente, com todas as características apropriadas, enquanto a transmissão é realizada, a CPU pode atender qualquer outra tarefa, e periodicamente verificar se a UART esta disponível.

Para a transmissão de caracter para a UART, é necessário verificar se o registrador *Transmitter Holding Register* (THR) está vazio. Se estiver vazio então, ele está pronto para receber o próximo caracter para a transmissão. Quando a UART completar a transmissão do caracter, o conteúdo de THR é levado para o registrador *Transmitter Shift Register* (TSR) e o THR está pronto para receber outro caracter.

A UART adiciona adequadamente o *start bit*, a paridade e os *stop bits* e então transmite os bits na velocidade especificada.

O processo de recepção, da UART, recebe a informação dentro do registrador *Receiver Shift Register* (RSR). Ao receber o dado, e os erros serem devidamente checados o caracter é

movido para o registrador. *Receiver Data Register* (RDR). A UART atualiza o registrador de estado *Line Status Register* (LSR) e diz que o dado recebido está pronto.

2.6.1.1 PROGRAMANDO A UART

Para usar a UART corretamente, deve-se proceder de duas maneiras em tempos diferentes. Primeiro precisa-se configurá-la e, depois, apenas usá-la. Isso significa que, inicializada e não desejando mais modificar a configuração inicial, basta apenas operá-la até o fim da aplicação, sem mais se preocupar com a sua configuração. Entretanto, é importante lembrar que não é possível modificar a configuração da transmissão durante o processo de transmissão, pois, fatalmente, isto acarretaria erro na transmissão.

Para a inicialização, deve-se programar cinco dos nove registradores da UART, demonstrados na tabela 2-2. Uma vez inicializados, eles devem ser ignorados enquanto a UART estiver em uso. Não se modifica a configuração do canal serial, enquanto se sustenta uma conexão com outro terminal. As portas (endereços) reservadas da UART estão entre os endereços 3F8H e 3FEH para a porta serial 1, e 2F8H e 2FEH para a porta serial 2.

Porta Serial 1	Porta Serial 2	Entrada/Saída	Registrador	Abreviação
3F8H	2F8H	S	Transmitter Holding Register	THR
3F8H	2F8H	E	Receiver Data Register	RDR
3F8H*	2F8H*	S	Baud Rate Divisor	LSB
3F9H*	2F9H*	S	Baud Rate Divisor	MSB
3F9H	2F9H	E	Interrupt-Enable Register	IER
3FAH	2FAH	E	Int.-Identification Register	IIR
3FBH	2FBH	S	Line-Control Register	LCR
3FCH	2FCH	S	Modem-Control Register	MCR
3FDH	2FDH	E	Line-Status Register	LSR
3FEH	2FEH	E	Modem-Status Register	MSR

Tabela 2-2 – Registradores da UART

* Quando o bit 7 do *Line-Control Register* – LCR – estiver com seu estado lógico igual a 1, significa configuração de velocidade.

Os cinco registradores utilizados para a inicialização estão relacionados na tabela 2-3.

Registradores	Abreviação
Baud Rate Divisor	LSB
Baud Rate Divisor	MSB
Line Control Register	LCR
Modem Control Register	MCR
Interrupt Enable Register	IER

Tabela 2-3 – Registradores utilizados para inicialização da UART

O primeiro parâmetro que deve ser inicializado é o registrador *Baud Rate Divisor*. Este valor é usado para dividir a alta-frequência do clock em sinais de clock que representem a taxa a ser utilizada na transmissão e recepção.

Para inicializar o divisor de taxa de velocidade, precisa-se, primeiramente, mudar o estado lógico do bit 7 do *Line Control Register* (LCR), para 1. Fazendo isto estamos habilitando os endereços 3F8H e 3F9H para receber as taxas de velocidades, uma vez que o *bit* 7 deste registrador (LCR) está com o valor devido. Dessa forma, pode-se carregar LSB e MSB com os valores da velocidade desejada e passá-los para as portas 3F8H e 3F9H.

Velocidade	Valor para os registradores	
	MSB	LSB
50	09H	00H
110	06H	00H
150	03H	00H
300	01H	80H
600	00H	C0H
1200	00H	60H
2400	00H	30H
4800	00H	18H
9600	00H	0CH
19200	00H	06H
38400	00H	03H
56000	00H	02H

Tabela 2-4 – Valores de MSB e LSB para a configuração da velocidade de transmissão

Normalmente, os três bits da parte alta do *byte* (7,6,5) são deixados em zero. A extensão de paridade pode ser 1 se quisermos que a paridade seja um valor constante. Por exemplo, pretendendo-se transmitir caracteres com 7 bits de dado, um *stop bit*, e paridade impar, envia-se para o endereço 3FBH, cuja funcionalidade dos bits estão relacionados na tabela 2-5, o valor 1AH (00011010 binário).

Bits	Nome	Valor	Função
0-1	Tamanho do caracter	00 01 10 11	Tamanho do caracter é 5 bits. Tamanho do caracter é 6 bits. Tamanho do caracter é 7 bits. Tamanho do caracter é 8 bits.
2	Tamanho do stop bit	0 1	1 stop bit de cada vez 1,5 stop bit se o tam. do caracter dor igual a 5 bits 2 stop bits se o tam. do caracter for igual 6,7 ou 8 bits.
3	Paridade	0 1	Gerador de paridade está desativado Gerador de paridade está ativado
4	Tipo de paridade	0 1	Se a paridade estiver habilitada (bit 3=1), bit de paridade é gerado para fazer o número de bits 1 do byte ser impar Se a paridade estiver habilitada (bit 3=1), o bit de paridade é gerado para fazer o número de bits 1 do byte ser par.
5	Extensão da paridade	0 1	Desabilitado. Se paridade habilitada é par, o bit de paridade tem valor constante zero, se for ímpar o valor constante será 1.
6	Set Break	0 1	Desabilitado. Depois de transmitido o dado, a linha é forçada para a condição de space(0)
7	Acesso a Baud Rate	0 1	Valor normal. Endereça o valor de baud rate.

Tabela 2-5 – Função dos bits em 3FBH

O próximo passo é ajustar a configuração do modem. Mesmo que a transmissão seja efetuada micro a micro, sem a presença de um modem, é importante configurá-lo, pelo menos de maneira a neutralizá-lo. O registrador usado para esta tarefa é o *Modem Control Register* (MCR) no endereço 3FCH onde a descrição de seus bit estão na tabela 2-6.

Bits	Nome	Valor	Função
0	Data Terminal Ready (DTR)	0 1	A linha DTR é forçada para 1, sinalizando ao modem que o terminal não está pronto. Alinha DTR é forçada para 0, sinalizando ao modem que o terminal está pronto.
1	Request To Send (RTS)	0 1	A linha RTS é forçada para 1, sinalizando ao modem que o PC não enviará dados. A linha RTS é forçada para zero perguntando ao modem se está pronto para enviar caracteres.
2	Out 1	X	Não usado.
3	Out 2	0 1	O sinal de interrupção do 8250 para o 8259 está desativado. O sinal de interrupção do 8250 para o 8259 está ativado.
4	Loop Back	0 1	Não testa o 8250. Testa o 8250, se a saída do 8250 estiver ligada à entrada do mesmo 8250, o dado enviado é imediatamente recebido.
5-7		000	Bits sempre em zero.

Tabela 2-6 – Descrição do registrador MCR

Normalmente, atribui-se o registrador MCR (3FCH) para 03H (0000011 binário) e ignora-se o resto. O valor 03H representa que os sinais DTR e RTS são enviados ao modem,

forçando-o a uma condição de pronto. Os sinais de sinalização de mensagens, estão descritos na tabela abaixo (tabela 2-7).

Sinais do computador para o modem	
DTR (data terminal ready)	Indica ao modem que o computador está ligado e pronto
RTS (request to send)	Indica ao modem que o computador deseja enviar dados
Sinais do modem para o computador	
DSR (data set ready)	Indica ao computador que o modem está ligado e pronto.
CTS (clear to send)	Indica ao computador que o modem está pronto para aceitar os dados da transmissão
DCD (data carrier detected)	Indica ao computador que o modem está estabilizado com outro modem com que está conectado.
RI (ring indicator)	Indicado que o telefone conectado com computador esta tocando

Tabela 2-7 – Descrição dos sinais de estabelecimento de comunicação (HandShaking)

O último registrador que precisa ser inicializado é o *interrupt enable register* (IER), que serve habilitar e desabilitar as interrupções descritas na tabela 2-8, cujo endereço é 3F9H.

Bits	Nome	Valor	Função
0	Enable Data Available	0 1	Não gerará interrupção quando um dado for recebido. Gerará interrupção quando um dado for movido para o RDR.
1	Enable THR empty	0 1	Não gerará interrupção quando o THR ficar vazio. Gerará interrupção quando o THR ficar vazio.
2	Enable Receive Status	0 1	Não gerará interrupção quando mudar o line status. Gerará a interrupção quando mudar o line status
3	Enable Modem Status	0 1	Não gerará interrupção quando mudar o modem status. Gerará interrupção quando mudar o modem status.
4-7		0000	Bits sempre em zero.

Tabela 2-8 – Descrição do registrador IER

2.7 PROTOCOLOS DE COMUNICAÇÃO

Geralmente os protocolos de comunicação são constituídos de três partes:

Cabeçalho	Mensagem	Consistência

Onde:

- cabeçalho: as informações de controle do pacote fazem parte deste campo (endereçamentos...);

- mensagem: este tipo de campo é formado pelas informações que serão transmitidas;

- consistência: são caracteres para a verificação de erros (paridade,...).

2.7.1 TIPOS DE PROTOCOLOS

Os protocolos podem ser classificados em função dos modos de transmissão e quanto as controles de transmissão. Os protocolos podem ser implementados como síncronos e assíncronos.

Quanto aos controles os protocolos podem ser orientados a caracter, que são protocolos que utilizam caracteres especiais para o controle de operação e tráfego de suas mensagens, como o BSC e protocolos orientados a bit que são protocolos que não utilizam caracteres especiais para delimitar blocos de mensagem. Todo controle é tratado a nível de bit como o X.25 CCITT.

3 TECNOLOGIAS ENVOLVIDAS NO PROTÓTIPO

Das tecnologias envolvidas no protótipo é importante destacar o uso de uma placa multi-serial, da balança e a impressora de código de barras. O protótipo foi implementado no ambiente de programação Delphi 4, acessando uma base local de dados *Access*.

3.1 CONSIDERAÇÕES SOBRE CÓDIGO DE BARRAS

O código de barras é um símbolo composto de barras paralelas com larguras e espaçamentos variados. Existem vários padrões de códigos de barras no mundo dos quais podemos citar o EAN-13, EAN-8, UPC-E, Code 39, Code 128 e outros.

Estabeleceu-se que as barras são regiões que não refletem a luz vermelha ao equipamento leitor, e o fundo (espaços) que refletem. Portanto códigos de barras não podem ser impressas com a coloração vermelha.

Para efetuar a leitura do código de barras o leitor faz uma leitura de todas as barras no sentido longitudinal, (da esquerda para a direita ou vice-versa). Quanto mais larga for a barra mais tempo o aparelho ficará medindo a ausência da reflexão. O mesmo acontece para a medida dos espaços entre as barras. Da medida do tempo de reflexão ou de sua ausência, o equipamento calcula a proporção das larguras e, mediante consulta de tabelas internas, faz a decodificação.

3.2 CONSIDERAÇÕES SOBRE A BALANÇA

A balança usada no protótipo é uma balança de 20Kg com um indicador digital Toledo modelo 9091, que segundo [TOL99], é destinado a uso conjunto com diferentes tipos de plataforma de pesagem, na pesagem de produtos, ou na contagem de peças, permitindo diversas combinações de capacidade e incrementos. Pertence a nova geração de indicação digital de peso, com eletrônica mais avançada, trazendo vantagens de rapidez, exatidão, flexibilidade e baixa manutenção.

3.2.1 PROTOCOLO DA BALANÇA

Não existem documentos formais sobre o protocolo da balança, a especificação dele foi adquirida, parte do manual e parte com o Centro de Treinamento Toledo.

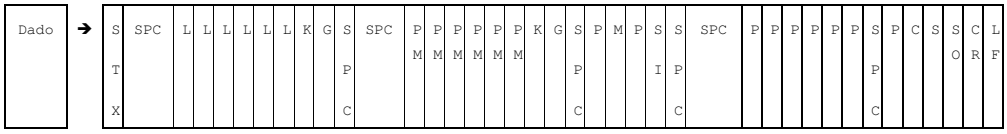
Descrição do protocolo:

Toda informação é constituída de 11 bits:

- 1 *start bit*
- 7 bits de dados
- 1 bit de paridade
- 2 *stop bits*

A velocidade de transmissão é selecionável via teclado da balança, somente nas velocidades 4800 e 9600 *baud*.

O formato padrão do protocolo (pacote) varia conforme parâmetros definidos na balança, via teclado da balança. O formato considerado neste protótipo é:



Onde:

Descrição das abreviaturas	
STX	Inicio do texto = 02H
SPC	Espaço = 20H
KG	Caracteres “KG”
L	Peso líquido incluindo o sinal e a vírgula
PM	Peso médio
PMP	Caracteres “PMP”
P	Quantidade de peças
SO	ShiftOut
CR	Carriage Return = 0DH
LF	Line Feed = 0AH

Tabela 3-1 – Descrições das abreviaturas do pacote da balança

Para maiores informações sobre a configuração do formato do pacote de dados da balança, consultar Anexo 1.

3.3 CONSIDERAÇÕES SOBRE A IMPRESSORA

A impressora utilizada no protótipo é uma impressora de etiquetas Datamax Allegro 2, que tem a capacidade de imprimir 17 tipos de código de barras, passando somente os dados os quais deseja-se transformar em um código de barras e um parâmetro que identifica o padrão de código.

A impressora usa o método de impressão térmico, onde queima um filme (ribbon) nas etiquetas para formar os desenhos.

3.3.1 PROTOCOLO DA IMPRESSORA

A informação enviada para a impressora têm como características:

Selecionável 7 ou 8 *bits*

HandShaking CTS/DTR

Tamanho do buffer da impressora é de aproximadamente 7000 *bytes*.

Sem *bit* de paridade.

Velocidade de transmissão selecionável entre 300, 600, 1200, 2400, 4800 e 9600 *baud*

Utiliza o conjunto de caracteres *ANSI ASCII*.

As informações enviadas para a balança são comandos de formatação de etiquetas (recebe dados para a modelagem da etiqueta) e comandos de controle da impressora (ex.: feed, pause, reset).

3.4 PLACA MULTISERIAL

Os PC's geralmente possuem duas portas seriais, que não são suficientes para a aplicação do protótipo, pois, uma porta se destina ao mouse, outra para uma balança e a última para a impressora. A placa multiserial usada no protótipo é uma placa que nos fornece mais duas portas seriais, satisfazendo assim as necessidades dos protótipo.

Especificações: 16-bit barramento ISA, velocidade até 460.8K, endereços configuráveis COM1 até COM7, Interrupções configuráveis 3, 4, 5, 6, 7, 9, 10, 11, 12, 14, 15.

3.5 CONSIDERAÇÕES SOBRE O BANCO DE DADOS UTILIZADO

O protótipo usa um banco de dados Access onde estão as tabelas utilizadas no protótipo sendo descrito na tabela abaixo, Tabela 3-2.

Tabelas	Campos	Utilidade da informação
COM	Dispositivo	Identifica se na porta está conectada uma impressora ou uma balança.
	BaudRate	Compatibiliza a velocidade de transmissão da balança e da porta aonde se encontra conectada.
	DataBits	Número de databits definidos no protocola da balança.
	Port	Identifica qual COM será alocada.
	Active	Indica se a porta deve ser alocada.
	Parity	Indica a paridade utilizada pelo protocolo.
Lote	NumLote	Identificador único do lote.
	CodProduto	Código do produto que referencia a chave primária da tabela Produto.
	QtdLote	Quantidade adquirida pelo programa vindos da balança.
Produto	CodProduto	Identificador único dos produtos cadastrados.
	DesProduto	Descrição do produto.
	QtdEstoque	Quantidade no estoque do produto, valor atualizado com os dados obtidos da balança.

Tabela 3-2 Descrição das tabelas utilizadas pelo protótipo

4 DESENVOLVIMENTO DO PROTÓTIPO

O desenvolvimento deste protótipo foi feito no ambiente de programação Delphi, acessando, através de um *driver* nativo, uma base de dados *Access*, onde são armazenadas informações específicas da estação (configuração das COM's, e o que está ligado na respectiva COM (balança ou impressora)), assim como um cadastro básico de produtos e as saídas do protótipo referida como Lote.

Para acessar os dispositivos de comunicações (COM's) o programa se faz do uso de algumas API's (Application Program Interface) do Windows, das quais tem-se um resumo de suas funções no anexo 2.

4.1 ESPECIFICAÇÃO DO PROTÓTIPO

A especificação do protótipo foi realizada através de fluxogramas, onde são descritos os comandos e funções a serem processados pelos protótipo.

4.1.1 DIAGRAMA DO PROTÓTIPO

Na Figura 4-1 é apresentado o diagrama geral do protótipo, onde é demonstrado o fluxo das informações no protótipo. Inicialmente a balança informa o peso e a quantidade do que foi pesado e o operador informa o código do produto que está na balança. Então o software confere o código do produto, atualiza a quantidade do produto, gera um registro de lote e manda o modelo da etiqueta para a impressora.

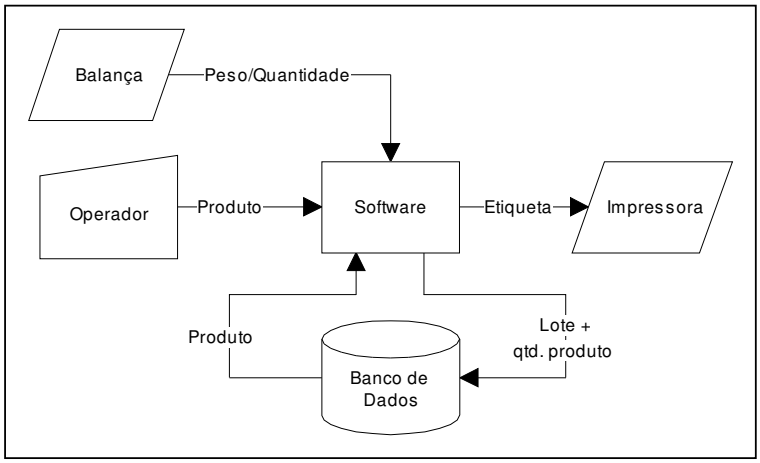


Figura 4-1 Diagrama geral do protótipo

4.1.2 DIAGRAMA DETALHADO DO PROTÓTIPO

Abaixo é demonstrado um nível maior de detalhamento do protótipo proposto. As propriedades dos objetos visuais das classes estão listadas no Anexo 3.

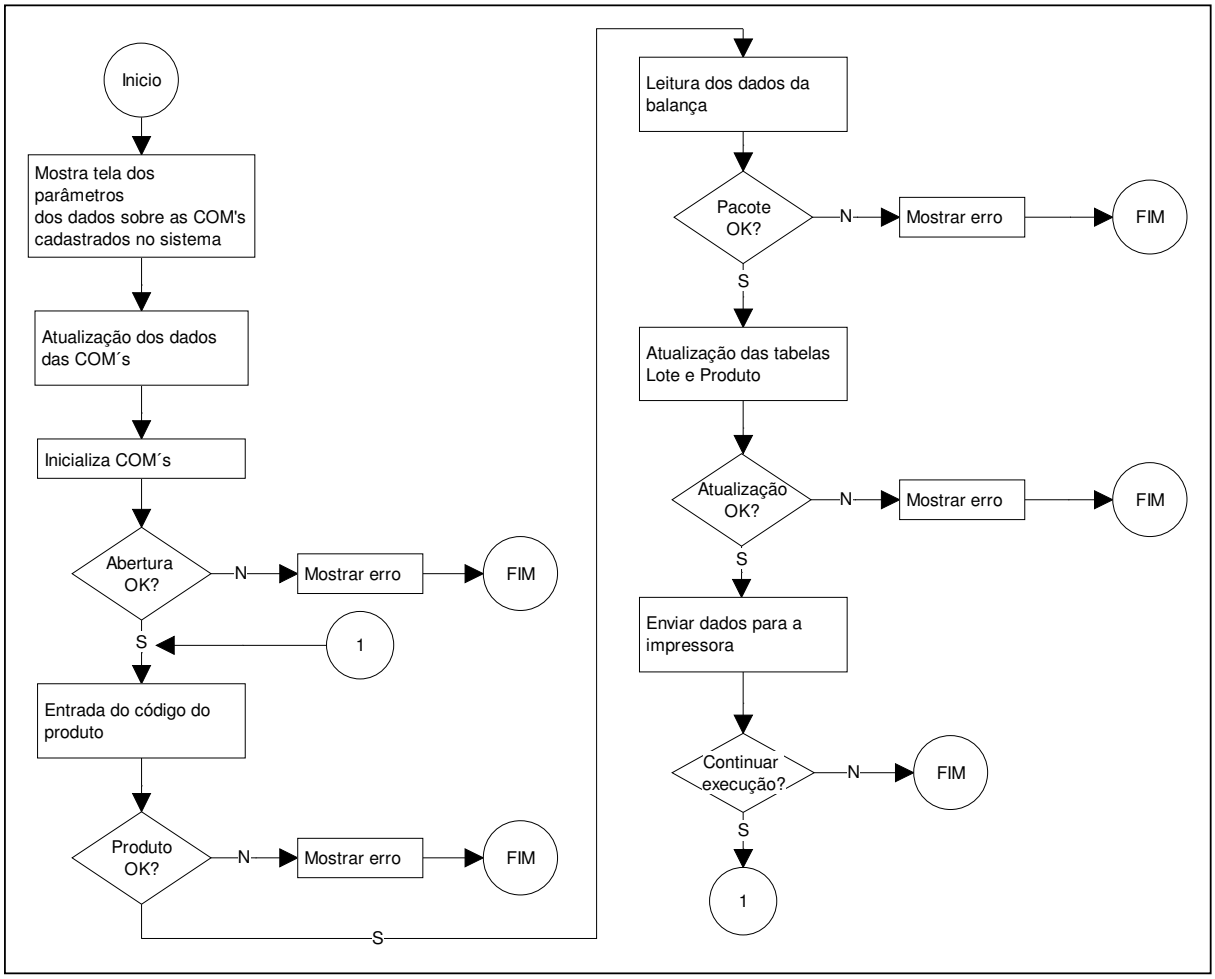


Figura 4-2 – Diagrama detalhado do protótipo

4.2 IMPLEMENTAÇÃO DO PROTÓTIPO

Conforme mencionado anteriormente, o protótipo foi desenvolvido em Delphi4, utilizando algumas API's do Windows necessárias para a comunicação com as portas, estas estão descritas no anexo 3.

O protótipo possui duas telas sendo uma tela principal, e uma outra para a configuração das portas utilizadas.

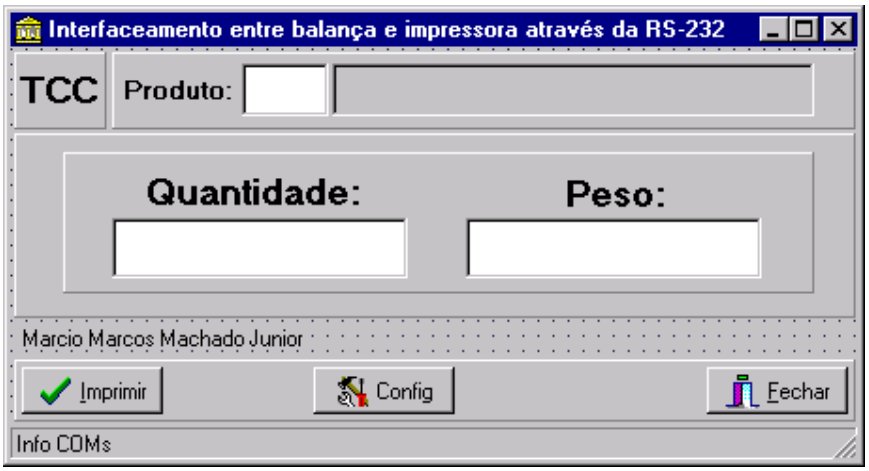


Fig4-1 – Tela principal do programa.

Campo Produto: Neste campo o operador informa o código do produto pesado, e contém a descrição do produto;

Campos Quantidade e Peso: Este campos são de leitura somente, e são preenchidos pelas entradas da balança;

Botão Imprimir: Confirma os dados para a emissão de uma etiqueta e atualização dos estoques, no banco de dados;

Botão Fechar: Termina a aplicação;

Botão Confia: Exibe a tela abaixo onde serão informadas informações sobre as comas.

Fig4-2 – Tela de configuração das portas.

Group box **Dispositivo**: Aqui identifica-se que tipo de dispositivo está conectado na porta;

Group box **Bits de dados**: Aqui informa-se o número de bits dos caracteres transmitidos que representam a informação;

Group box **Paridade**: Aqui identifica-se qual o tipo de paridade utilizada pelo protocolo;

Group box **Velocidade**: A identifica-se a velocidade que o dispositivos estão configurados para a efetivação da comunicação.

4.2.1 BREVE DESCRIÇÃO DA "OPERAÇÃO NA BALANÇA"

O operador precisa contar uma amostra de peças, as quais vão ser pesadas, colocar sobre a balança. Depois se faz necessário informar a balança quantas peças estão sobre a balança, para que a mesma possa calcular o peso médio de cada peça. Após esta operação deve-se colocar sobre a balanças as peças que deseja-se pesar e acionar o botão verde que se

encontra em seu painel, o qual enviará os dados que estão visíveis no visor para o computador.

5 CONCLUSÕES

A motivação ao trabalho foi a necessidade de automatizar um processo de apontamentos e identificação em um controle de estoques, onde o tamanho das peças torna inviável a contagem das mesmas de forma manual e a segurança no apontamento das informações.

Em relação as bibliografias necessárias ao desenvolvimento do trabalho, encontrei algumas dificuldades na documentação que descreve o protocolo da balança. As demais bibliografias não foram difíceis de encontrar.

No desenvolvimento do protótipo o maior problema encontrado foi conseguir e compreender as API's do Windows necessárias para o controle das portas. Quanto a linguagem utilizada e a implementação do protótipo não existiram maiores complicações

5.1 SUGESTÃO DE CONTINUIDADE

Como sugestão de continuidade deste trabalho sugiro o desenvolvimento de um software para a rastreabilidade do produto, ou seja, adicionar campo necessários na aquisição de dados para identificar no lote, o lote da matéria-prima, máquina em que o componente foi produzido, data e outros dados até um determinado nível desejado de rastreabilidade (rastrear o operador ou a máquina por exemplo), para a identificação de erros em um certo lote de produtos fabricado pela empresa. Isto visa a detecção de erros nos produtos fabricados, por exemplo, se uma matéria prima não estiver conforme, posso identificar o lote de todas as peças produzidas que utilizaram aquele lote de matéria prima.

6 REFERÊNCIAS BIBLIOGRÁFICAS

[DAT99] DATAMAX. **Allegro 2 – Operator´s Manual**. São Paulo, 1999.

[ERD92] ERDEI, Guillermo E. **Código de Barras – Desenvolvimento, Impressão e Controle de Qualidade**. São Paulo:Editora Makron Books, 1992

[TAF96] TAFFNER, Malcon Anderson, LOESCH, Cláudio, STRINGARI, Sergio. **Comunicação de Dados usando Linguagem "C"**. Blumenau: Editora da FURB: 1996.

[TAN94] TANENBAUM, Andew S. **Redes de Computadores**. Rio de Janeiro: Editora Campous.

[TOL99] TOLEDO. **9091 Indicador Digital – Manual do Usuário**. São Paulo, 1999.

[ZYN91] ZYNGIER, Mauro Luíz. **Código de Barras, da teoria à prática**. São Paulo: Editora Nobell, 1991

ANEXO 1

DESCRIÇÃO DOS FORMATOS DE DADOS DA SAÍDA SERIAL

1. Características de Gerais:

1.1 Tipo de Interface:

A interface de saída serial de dados pode ser do tipo RS232-C ou em Loop de corrente (ativo ou passivo, depende do cabo) com isolação óptica. O código utilizado no pacote de dados é ASCII e as possíveis taxas de transmissão são de 300, 1200, 2400, 4800 e 9600 Baud.

1.2 Interconexão:

a) Saída em RS 232-C:

Conector de 9 pinos tipo DB9 - Macho.

Pino 1	NC
Pino 2	RxD
Pino 3	TxD
Pino 4	DTR
Pino 5	Terra (GND)
Pino 6	DSR
Pino 7	NC
Pino 8	NC
Pino 9	NC

b) Saída em Loop de corrente:

Conector de 9 pinos tipo DB9 - Macho.

Pino 1	+28VCC
Pino 2	Tx+ (entrada de corrente)
Pino 4	Tx- (saída de corrente)
Pino 5	Terra (GND)

Demais pinos abertos.

1.3 Formato de um Byte:

	P01 ATÉ P04	P05 **(*)**
Taxa de transmissão	Programável	Programável
Paridade	Par	Nenhuma
Código	ASCII	ASCII
Número de bits de dados	7 (LSB primeiro)	8 (LSB primeiro)
Número de stop bits	2	1

(*) Não disponível no 9091 Rodoviário.

1.4 Cálculo do Checksum

O cálculo do Checksum é obtido através do complemento de 2 da soma de todos os bytes recebidos de STX, inclusive, à CR inclusive.

2. Formatos de Comunicação para Indicador Digital Modelo 9091 Pesador/Contador e Balanças Eletrônicas Pesadoras/Contadoras Modelos 3300 e 3400:

2.1 Comunicação utilizando o protocolo P01, usado comumente para transmitir dados ao Impressor Matricial de Etiquetas Modelo 351:

Programação da Balança	
C10 = d	Impressão de Bruto/Tara/Líquido
C11 = L	Envio de dados de peso em 1 linha
C13 = 300	Taxa de comunicação (bps)

C14 = P01	Formato de dados da saída serial
C28 = d	Sem impressão de Data/Hora
C31 = d	Sem impressão de Código

Formato para Balança Pesadora:

STX BBBBBBBB kg SPC TTTTTTT kg SPC TR SI SPC LLLLLLL kg SPC LIQ (SO) CR (CS) LF

Programação da Balança	
C10 = L	Impressão de Bruto ou Líquido
C11 = L	Envio de dados de peso em 3 linhas
C13 = 300	Taxa de comunicação (bps)
C14 = P01	Formato de dados da saída serial
C28 = L	Com impressão de Data/Hora
C31 = L	Com impressão de Código

Formato para Balança Pesadora:

STX DD/MM/AA SPC HH:MM SPC CCCCCC SPC BBBBBBBB kg SPC **ou** LLLLLLL kg SPC LIQ (SO) CR (CS) LF

Nota: na Balança 3400 é CCCCCCCCCCCC (12 caracteres).

ABREVIATURAS	
STX	Start of Text = 02H
(SO)	Shift Out = 0EH
SI	Shift In = 0FH
SPC	Espaço = 20H
CR	Carriage Return = 0DH
(CS)	Byte de Check-sum (se C12 = L)
LF	Line Feed = 0AH
C	Código

B	Peso Bruto incluindo o sinal e a vírgula
T	Tara incluindo a vírgula
L	Peso Líquido incluindo o sinal e a vírgula
DD/MM/AA	Dia/Mês/Ano
HH:MM	Hora e minutos

Observação:

Se a transmissão contínua estiver ligada (Parâmetro C15 = L) e existir sobrecarga da balança, então a saída de dados será interrompida.

2.2 Comunicação utilizando o protocolo P02, usado comumente para transmitir dados ao Impressor Térmico de Código de Barras Modelo 8861:

Programação da Balança	
C10 = d	Impressão de Bruto/Tara/Líquido
C11 = d	Envio de dados de peso em 3 linhas
C13 = 1200	Taxa de comunicação (bps)
C14 = P02	Formato de dados da saída serial
C28 = L	Com impressão de Data e Hora
C31 = L	Com impressão de Código

Formato para Balança Pesadora:

STX DD/MM/AA SPC HH:MM SPC SPC SPC SPC SPC SPC SPC SPC SPC CR (CS) LF

BBBBBBBB kg SPC CR (CS) LF

TTTTTTT kg SPC TR SPC CR (CS) LF

CCCCCC CR (CS) LF

LLLLLLL kg SPC LIQ CR (CS) LF

Nota: na Balança 3400 é CCCCCCCCCCCC (12 caracteres).

Programação da Balança	
C10 = d	Impressão de Bruto/Tara/Líquido
C11 = d	Envio de dados de peso em 3 linhas
C13 = 1200	Taxa de comunicação (bps)
C14 = P02	Formato de dados da saída serial
C28 = L	Com impressão de Data/Hora
C31 = d	Sem impressão de Código

Formato para Balança Pesadora:

STX DD/MM/AA SPC HH:MM SPC SPC SPC SPC SPC SPC SPC SPC SPC CR (CS) LF

CR (CS) LF

BBBBBBBB kg SPC CR (CS) LF

TTTTTTT kg SPC TR SPC CR (CS) LF

LLLLLLL kg SPC LIQ CR (CS) LF

Programação da Balança	
C10 = d	Impressão de Bruto/Tara/Líquido
C11 = d	Envio de dados de peso em 3 linhas
C13 = 1200	Taxa de comunicação (bps)
C14 = P02	Formato de dados da saída serial
C28 = d	Sem impressão de Data/Hora
C31 = d	Sem Impressão de Código

Formato para Balança Pesadora:

STX CR (CS) LF

CR (CS) LF

BBBBBBBB kg SPC CR (CS) LF

TTTTTTT kg SPC TR SPC CR (CS) LF

LLLLLLL kg SPC LIQ CR (CS) LF

Programação da Balança	
C10 = L	Impressão de Bruto ou Líquido
C11 = d	Envio de dados de peso em 1 linha
C13 = 1200	Taxa de comunicação (bps)
C14 = P02	Formato de dados da saída serial
C28 = d	Sem impressão de Data/Hora
C31 = L	Com impressão de Código

Formato para Balança Pesadora:

STX CR (CS) LF

CR (CS) LF

CR (CS) LF

CCCCCC CR (CS) LF

BBBBBBBB **ou** LLLLLLL kg SPC CR (CS) LF

Nota: na Balança 3400 é CCCCCCCCCCCC (12 caracteres).

Programação da Balança	
C10 = L	Impressão de Bruto ou Líquido
C11 = d	Envio de dados de peso em 3 linhas
C13 = 1200	Taxa de comunicação (bps)
C14 = P02	Formato de dados da saída serial
C28 = L	Com impressão de Data/Hora
C31 = L	Com impressão de Código

Formato para Balança Pesadora:

STX DD/MM/AA SPC HH:MM SPC SPC SPC SPC SPC SPC SPC SPC SPC CR (CS) LF

CR (CS) LF

CR (CS) LF

CCCCCC CR (CS) LF

BBBBBBB **ou** LLLLLLL kg SPC LIQ CR (CS) LF

Nota: na Balança 3400 é CCCCCCCCCCCC (12 caracteres).

Programação da Balança	
C10 = L	Impressão de Bruto ou Líquido
C11 = d	Envio de dados de peso em 3 linhas
C13 = 1200	Taxa de comunicação (bps)
C14 = P02	Formato de dados da saída serial
C28 = d	Sem impressão de Data/Hora
C31 = d	Sem impressão de Código

Formato para Balança Pesadora:

STX CR (CS) LF

CR (CS) LF

CR (CS) LF

CR (CS) LF

BBBBBBBB **ou** LLLLLLL kg SPC CR (CS) LF

ABREVIATURAS	
STX	Start of Text = 02H
SPC	Espaço = 20H
CR	Carriage Return = 0DH
(CS)	Byte de Check-sum (se C12 = L)
LF	Line Feed = 0AH
C	Código
B	Peso Bruto incluindo o sinal e a vírgula
T	Tara incluindo a vírgula

L	Peso Líquido incluindo o sinal e a vírgula
DD/MM/AA	Dia/Mês/Ano
HH:MM	Hora e minutos

2.3 Comunicação utilizando o protocolo P03:

Programação da Balança	
C14 = P03	Formato de dados da saída serial

Formato para Balança Pesadora:

STX,SWA,SWB,SWC,I,I,I,I,I,I,I,T,T,T,T,T,T,CR,(CS)

SWA - STATUS WORD "A":	
BIT 2, 1 e 0	001 = Display x 10 010 = Display x 1 011 = Display x 0.1 100 = Display x 0.01 101 = Display x 0.001 110 = Display x 0.0001
BIT 4 e 3	01 = Tamanho do incremento é 1 10 = Tamanho do incremento é 2 11 = Tamanho do incremento é 5
BIT 6 e 5	01 sempre
BIT 7	Paridade par

SWB - STATUS WORD "B":		
BIT 0	Peso Líquido	= 1
BIT 1	Peso Negativo	= 1
BIT 2	Sobrecarga	= 1

BIT 3	Motion	= 1
BIT 4	Sempre	= 1
BIT 5	Sempre	= 1
BIT 6	Se Auto Zerado	= 1
BIT 7	Paridade Par	

SWC - STATUS WORD "C":		
BIT 0	Sempre	= 0
BIT 1	Sempre	= 0
BIT 2	Sempre	= 0
BIT 3	Tecla Imprimir	= 1
BIT 4	Expandido	= 1
BIT 5	Sempre	= 1
BIT 6	Sempre	= 1
BIT 7	Paridade Par	

Observações:

Se existir sobrecarga da balança, o campo de peso IIIIII apresentará 000000.

A taxa de atualização da saída serial de dados no modo contínuo é de 1/185ms (5,4 atualizações por segundo) a 4800 bps, e de 1/165ms (6,1 atualizações por segundo) a 9600 bps. Os três níveis de filtragem digital não afetam a taxa de atualização.

O tempo máximo de estabilização da indicação depende do filtro digital selecionado:

Sem filtro	1,10 seg.
Filtro leve	1,60 seg.
Filtro médio	2,80 seg.
Filtro pesado	3,75 seg.

ABREVIATURAS	
STX	Start of Text = 02H
SPC	Espaço = 20H
CR	Carriage Return = 0DH
(CS)	Byte de Check-sum (se C12 = L)
I	Peso indicado no Display (Líquido ou Bruto)
T	Tara

2.4 Comunicação utilizando o protocolo P04, usado comumente para transmitir dados aos Impressores PRINTWEIGHT ou EPSON LX-300:

Programação da Balança	
C14 = P04	Formato de dados da saída serial

Este protocolo é igual ao P01, só que sem o envio do caracter (SI).

2.5 Comunicação utilizando o protocolo P05, usado comumente para transmitir dados a microcomputadores, PDVs e outros periféricos, com Handshake de software:

Programação da Balança	
C14 = P05	Formato de dados da saída serial

Formato para Balança Pesadora:

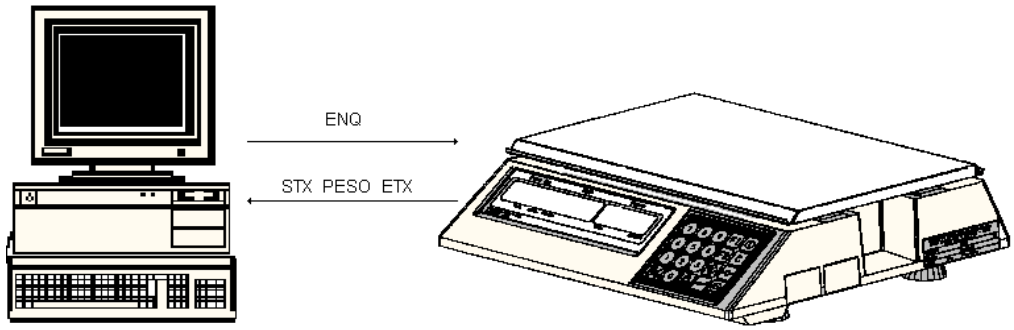
ENQ STX BBBBBBB **ou** LLLLLLL ETX

ABREVIATURAS	
ENQ	Enquire = 05H

STX	Start of Text = 02H
B	Peso Bruto incluindo o sinal e a vírgula
L	Peso Líquido incluindo o sinal e a vírgula
ETX	End of Text = 03H

Observações:

1. **Não transmite nº de peças.** A informação do peso só será transmitida na condição de não movimento na plataforma/prato de pesagem, e quando o microcomputador ou PDV ou outro periférico enviar à balança o caracter de controle "ENQ" solicitando a transmissão do peso:



2. Para uso do protocolo P05, em produto instalado no campo, adquirido normalmente sem a indicação que utilizaria o protocolo P05, **há a necessidade de mudar os jumpers da placa RS-232** (W1 e W2) de 1/2 para 2/3, serviço esse que só pode ser executado por técnico autorizado Toledo, pois há a necessidade de romper o lacre da balança para acessar a placa.

2.6 Comunicação para Balança Contadora:

É igual aos formatos já descritos para a Balança Pesadora, a menos de:

- Sempre será enviado somente um peso (bruto **ou** líquido);

- Nos outros protocolos (versão pesadora), onde tínhamos o envio de:

TTTTTTT kg SPC TR SI

teremos

GGGGGGG kg SPC PMP

e

LLLLLLL SPC kg SPC LIQ

teremos

HHHHHHH SPC PCS

onde:

GGGGGGG é peso médio por peça (incluindo a vírgula ou ponto)

HHHHHHH é número de peças

Na versão contadora, o parâmetro C10-IMPRESSÃO DE 1 OU 3 PESOS, se ligado, não transmite o peso e se desligado, só transmite peso bruto ou líquido.

Observação:

Não há diferença de protocolo na versão pesadora e contadora, no caso do protocolo P03.

3. Formatos de Comunicação para Indicador Digital Modelo 9091 Rodoviário:

3.1 Comunicação utilizando o protocolo P01, usado comumente para transmitir dados ao Impressor Matricial de Etiquetas Modelo 351:

Programação da Balança	
C10 = d	Impressão de Bruto/Tara/Líquido

C11 = 1L	Envio de dados de peso em 1 linha
C13 = 300	Taxa de comunicação (bps)
C14 = P01	Formato de dados da saída serial
C28 = d	Sem impressão de Data/Hora
C31 = d	Sem impressão de Código
C38 = d	Sem Impressão de Numerador Consecutivo

Formato:

STX CCCCCC SPC BBBBBBB kg SPC TTTTTTT kg SPC TR SI SPC LLLLLLL kg SPC LIQ (SO) CR (CS) LF

Programação da Balança	
C10 = L	Impressão de Bruto ou Líquido
C11 = 1L	Envio de dados de peso em 1 linha
C13 = 300	Taxa de comunicação (bps)
C14 = P01	Formato de dados da saída serial
C28 = d	Sem impressão de Data/Hora
C31 = d	Sem impressão de Código
C38 = d	Sem impressão de Numerador Consecutivo

Formato:

STX CCCCCC SPC BBBBBBB kg SPC **ou** LLLLLLL kg SPC LIQ (SO) CR (CS) LF

Programação da Balança	
C10 = d	Impressão de Bruto/Tara/Líquido
C11 = 1L	Envio de dados de peso em 1 linha
C13 = 300	Taxa de comunicação (bps)
C14 = P01	Formato de dados da saída serial

C28 = d	Sem impressão de Data/Hora
C31 = L	Com impressão de Código
C38 = L	Com impressão de Numerador Consecutivo

Formato:

STX NNNNNN SPC CCCCCC SPC BBBBBBB kg SPC TTTTTTT kg SPC SI SPC LLLLLLL kg SPC LIQ (SO) CR (CS) LF

Programação da Balança	
C10 = L	Impressão de Bruto ou Líquido
C11 = 1L	Envio de dados de peso em 1 linha
C13 = 300	Taxa de comunicação (bps)
C14 = P01	Formato de dados da saída serial
C28 = d	Sem impressão de Data/Hora
C31 = L	Com impressão de Código
C38 = L	Com impressão de Numerador Consecutivo

Formato:

STX NNNNNN SPC CCCCCC SPC BBBBBBB kg SPC **ou** LLLLLLL kg SPC LIQ (SO) CR (CS) LF

Programação da Balança	
C10 = d	Impressão de Bruto/Tara/Líquido
C11 = 1L	Envio de dados de peso em 1 linha
C13 = 300	Taxa de comunicação (bps)
C14 = P01	Formato de dados da saída serial
C28 = L	Com impressão de Data/Hora
C38 = L	Com impressão de Numerador

	Consecutivo

Formato:

STX DD/MM/AA SPC HH:MM SPC NNNNNN SPC CCCCCC SPC BBBBBBB kg SPC TTTTTTT SPC TR SI SPC LLLLLLL kg SPC LIQ (SO) CR (CS) LF

ABREVIATURAS	
STX	Start of Text = 02H
(SO)	Shift Out = 0EH
SI	Shift In = 0FH
SPC	Espaço = 20H
CR	Carriage Return = 0DH
(CS)	Byte de Check-sum (se C12 = L)
LF	Line Feed = 0AH
N	Numerador Consecutivo
B	Peso Bruto incluindo o sinal e a vírgula
T	Tara incluindo a vírgula
L	Peso Líquido incluindo o sinal e a vírgula
C	Identificação (código)

Observação:

Se a transmissão contínua estiver ligada (Parâmetro C15 = L) e existir sobrecarga da balança, então a saída de dados será interrompida.

3.2 Comunicação utilizando o protocolo P02, usado comumente para transmitir dados ao Impressor Térmico de Código de Barras Modelo 8861:

Programação da Balança	
C10 = d	Impressão de Bruto/Tara/Líquido
C11 = 3L	Envio de dados de peso em 3 linhas
C13 = 1200	Taxa de comunicação (bps)
C14 = P02	Formato de dados da saída serial
C28 = L	Com impressão de Data/Hora
C31 = L	Com impressão de Código
C38 = d	Sem impressão de Numerador Consecutivo

Formato:

STX DD/MM/AA SPC HH:MM SPC SPC SPC SPC SPC SPC SPC SPC SPC CR (CS) LF

BBBBBBB kg SPC CR (CS) LF

TTTTTTT kg SPC TR SPC CR (CS) LF

CCCCCC CR (CS) LF

LLLLLLL kg SPC LIQ CR (CS) LF

Programação da Balança	
C10 = L	Impressão de Bruto ou Líquido
C11 = 3L	Envio de dados de peso em 3 linhas
C13 = 1200	Taxa de comunicação (bps)
C14 = P02	Formato de dados da saída serial
C28 = d	Sem impressão de Data/Hora
C31 = L	Com impressão de Código
C38 = d	Sem impressão de Numerador Consecutivo

Formato:

STX CR (CS) LF

CR (CS) LF

CR (CS) LF

CCCCCC CR (CS) LF

BBBBBBB **ou** LLLLLLL kg SPC LIQ CR (CS) LF

Programação da Balança	
C10 = L	Impressão de Bruto ou Líquido
C11 = 3L	Envio de dados de peso em 3 linhas
C13 = 1200	Taxa de comunicação (bps)
C14 = P02	Formato de dados da saída serial
C28 = L	Com impressão de Data/Hora
C31 = L	Com impressão de Código
C38 = d	Sem impressão de Numerador Consecutivo

Formato:

STX DD/MM/AA SPC HH:MM SPC SPC SPC SPC SPC SPC SPC SPC SPC CR (CS) LF

CR (CS) LF

CR (CS) LF

CCCCCC CR (CS) LF

BBBBBBB **ou** LLLLLLL kg SPC LIQ CR (CS) LF

Programação da Balança	
C10 = L	Impressão de Bruto ou Líquido
C11 = 3L	Envio de dados de peso em 3 linhas

C13 = 1200	Taxa de comunicação (bps)
C14 = P02	Formato de dados da saída serial
C28 = d	Data e hora na impressão
C31 = L	Com impressão de Código
C38 = d	Sem impressão de Numerador Consecutivo

Formato:

STX CR (CS) LF

CR (CS) LF

CR (CS) LF

CCCCCC CR (CS) LF

BBBBBBB kg **ou** LLLLLLL kg SPC LIQ CR (CS) LF

ABREVIATURAS	
STX	Start of Text = 02H
(SO)	Shift Out = 0EH
SI	Shift In = OFH
SPC	Espaço = 20H
CR	Carriage Return = 0DH
(CS)	Byte de Check-sum (se C12 = L)
LF	Line Feed = 0AH
C	Numerador Consecutivo
B	Peso Bruto incluindo o sinal e a vírgula
T	Tara incluindo a vírgula
L	Peso Líquido incluindo o sinal e a vírgula
I	Identificação (código)
DD/MM/AA	Dia/Mês/Ano
HH:MM	Hora e minutos

3.3 Comunicação utilizando o protocolo P03:

Programação da Balança	
C14 = P03	Formato de dados da saída serial

Igual ao item 2.3.

3.4 Comunicação utilizando o protocolo P04, usado comumente para transmitir dados aos Impressores PRINTWEIGHT ou EPSON LX-300:

Programação da Balança	
C14 = P04	Formato de dados da saída serial

Este protocolo é igual ao P01, só que sem o envio do caracter (SI).

Documento elaborado por Carlos Alberto Polonio - Analista de Produtos da Linha Industrial Standard

Tel. (011) 6160-9042 ou 6160-9000 Ramal 9042

Fax (011) 6914-6917 ou 6915-7766

E-mail: ind@toledobrasil.com.br

ANEXO 2

Descrições básicas das API´s do windows utilizadas

PurgeComm: A função descarta todos as caracteres dos buffers de input e output de um determinado dispositico de comunicação. Pode também terminar com leituras e gravações pendentes no dispositivo.

```
BOOL PurgeComm(

  HANDLE hFile,
  DWORD dwFlags
);
```

SetCommMask: especifica o conjunto de eventos para serem monitorados em dispositivo de comunicação.

```
BOOL SetCommMask(

  HANDLE hFile,
  DWORD dwEvtMask
);
```

CreateFile: a função CreateFile, cria ou abre os seguintes objetos e retorna o handle que é usado para acessar o objeto.

- files
- pipes
- mailslots
- communications resources
- disk devices (Windows NT only)
- consoles

directories (open only)

```
HANDLE CreateFile(

  LPCTSTR lpFileName,
  DWORD dwDesiredAccess,
  DWORD dwShareMode,
  LPSECURITY_ATTRIBUTES lpSecurityAttributes,
  DWORD dwCreationDistribution,
  DWORD dwFlagsAndAttributes,
  HANDLE hTemplateFile
);
```

SetCommState: esta função configura dispositivos de comunicação de acordo com as especificações do data control block (DCB). A função reinicializa todo hardware e controles, mas não esvazia filas de entrada e saída. É usada para configurar o 8250 (UART). Os seguintes valores são restritos para ByteSize e Stopbits da estrutura:

.
.

```
BOOL SetCommState(

  HANDLE hFile,
  LPDCB lpDCB
 );
```

SetCommTimeouts: esta função seta s parâmetro de time-out para todas as operações de leitura e gravação no dispositivo especificado.

```
BOOL SetCommTimeouts(

  HANDLE hFile,
  LPCOMMTIMEOUTS lpCommTimeouts
 );
```

EscapeCommFunction: direciona um dispositivo de comunicação para executar uma função avançada.

```
BOOL EscapeCommFunction(

  HANDLE hFile,
  DWORD dwFunc
 );
```

Funções estendidas:

CLRDTR
CLRRTS
SETDTR
SETRTS
SETXOFF
SETXON
SETBREAK
CLRBREAK
CLRBREAK

WriteFile: esta função grava dados em um arquivo é foi desenvolvida para ambas operações síncronas e assíncronas. A função começa a gravar no arquivo na posição indicada pelo apontador do arquivo. Depois da operação de gravação é concluída, o apontador é ajustado pelo número de bytes gravados, exceto quando o arquivo é aberto com FILE_FLAG_OVERLAPPED. Se o handle fai criado para “overlapped’s” entradas e saídas, a aplicação precisa ajustar o ponteiro do arquivo depois que a operação de gravação foi concluída.

```
BOOL WriteFile(

  HANDLE hFile,
```

```
  LPCVOID lpBuffer,
  DWORD nNumberOfBytesToWrite,
  LPDWORD lpNumberOfBytesWritten,
  LPOVERLAPPED lpOverlapped
 );
```

WaitForSingleObject: retorna quando um dos seguintes casos ocorrem:

.
.

```
DWORD WaitForSingleObject(

  HANDLE hHandle,
  DWORD dwMilliseconds
 );
```

ReadFile: esta função le dados de um arquivo, começando na posição indicada pelo apontador. Depois da operação de leitura foi completa, o apontador do arquivo é ajustado para o número de bytes lidos, amenos que o arquivo tenha sido aberto com o tributo de overlapped. Se o handle do arquivo foi aberto para overlapped entrada e saída, a aplicação necessita ajustar a posição do apontador do arquivo depois da operaçãi de leitura.

```
BOOL ReadFile(

  HANDLE hFile,
  LPVOID lpBuffer,
  DWORD nNumberOfBytesToRead,
  LPDWORD lpNumberOfBytesRead,
  LPOVERLAPPED lpOverlapped
 );
```

ResetEvent: esta função seta o estado do evento especificado para não sinalizado.

```
BOOL ResetEvent(

  HANDLE hEvent
 );
```

WaitCommEvent : esta função espera algum evento ocorrer para um determinado dispositivo de comunicação. O conjunto de eventos monitorados por esta função estão contidos no “event mask” associado com o handle do dispositivo.

```
BOOL WaitCommEvent(
  HANDLE hFile,
  LPDWORD lpEvtMask,
  LPOVERLAPPED lpOverlapped,
 );
```

ANEXO 3

```
program PBalanca;

uses
  Forms,
  UPricipal in 'UPricipal.pas' {frmPrincipal},
  UDMPrincipal in 'UDMPrincipal.pas' {dmPrincipal: TDataModule},
  UDMCOM in 'UDMCOM.pas' {dmCOM: TDataModule},
  UImprimeEtiqueta in 'UImprimeEtiqueta.pas',
  UConfiguracao in 'UConfiguracao.pas' {frmConfiguracao};

{$R *.RES}

begin
  Application.Initialize;
  Application.Title := 'Entrada de lotes';
  Application.CreateForm(TdmCOM, dmCOM);
  Application.CreateForm(TfrmPrincipal, frmPrincipal);
  Application.CreateForm(TdmPrincipal, dmPrincipal);
  Application.CreateForm(TfrmConfiguracao, frmConfiguracao);
  Application.Run;
end.


unit UPricipal;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  ExtCtrls, StdCtrls, Buttons, ComCtrls, Menus, SerialTCC, Mask, DBCtrls,
  DBCGrids, db;

type
  TfrmPrincipal = class(TForm)
    Panel1: TPanel;
    edCodProduto: TEdit;
    edDesProduto: TEdit;
    Panel3: TPanel;
    sbPrincipal: TStatusBar;
    Panel4: TPanel;
    btnConfirma: TBitBtn;
    BitBtnSair: TBitBtn;
    btnPrintLastLabel: TBitBtn;
    Panel5: TPanel;
    Label2: TLabel;
    Label4: TLabel;
    edPeso: TEdit;
    edQuantidade: TEdit;
    btConfig: TBitBtn;
    Panel6: TPanel;
    Label3: TLabel;
    Label1: TLabel;
```

```
    procedure btnConfirmaClick(Sender: TObject);
    procedure FormActivate(Sender: TObject);
    function VerificaNumero(Sender: TEdit):boolean;
    procedure Modificado(Sender: TObject);
    procedure BitBtnSairClick(Sender: TObject);
    procedure btnPrintLastLabelClick(Sender: TObject);
    procedure Configuraes1Click(Sender: TObject);
    procedure edCodProdutoExit(Sender: TObject);

  private
    { Private declarations }
  public
    { Public declarations }
  end;

//const

//   Inverter=1;
//   Colocar=2;
//   Tirar=3;
var
   frmPrincipal     :TfrmPrincipal;
   ErroPortas       :boolean;
   ErroCamposNulos  :boolean;
   PortasComErro    :string;
implementation

uses UDMPrincipal, UConfiguracao, UDMCOM,
  UImprimeEtiqueta,  UUtil;//winbase;

{$R *.DFM}

procedure InicializaComs; forward;

function TfrmPrincipal.VerificaNumero;
begin
   VerificaNumero:=True;
   if Sender.text<>'' then
   try
      StrToInt(Sender.text);
   except
      On EConvertError do
      begin
         ShowMessage('Número inválido.');
         Sender.SetFocus;
         VerificaNumero:=False;
      end;
   end;
end;

procedure VerificaCampos(edit:TEdit);
begin
   if edit.text='' then
   begin
      ShowMessage('O campo '+edit.hint+' não pode ser nulo.');
      ErroCamposNulos:=True;
   end;
end;
```

```
procedure TfrmPrincipal.btnConfirmaClick(Sender: TObject);
begin

//Consiste campos
   ErroCamposNulos:=False;
   VerificaCampos(edCodProduto );
   VerificaCampos(edPeso);
   VerificaCampos(edQuantidade);

   if not ErroCamposNulos then
   begin

      ImprimeEtiqueta(DMCOM.COMImpressora,  '1',  '1',  edCodProduto.text,
edDesProduto.text, edQuantidade.text);

      frmPrincipal.Cursor:=crDefault;

      edQuantidade.text:='';
      edPeso.text:='';

   end;

end;

procedure ConfiguraCom(var com:TSerialTCC;porta:integer);
var aux2:string;
begin
   frmPrincipal.sbPrincipal.panels[0].text:='Abrindo                    porta
'+IntToStr(porta);
   with dmPrincipal.tbCom do
   begin
      if FindKey([porta]) then
      begin
         if FieldByName('Active').asString='S' then
         begin
            with com do
            begin
               Baudrate:=FieldByName('BaudRate').asInteger;
               if FieldByName('DataBits').asInteger=7 then
                  DataBits:=d7Bit
               else
                  DataBits:=d8Bit;
               Port    :=FieldByName('Port'    ).asInteger;

               if FieldByName('Tipo').asString='I' then
                  dmCom.COMImpressora:=COM;

               aux2:=FieldByName('Parity').asString;

               case aux2[1] of

                  'O':parity:=paOdd;

                  'N':parity:=paNone;

               end;

               active:=true;
```

```
               if ErrorCode>0 then
               begin

                  ErroPortas:=true;
                  PortasComErro:=PortasComErro+' '+IntToStr(porta)+' '+
                  SysErrorMessage(ErrorCode)+';';

               end;
            end;
         end;
      end;
   end;
end;

procedure InicializaComs;
begin
   ErroPortas:=false;
   PortasComErro:='';
   frmPrincipal.sbPrincipal.panels[0].text:='Configurando COMs...';
   dmPrincipal.tbCom.open;
   with dmCOM do
   begin
      COM1.Active:=false;
      COM2.Active:=false;
      COM3.Active:=false;
      COM4.Active:=false;
      ConfiguraCom(COM1,1);
      ConfiguraCom(COM2,2);
      ConfiguraCom(COM3,3);
      ConfiguraCom(COM4,4);
   end;
   if ErroPortas then
      frmPrincipal.sbPrincipal.panels[0].text:='Erro   na   abertura   das
portas:'+PortasComErro
   else
      frmPrincipal.sbPrincipal.panels[0].text:='COMs OK!';
   dmPrincipal.tbCom.Close;
end;

procedure TfrmPrincipal.FormActivate(Sender: TObject);
//var
//   f:file;

begin

  {$I-}
//   AssignFile(f, 'c:\zm.pcx');
//   FileMode := 0;
//   Reset(f);
//   CloseFile(f);
  {$I+}

  {
   if (IOResult <> 0) then
   begin
      ShowMessage('Arquivo c:\TCC.PCX inexistente.');
      frmPrincipal.Close;
   end;
 }
```

```
   InicializaComs;

end;

procedure TfrmPrincipal.Modificado(Sender: TObject);
begin
   if not btnConfirma.enabled then btnConfirma.enabled:=True;
end;

procedure TfrmPrincipal.BitBtnSairClick(Sender: TObject);
begin
   close;
end;

procedure TfrmPrincipal.btnPrintLastLabelClick(Sender: TObject);
begin

   PrintLastLabel(DMCOM.COMImpressora);

end;

procedure TfrmPrincipal.Configuraes1Click(Sender: TObject);
begin
   frmConfiguracao:=TfrmConfiguracao.create(frmPrincipal);
   dmPrincipal.tbCom.open;
   frmConfiguracao.showmodal;
   InicializaComs;
   frmConfiguracao.Free;
end;

procedure TfrmPrincipal.edCodProdutoExit(Sender: TObject);
begin

   if edCodProduto.text<>'' then

      with dmPrincipal.tbProduto do
      begin

         if FindKey([StrToInt(edCodproduto.text)]) then

            edDesProduto.text:=FieldByName('DesProduto').asString

         else
         begin

            edDesProduto.text:='';
            ShowMessage('Produto não cadastrado.');

         end;

      end;

end;

end.


unit UConfiguracao;
```

```
interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  DBCtrls, StdCtrls, Mask, ExtCtrls, ComCtrls;

type
  TfrmConfiguracao = class(TForm)
    Panel1: TPanel;
    Panel3: TPanel;
    DBRadioGroup1: TDBRadioGroup;
    DBEdit1: TDBEdit;
    Label1: TLabel;
    DBComboBox1: TDBComboBox;
    Label2: TLabel;
    DBRadioGroup2: TDBRadioGroup;
    DBRadioGroup3: TDBRadioGroup;
    DBRadioGroup4: TDBRadioGroup;
    Panel4: TPanel;
    DBNavigator1: TDBNavigator;
    StatusBar1: TStatusBar;
    Panel6: TPanel;
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  frmConfiguracao: TfrmConfiguracao;

implementation

uses UDMPrincipal;

{$R *.DFM}

end.


unit UDMCOM;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  SerialTCC;

type
  EFormatoIlegal = class(Exception);
  TdmCOM = class(TDataModule)
    COM1: TSerialTCC;
    COM2: TSerialTCC;
    COM3: TSerialTCC;
    COM4: TSerialTCC;
    COMImpressora: TSerialTCC;
    procedure COMRxData(Sender: TObject);
  private
    { Private declarations }
```

```
  public
    { Public declarations }
  end;

var
  dmCOM: TdmCOM;

implementation

uses UPricipal;

{$R *.DFM}

procedure TdmCOM.COMRxData(Sender: TObject);
var
   a:integer;
   c:char;
   mensagem, aux:string;
begin
   mensagem:='';
   sleep(200);
   while (sender as TSerialTCC).ReadChar(c)>0 do
   begin
      mensagem:=mensagem+c;
   end;
   if length(mensagem)>0 then
      if sender<>COMImpressora then
      begin
         with frmPrincipal do
         begin
            edPeso.text      :=mensagem[22]+
                               mensagem[23]+
                               mensagem[24]+
                               mensagem[25]+
                               mensagem[26]+
                               mensagem[27];
            edQuantidade.text:=mensagem[50]+
                               mensagem[51]+
                               mensagem[52]+
                               mensagem[53]+
                               mensagem[54]+
                               mensagem[55];
         end;
      end;

end;

end.

unit UDMPrincipal;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  Db, DBTables, Serial3;

type
  TdmPrincipal = class(TDataModule)
```

```
    tbCOM: TTable;
    tbProduto: TTable;
    tbLote: TTable;
    dstCOM: TDataSource;
    dstProduto: TDataSource;
    dstLote: TDataSource;
    dsqLote: TDataSource;
    qryLote: TQuery;
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  dmPrincipal: TdmPrincipal;

implementation

uses UPricipal;

{$R *.DFM}


end.

unit UImprimeEtiqueta;

interface

   uses SerialTCC, SysUtils, Dialogs, Windows;

   procedure                  ImprimeEtiqueta(COM:TSerialTCC;NumLote:string;
NumSeqLote:string;          CodProduto:String;          DesResumida:String;
Quantidade:string);
   procedure TogglePause(COM:TSerialTCC);
   procedure PrintLastLabel(COM:TSerialTCC);

   function TestRS232(COM:TSerialTCC):boolean;
   function ImpressoraOK(COM:TSerialTCC;var status:string):boolean;
   function            ColocaMascara(Text:string;            Mascara:String;
Zerar:boolean):string;
   function Paused(COM:TSerialTCC):boolean;

implementation

const
   stx     = chr(2);
   soh     = chr(1);
   esc     = chr(27);
   enter   = chr(13);
   f9      = 120;
   Inverter= 1;
   Colocar = 2;
   Tirar   = 3;


procedure PrintLastLabel;
var
```

```
   status:String;
begin
   if impressoraOK(COM,status) then
      COM.WriteString(stx+'G'+enter)
   else
      ShowMessage(Status);
end;


function ImpressoraOk(COM:TSerialTCC;var Status:string):boolean;
var
   c:char;
   s:string[8];
begin
   ImpressoraOK:=true;
   COM.WriteString(stx+'k');
   sleep(100);
   c:=#0;
   COM.ReadChar(c);
   COM.ZapRxQueue;
   if c<>'Y' then
   begin
      ImpressoraOK:=False;
      status:='Sem resposta da impressora.';
   end
   else
   begin
      s:='';
      status:='';
      COM.WriteString(soh+'A');
      sleep(100);
      while COM.ReadChar(c)>0 do
      begin
         s:=s+c;
      end;

      if 'Y' in [s[1], s[2], s[3], s[6]] then
         ImpressoraOK:=false;
      if s[1]='Y' then
         status:='Interpretador ocupado; ';
      if s[2]='Y' then
         status:=status+'Sem papel; ';
      if s[3]='Y' then
         status:=status+'Sem Ribbon; ';
      if s[6]='Y' then
         status:=status+'Impressora pausada; ';
   end;

end;
function TestRS232(COM:TSerialTCC):boolean;
var
   s:string;
begin
   with COM do
   begin
      ZapRxQueue;
      WriteString(stx+'k');
      sleep(100);
      ReadString(s);
```

```
      if s='Y' then
         TestRS232:=true
      else
         TestRS232:=false;
   end;
end;

procedure TogglePause;
begin
   Com.WriteString(soh+'B');
end;

function Paused;
var s:string;
begin
   with com do
   begin
      ZapRxQueue;
      WriteString(soh+'A');
      sleep(1500);
      ReadString(s);
      if length(s)=6 then
         if s[6]='Y' then Paused:=true
         else             Paused:=false
      else
         paused:=true;
   end;
end;

function ColocaMascara;
var
   a,b:byte;
   Retorno:string;
   conta:byte;
begin

   Retorno:='';
   conta:=0;

   for a:=1 to length(Mascara) do if Mascara[a]=' ' then conta:=conta+1;

   if Length(Text)>Conta then
   begin
      ShowMessage('Numero maior que a máscara informada.'+enter+
                  'Máscara:"'+mascara+'"'+inttostr(length(mascara))+enter+
                  'Texto  :"'+text+'"'+inttostr(length(text)));
      exit;
   end;

   b:=length(mascara);

   for a:=length(Text) DownTo 1 do
   begin
      while mascara[b] <> ' ' do
      begin
         Retorno:=mascara[b]+Retorno;
         b:=b-1;
      end;
      b:=b-1;
```

```
      Retorno:=Text[a]+Retorno;
   end;

   if zerar then
       while b>0 do
       begin
           if mascara[b]=' ' then Retorno:='0'+Retorno
                             else Retorno:= mascara[b]+Retorno;
           b:=b-1;
       end
   else Retorno:=IntToStr(StrToInt(Retorno));

   ColocaMascara:=retorno;

end;

procedure ImprimeEtiqueta;
begin
   with COM do
   begin

      if Paused(COM) then TogglePause(COM);

      WriteString(
      stx + 'm'     + enter +  //Sets printer to metric
      stx + 'M3000' + enter +  //Maximum label length
      stx + 'c0660' + enter +  //Set continous paper length
      stx + 'f000'  + enter +  //Set form stop position
      stx + 'O0220' + enter +  //Set start off print position
      stx + 'e'     + enter +  //Select edge sensor
      stx + 'L'     + enter +  //Enter label formating mode
      'C0000'       + enter +  //Set column offset amount
      'R0000'       + enter +  //Ribon saver
      'H20'         + enter +  //Enters the printer heat setting
      'D11'         + enter +  //Set width and heigth size
      'SC'          + enter +  //Set slew rate speed for feeding label
      'PC'          + enter +  //Set for print cycle
      'z'           + enter +  //Zero conversion "0"
      '^00'         + enter +  //Sets count by amount
      '131100003300273 Lote:'+NumLote+enter+
      '141100002700263 '+CodProduto+ enter +
      '121100001950125 '+copy(DesResumida,1,30) + enter +
      '121100001650125 '+copy(DesResumida,31,20) + enter +
//      '121100001900230 '+copy(DesResumida,41,9)+ enter +
      '131100001050125  Data  Fab.:'+FormatDateTime('dd/mm/yyyy',now)+  enter
+
      '131100000400125 Quantidade: '+Quantidade + enter +
      '4f4310000150085        0000'        +        ColocaMascara(NumLote,'
',true)+ColocaMascara(NumSeqLote,'  ',true) + enter +
      '1X11000024001121052500003'     + enter +
      '1X11000024202801000301 54'     + enter +
      '1X1100000000110B530400003003' + enter +
      'E' + enter);
   end;
end;


end.
```

```
unit SerialTCC;
{
  Componente de interface serial
}

interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
  Forms, Dialogs,Windows;

type

  {--------------------------------------------------------------------------}
  { TSerial declareções dos tipos}

  eParity = ( paNone, paOdd ) ;
  eDataBits = (d7bit, d8bit ) ;
  sPorts = 1..9 ;
  eFlowControl = (fcNone, fcRTS_CTS, fcDTR_DSR, fcXON_XOF) ;
  eMsgState = (msNone, msStarted, msEnded, msDone) ;
  eModemInSignal = (msCTS, msDSR, msRLSD) ;
  eModemOutSignal = (msRTS, msDTR) ;

  TRxMessageEvent = procedure (Sender : TObject ; RxMessage : AnsiString) of object;
  TSignalChangeEvent = procedure (Sender : TObject ; Signal : eModemInSignal ; SignalState : Boolean) of
object;
  TWriteCompletedEvent = procedure of object;

  {--------------------------------------------------------------------------}
  { TSerial classe principal }

  TSerialTCC = class(TComponent)
  private
    { Declarações das propriedades }
    FPort : sPorts ;             { Propriedade porta }
    FBaudrate : Integer ;        { Propriedade Baudrate}
    FParity : eParity ;          { Propriedade paridade }
    FDataBits : eDataBits ;      { Propriedade Data bits }
    FActive : Boolean ;          { Propriedade Active }
    FRxQSize : Word ;            { Propriedade Input queue size property }
    FTxQSize : Word ;            { Propriedade Output queue size }
    FFlowMode : eFlowControl ;   { Flow control mode }
    FCheckParity : Boolean ;     { Check parity flag }
    FErrorCode : Integer ;       { Error code field }

    { Variáveis }
    FCID : THANDLE ;             { id das Com´s retornadas por "CreateFile" }
    FDCB : TDCB ;                { DCB para API do Windows }

    FCTO : TCOMMTIMEOUTS ;       { Estrutura dos timeout's das Com's }

    FTS  : TCOMSTAT;             { Estrutura que contém infomações sobre o estado das Com's }
    FMessageState : eMsgState;   { Mensagem estato dos flags }
    FMessageAppendLeft : Word;   { Mensagem overrun count }
    FMessage : AnsiString ;      { Mensagem capturada }
    FThreadActive : Boolean ;    { Flag do processo (Thread) ativo }
    CTSSignalState : Boolean ;   { Estado do sinal armazenado }
    DSRSignalState : Boolean ;   { Estado do sinal armazenado }
```

```
    RLSDSignalState : Boolean ;  { Estado do sinal armazenado }
    WrittenBytes : DWORD;        { Bytes escritos }

    OverlapBlock : TOverlapped;  { Contém informacao usada em entradas e saídas assíncronas }

    AbortInProgress : Boolean;

    { Ponteiros para os eventos/metodos }
    FOnRxData : TNotifyEvent;
//    FOnTxEmpty : TNotifyEvent;
//    FOnRing : TNotifyEvent;
//    FOnSignalChange : TSignalChangeEvent;
//    FOnWriteCompleted : TWriteCompletedEvent;

    { Apontador para o Thread }
    FEventThread : TThread ;

    { Procedures Privadas}
    procedure SetPort (nPort : sPorts);
    procedure SetBaud (Baud : Integer);
    procedure SetParity (Parity : eParity);
    procedure SetData (Data : eDatabits);
    procedure SetActive (bActive : Boolean);
    procedure SetRxQSize (qSize : Word);
    procedure SetTxQSize (qSize : Word);
    procedure SetFlowMode (mode : eFlowControl);
    procedure SetParityCheck (check : Boolean);

    function  GetRxWaiting : Integer;
    function  GetTxWaiting : Integer;

    procedure ReadErrorState;
    procedure SetupDCB;
    procedure ReOpenPort(toOpen : Boolean);
    procedure OpenErrorDlg (err : Integer);
    procedure EnableEvents;
    procedure ProcessError;

  protected
    { Protected declarations }

  public
    { Declarações públicas }
    constructor Create (aOwner : TComponent); override;
    destructor Destroy; override;

    procedure WriteChar (c : Char);
    procedure WriteString (s : AnsiString);
    function ReadChar (var c : Char) : Integer;
    function ReadString (var s : AnsiString) : Integer;
    procedure ZapTxQueue;
    procedure ZapRxQueue;
    procedure SetSignal (signal : eModemOutSignal; state : Boolean) ;

    { Propriedades públicas ( somente run-time ) }
    property Active : Boolean read FActive write SetActive default False ;

  published
    { Published declarations }
```

```
    { Propriedades publicadas }
    property Port : sPorts read FPort write SetPort default 1;
    property Baudrate : Integer read FBaudrate write SetBaud default 9600;
    property Parity : eParity read FParity write SetParity default paNone ;
    property DataBits : eDataBits read FDataBits write SetData default d7bit ;
    property RxQueueSize : Word read FRxQSize write SetRxQSize default 1024 ;
    property TxQueueSize : Word read FTxQSize write SetTxQSize default 1024 ;
    property FlowMode : eFlowControl read FFlowMode write SetFlowMode default fcNone ;
    property CheckParity : Boolean read FCheckParity write SetParityCheck default True ;
    property RxWaiting : Integer read GetRxWaiting ;
    property TxWaiting : Integer read GetTxWaiting ;
    property ErrorCode : Integer read FErrorCode ;

    { Propriedades Eventos/metodos }
    property OnRxData: TNotifyEvent read FOnRxData write FOnRxData;
//    property OnTxEmpty: TNotifyEvent read FOnTxEmpty write FOnTxEmpty;
//    property OnRing : TNotifyEvent read FOnRing write FOnRing;
//    property OnSignalChange : TSignalChangeEvent read FOnSignalChange write FOnSignalChange;
//    property OnWriteCompleted :TWriteCompletedEvent read FOnWriteCompleted write FOnWriteCompleted;
  end;

  {------------------------------------------------------------------------------}
  { SubClasse de controle dos Threads }
  TEventThread = class(TThread)
  private
    { Declarções privadas }
    FParentInstance : TSerialTCC ;
//    procedure DoTxEvent;
    procedure DoRxEvent;
//    procedure DoCTSEvent;
//    procedure DoDSREvent;
//    procedure DoRLSDEvent;
//    procedure DoRingEvent;

  protected
    procedure Execute; override;
  public
    constructor Create(parentTSerial : TSerialTCC);
  end;

procedure Register;

const
    aBaudrates : Array [1..15] of Integer =
              (110, 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 38400, 56000, 57600, 115200, 128000,
256000);

    aDatabits : Array[edataBits] of Byte =
          (7, 8) ;

    aParity : Array[eParity] of Byte =  (NOPARITY, ODDPARITY) ;

implementation

{------------------------------------------------------}
{ Registrando o componente! }
procedure Register;
begin
```

```
  RegisterComponents('System', [TSerialTCC]);
end;

{------------------------------------------------------}
{ Constructor/destructor procedures }
constructor TSerialTCC.Create (aOwner : TComponent);
begin
  inherited Create (aOwner);

  { Inicializando as propriedades default's }
  FPort := 1;
  FBaudrate := 9600;
  FParity := paNone ;
  FDataBits := d7bit ;
  FActive := False ;
  FRxQSize := 1024 ;
  FTxQSize := 1024 ;
  FCheckParity := True ;
  FMessageState := msNone ;
  FMessageAppendLeft := 0 ;

  CTSSignalState := False ;
  DSRSignalState := False ;
  RLSDSignalState := False ;


  OverlapBlock.Offset := 0;
  OverlapBlock.OffsetHigh := 0;
  OverlapBlock.Internal := 0;
  OverlapBlock.InternalHigh := 0;
  OverlapBlock.hEvent :=  CreateEvent(nil, FALSE, TRUE, pChar('Event1')) ;

  { Dispara o Thread object }
  FEventThread := TEventThread.Create(self) ;
  FThreadActive := False ;

  AbortInProgress := False ;

end;

destructor TSerialTCC.Destroy;
begin
  { Fecha a porta de esta estiver aberta no momento }
  ReopenPort (False);

  FEventThread.Terminate ; { Finaliza o thread }

  inherited Destroy; { chama destroy herdada }
end;

{------------------------------------------------------}
{ Procedures para acessar as propriedades }
procedure TSerialTCC.SetPort (nPort : sPorts);
begin
  if FPort <> nPort then
  begin
    FPort := nPort ;
    ReOpenPort(FActive);
  end
```

```
end;

{ Seta a velocidade de transmissão }
procedure TSerialTCC.SetBaud (Baud : Integer);
var
  i : Integer ;
  baud_ok : Boolean;
begin
  baud_ok := False;
  for i:=Low(aBaudrates) to High(aBaudrates) do
  begin
      if Baud=aBaudrates[i] then baud_ok := True ;
  end;

  if baud_ok then
  begin
    if FBaudrate<>Baud then
    begin
      FBaudrate := Baud ;
      ReopenPort(FActive)
    end
  end
  else
    MessageDlg ('Baudrate inválida!', mtError, [mbOK], 0)
end;

{ Seta pararidade }
procedure TSerialTCC.SetParity (Parity : eParity);
begin
  if FParity <> Parity then
  begin
    FParity := Parity ;
    ReopenPort(FActive)
  end
end;

{ Seta o número de data bits }
procedure TSerialTCC.SetData (Data : eDatabits);
begin
  if FDataBits <> Data then
  begin
      FDataBits := Data ;
      ReopenPort(FActive)
  end
end;

{ Abre a porta de comunicação }
procedure TSerialTCC.SetActive (bActive : Boolean);
begin
  ReOpenPort(bActive);
end;

{ Define o tamanho da fila de recepção }
procedure TSerialTCC.SetRxQSize (qSize : Word);
begin
  FRxQSize := qSize ;
  ReOpenPort(FActive);
end;
```

```
{ Define o tamanho da fila de transmissão }
procedure TSerialTCC.SetTxQSize (qSize : Word);
begin
  FTxQSize := qSize ;
  ReOpenPort(FActive);
end;

{ Seta o controle de fluxo (protocolo) RTS/CTS, DTR/DSR, XON/XOF, nenhum}
procedure TSerialTCC.SetFlowMode (mode : eFlowControl);
begin
  if FFlowMode <> mode then
  begin
    FFlowMode := mode ;
    ReopenPort(FActive)
  end
end;

{ Verificar a paridade? }
procedure TSerialTCC.SetParityCheck (check : Boolean);
begin
  if FCheckParity<>check then
  begin
    FCheckParity := check ;
    ReopenPort(FActive)
  end
end;

{ Bytes a esperando para serem lidos }
function TSerialTCC.GetRxWaiting : Integer;
begin
  if active then
  begin
    ReadErrorState;
    GetRxWaiting := FTS.cbInQue // FTS:TCOMStat
  end
  else
    GetRxWaiting := 0
end;

{ Bytes para serem transmitidos }
function TSerialTCC.GetTxWaiting : Integer;
begin
  if active then
  begin
    ReadErrorState;
    GetTxWaiting := FTS.cbOutQue
  end
  else
    GetTxWaiting := 0
end;


{------------------------------------------------------}
{ Internal procedures }
procedure TSerialTCC.ReadErrorState;
var
  errorcode : DWORD;
begin
  if active then
```

```
    if not ClearCommError (FCID,      // "handle" do dispositivo de comunicacao
                        errorcode, // Variavel para receber o erro
                        @FTS)      // Endereço do buffer do estado de comunicação
    then ProcessError
end;

procedure TSerialTCC.SetupDCB;
begin

  FDCB.DCBLength := SizeOf (TDCB) ;
  FDCB.BaudRate := FBaudRate ;
  FDCB.Flags := $0001;          { Binary flag }

  if FCheckParity then FDCB.Flags := FDCB.Flags or $0002;

  case FFlowMode of
    fcRTS_CTS: FDCB.Flags := FDCB.Flags or (RTS_CONTROL_HANDSHAKE shl 12) or $0004;
    fcDTR_DSR: FDCB.Flags := FDCB.Flags or (DTR_CONTROL_HANDSHAKE shl 4) or $0008;
    fcXON_XOF: FDCB.Flags := FDCB.Flags or $0300;
  end;

  FDCB.XonLim := FRxQSize div 3;               { Send XON when 1/3 full }
  FDCB.XoffLim := FRxQSize div 3;              { Send XOF when 2/3 full }
  FDCB.ByteSize := aDatabits[FDataBits] ;
  FDCB.Parity := aParity[FParity] ;
  FDCB.StopBits := ONESTOPBIT; // Balanca e impressora usam 1 stopbit

  FDCB.XonChar := Chr(17);
  FDCB.XoffChar := Chr(19);
  FDCB.ErrorChar := '?';

  { Seta os parametros do dbc na porta }
  if FActive then
  begin
    if not SetCommState(FCID, // "handle" do dispositivo de comunicacao (CreateFile)
                    FDCB) // endereco da estrutura de controle
    then ProcessError;

    { Set comms timeouts }
    FCTO.ReadIntervalTimeout := MAXDWORD ; { Non-blocking read }
    FCTO.ReadTotalTimeoutMultiplier := 0 ;
    FCTO.ReadTotalTimeoutConstant := 0 ;

    { Block writes only if flow control is active }
    if FFlowMode <> fcNone then
    begin
      FCTO.WriteTotalTimeoutMultiplier := 0 ;
      FCTO.WriteTotalTimeoutConstant := 10000 ;
    end;
    // SetCommTimeOuts seta os parâmetros de time-out para todas as
    // operações de escrita e leitura no dispositivo de comunicação
    if not SetCommTimeouts (FCID, // "Handle"
                        FCTO) // Estrutura COMMTimeOuts usada para determinar
                            // o comportamento de ReadFile, WriteFile
    then ProcessError
  end
end;

//Abertura da porta de comunicação
```

```
procedure TSerialTCC.ReOpenPort(toOpen : Boolean);
var
  pname : Array[0..10] of char;
  ret : Integer;
  count : Integer ;
begin
  if FActive then
  begin
      { Porta ativa entao fecha }
      { Limpa a fila de TX e RX }
      //PurgeComm discarta todos os caracteres do buffer de output e input de disp. de com.
      if not PurgeComm (FCID, // "Handle"
                  PURGE_TXABORT or PURGE_RXABORT) // Limpa o buffer de entrada e saida
      then ProcessError;

      //Desabilita os sinais DTR e RTS
      SetSignal (msDTR, False) ;
      SetSignal (msRTS, False) ;

      // Propriedade "Ativa" falsa
      FActive := False;


      // Especifica conjunto de eventos a ser monitorado no disp de com.
      if not SetCommMask (FCID, 0) then // nenhum evento
      ProcessError;

      { espera pelo thread acabar }
      count := 100 ;
      while FThreadActive and (count > 0) do
      begin
        Sleep(75) ;
        count := count-1 ;
      end;



      // Fecha o "Handle" , desaloca o device
      if not CloseHandle(FCID)
      then ProcessError;

  end;

  if (toOpen) then
  begin
    { Abre a porta usando o parâmetros especificados }
    StrPcopy(pname, Format('COM%d', [FPort]));
    ret := CreateFile (pname, (GENERIC_READ or GENERIC_WRITE), 0, nil,
                  OPEN_EXISTING, FILE_FLAG_OVERLAPPED, 0) ;


    if ret <> INVALID_HANDLE_VALUE then
    begin
       { Guarda o handle }
       FCID := ret;

       { seta os tamanhos da comm }
       if not SetupComm (FCID, FRxQSize, FTxQSize)
       then ProcessError;
```

```
        { define os pontos de interrupção  xon/xof e seta as opções do DCB }
        FActive := True;
        SetupDCB;

        { Ativa DTR & RTS por default }
        SetSignal(msDTR, TRUE);
        setSignal(msRTS, TRUE);

        { dispara o processamento de eventos }
        EnableEvents;
      end
      else ProcessError;
    end
end;

procedure TSerialTCC.EnableEvents;
begin
  if not SetCommMask (FCID, EV_RXCHAR or EV_TXEMPTY or EV_CTS or EV_DSR or
     EV_RLSD or EV_RING)
  then ProcessError;
  FEventThread.Resume ;           { Resume event thread }
end;


procedure TSerialTCC.OpenErrorDlg (err : Integer);
begin
  { Locate message }
  MessageDlg (SysErrorMessage(err), mtError, [mbOK], 0)
end;

{ General error handler }
procedure TSerialTCC.ProcessError;
begin
  FErrorCode := GetLastError() ;

end;


{-------------------------------------------------------}
{ Method procedures }
procedure TSerialTCC.WriteChar (c : char);
var
  p : array[0..1] of char ;
  Res : Integer;
begin
  StrPCopy(p,c) ;
  if not WriteFile (FCID, p, 1, WrittenBytes, @OverlapBlock) then
  begin
    Res := WaitForSingleObject(OverlapBlock.hEvent,1000);
    if  Res <> 0 then ProcessError
    else
    begin
      ResetEvent(OverlapBlock.hEvent);
      OverlapBlock.Offset := 0;
      OverlapBlock.OffsetHigh := 0;
    end;
  end;
end;
```

```
procedure TSerialTCC.WriteString (s : AnsiString);
var
  Res, timeout : Integer;
begin
  { Timeout based on string length & baudrate * 2 }
  timeout := (20000 div baudrate) * Length(s)  ;
  if timeout < 500 then timeout := 500 ;

  if not WriteFile (FCID, s[1], Length(s), WrittenBytes, @OverlapBlock) then
  begin
    Res := WaitForSingleObject(OverlapBlock.hEvent,timeout);
    if  Res <> 0 then    ProcessError
    else
    begin
      ResetEvent(OverlapBlock.hEvent);
      OverlapBlock.Offset := 0;
      OverlapBlock.OffsetHigh := 0;
    end;
  end;
end;

function TSerialTCC.ReadChar (var c: Char) : Integer;
var
  buf : Array[0..1] of Char ;
  Res : integer;
begin
  ReadChar := 0;
  if not ReadFile (FCID, buf, 1, WrittenBytes, @OverlapBlock) then
  begin
    Res := WaitForSingleObject(OverlapBlock.hEvent,100);
    case Res of
      WAIT_OBJECT_0 :
      begin
        ResetEvent(OverlapBlock.hEvent);
        OverlapBlock.Offset := 0;
        OverlapBlock.OffsetHigh := 0;
        if (WrittenBytes=1) then c := buf[0] ;
        ReadChar := WrittenBytes;
      end;
      WAIT_TIMEOUT :
      begin
        ResetEvent(OverlapBlock.hEvent);
        OverlapBlock.Offset := 0;
        OverlapBlock.OffsetHigh := 0;
        WrittenBytes := 0;
        ReadChar := 0;
      end;
      else
      begin
        ProcessError;
        exit;
      end;
    end;
  end
  else
  begin
    ResetEvent(OverlapBlock.hEvent);
    OverlapBlock.Offset := 0;
```

```
    OverlapBlock.OffsetHigh := 0;
    if (WrittenBytes=1) then c := buf[0] ;
    ReadChar := WrittenBytes;
  end;
end;

function TSerialTCC.ReadString (var s : AnsiString) : Integer;
var
  TestChar : char;
  i : integer;
  iCount : integer;
begin
  S := '';
  for i := 0 to FRXQSize do
  begin
    if self.ReadChar(TestChar) > 0 then S := S + TestChar
    else System.Break;
  end;
  iCount := Length(s);
  Result := iCount;
end;

procedure TSerialTCC.ZapTxQueue;
begin
   if not PurgeComm (FCID, PURGE_TXABORT)
   then ProcessError
end;

procedure TSerialTCC.ZapRxQueue;
begin
   if not PurgeComm (FCID, PURGE_RXABORT)
   then ProcessError
end;

procedure TSerialTCC.SetSignal (signal : eModemOutSignal; state : Boolean);
var
    func : DWORD;
begin
    func := 0;
    if signal=msRTS then
        if state then func := SETRTS
                 else func := CLRRTS
    else if signal=msDTR then
        if state then func := SETDTR
                 else func := CLRDTR;

    EscapeCommFunction (FCID, func) ;
end;


{------------------------------------------------------------------------------}
{ Procedure para os eventos }

{
  Procedures principais do Thread
  Espera pelo evento e chama a procedure do usuário;
  "Synchronize" permite que o evento do usuario acesse as
  procedures VCL do thread.
}
```

```
procedure TEventThread.Execute;
var
  event : DWORD ;
  cstate : DWORD ;
  c      : Char ;
begin
  with FParentInstance do

    while not Terminated do
    begin
      if Active then
      begin
        { Seta a flag de ativo }
        FThreadActive := True ;

        { Espera pelo evento das comms }
        if not WaitCommEvent (FCID, event, nil)
        then
        begin
            if Active then Synchronize(ProcessError);
        end;



          if (event and EV_RXCHAR) <> 0 then
             { Execute OnRxData event }
             Synchronize(DoRxEvent);

        end
      else  { Not active }
      begin
        { unset active flag }
        FThreadActive := False ;
        Sleep(100) ;
      end;
    end;
  end;
end;

{ Constructor for thread object }
constructor TEventThread.Create(parentTSerial : TSerialTCC);
begin
  inherited Create(True);                { Create the thread suspended }
  FreeOnTerminate := True ;
  FParentInstance := parentTSerial ;
end;

procedure TEventThread.DoRxEvent;
begin
   { Execute OnRxData event }
   with FParentInstance do
   if Assigned(FOnRxData) then FOnRxData(FParentInstance);
end;

end.
```