Aprovado 9,5
21/01/2002

ALEXANDRE J. VEZZA CAMPOS
ANSELMO LAURINI SANT'ANNA
ROGER G. RODRIGUES

604

MONOGRAFIA MBA-USP

GESTÃO PORTUÁRIA

AUTOMAÇÃO DO TERMINAL PARA FERTILIZANTES

Orient.: Eduardo Macioios

SANTOS
2.001

Sumário

Lista de Tabelas

Lista de Figuras

Lista de Fluxogramas e Diagramas do modelo proposto de automação

Lista de Abreviaturas e Símbolos

m^2	Metro quadrado
“	Polegadas
m	Metro
tons	Tonelagens
hr.	Hora
Arm.	Armazém
Gte’s	Guindastes
hrs	Horas
Ogmo	Orgão Gestor de Mão de Obra
E.P.I.’s	Equipamentos de Proteção Individual
%	Porcentagem
°C	Grau
BL	Bill of Lading
DI’s	Documentos de Importação
un	Unidade
D.R.F.	Departamento da Receita Federal
n.° s	Números
HP	Horse Power
m^3	Metros cúbicos
CD	Compact Disc
Mb/s	Megabytes por segundo
ms	Microssegundos
I/O	Input/Output
RAM	Random Access Memory
kHz	Quilo Hertz
MHz	Mega Hertz
v	Volts
Ω	Omega

Capitulo I

I.I. **Introdução: Terminal de Fertilizante – TEFER**
I.II. **Terminais de Granel Sólido no Brasil e no Mundo**
I.III. **Objetivos: Modelo de automação do TEFER**

Capitulo II

II.I. **Modelo atual de funcionamento**
II.II. **Codesp – TEFER – Lay out do fluxo documental (modelo de informação)**

Capitulo III

III.I **Descrição dos equipamentos, nível de automação nos equipamentos e o supervisório administrativo.**
III.II. **Descrever tecnologias do mercado**

Capitulo IV

IV.I. **Modelo proposto**

Capitulo V

V.I. **Conclusão**

Bibliografia

Capitulo I

I.I. Introdução: Terminal de Fertilizantes - TEFER

Fig.. 1 – Terminal de Fertilizantes - TEFER

a) Histórico:

- Terminal para Fertilizantes - TEFER, foi inaugurado em 1971, em uma área de aproximadamente 300.000 m^2, localizado na Margem Esquerda do Porto de Santos, no município do Guarujá, distrito de Vicente de Carvalho - Conceiçãozinha.

b) Acessos:

- Rodoviários (com pátio para caminhões);
- Ferroviário para embarque em todos os Armazéns, para bitolas largas e estreitas.

c) Piers:

- 02 Piers acostáveis com 283,50 metros (cada).

d) Calado permitido (pés):

- TEFER1 - 37' 00" NAABSA BW;
- TEFER2 - 39' 00" NAABSA BW.

e) Calado Aéreo:

- TEFER1 - 17 metros;
- TEFER2 - 17 metros.

f) Equipamentos:

- 06 Armazéns com capacidade de estocagem de 30.000 tons/ Arm, equipados com esteiras automáticas com Trippers;
- 9.900 metros, de esteiras transportadoras, nos piers e no interior dos Armazéns, sendo, no Pier 1, instaladas esteiras de 24" com capacidade nominal de 1.200 tons/hr; e no Pier 2, esteiras de 42" com capacidade nominal de 1.200 tons/hr;
- 11 Guindastes (elétricos) de terra equipados com Grab's automáticos de 3/5 tons, e capacidade de 300 tons/hr;
- 04 Balanças Rodoviárias - para 80 tons, interligadas e equipadas com programas de gerenciamento de pesagens;
- 02 balanças Ferroviárias para 150 tons, ligadas ao terminal de pesagem. 12 Pás Carregadeiras para 05 tons;
- 02 Pás Carregadeiras para 2,0 tons;
- 02 empilhadeiras para 3,0 tons;
- 02 empilhadeiras 7,0 tons;
- 01 empilhadeira 10,0 tons;
- 02 Tratores para tracionamento de vagões;
- 01 Caminhão VW trucado equipado com Munk de 16 tons;
- 01 Cavalo e Carreta para 25 tons;
- 01 Caminhão (Toco) para serviços gerais.

Movimento de importação de adubos no TEFER, foi o seguinte:

- 1995- 1.747.306 tons;
- 1996- 1.811.274 tons;
- 1997- 1.782.813 tons;
- 1998- 1.438.188 tons;
- 1999- 1.555.504 tons;
- 2000- 1.947.963 tons.

Histórico Fertimport/ TEFER:

- Com a saída da Codesp das Operações do Porto, transformando-se em Autoridade Portuária do Porto de Santos; a Fertimport foi credenciada a operar o TEFER, assumindo as operações e manutenção do mesmo até a finalização da licitação;
- Sendo assim, em 03/08/99, a Fertimport assumiu as operações do TEFER, buscando como objetivo principal entregar a Safra de 1999, pois o terminal encontrava-se em estado precário, por falta de manutenção;

7804-2649.

- Paralelamente, a Codesp assina junto a Cetesb, o TAC, termo de Alteração de Conduta Ambiental, e a Fertimport assume parte das responsabilidades destas implantações, em até 18 meses.

As ações corretivas são as seguintes:

- Implantação de cinturão Verde ai redor do TEFER;
- Implantação dos Raspadores nas esteiras e GTE´s do Pier 2;
- Organização do fluxo de veículos;
- Colocação de anteparos nos GTE´s , para não permitir a quede de produtos no estuário;
- Murar o Pier 2, e instalar caixas de contenção de produtos, para permitir a lavagem do Pier sem contaminar o estuário;
- Tratamento de águas residuais no pátio de enxofre;
- A critério conjunto dos - operadores do terminal e fiscalização Codesp, paralisar as operações de descarga de enxofre, tão logo seja perceptível no ar, odor característico de gás sulfídrico (H_2S), até o retorno de condições favoráveis para dispersão dos gases na atmosfera;
- Limpeza das áreas e equipamentos utilizados na movimentação de enxofre imediatamente após o término da descarga do navio;
- Proteção das pilhas de enxofre, a fim de minimizar o arraste do produto pelos ventos;
- Implantação de sistema de carregamento do enxofre em veículos, provido de equipamento de controle de poluentes (semienciausuramento, nebulizadores, etc.).

Operacionalmente foram tomadas as seguintes providências :

- Entrega de produtos aos importadores, somente nos períodos : 7:00 às 13:00 hrs e das 13:00 às 19:00 hrs;
- Implantação de sinalização de trafego no terminal, com placas sinalizadoras e determinação de faixas de circulação e filas dos autos;
- Construção de nova portaria com áreas definidas de circulação p/ tipo de veículos e pessoas;
- Implantação de pás carregadeiras com maior capacidade de carregamento, de 3,5 para 5 tons, objetivando maior produtividade para Armazém;
- Implantação de rotinas de limpeza das vias permanentes (ruas e trilhos), e nos Armazéns, com máquinas varredeiras e mini-carregadeiras;
- Nos Piers; lavagens dos equipamentos (Guindastes e esteiras), objetivando a não contaminação dos produtos descarregados, não gerando mais "raspa".

Estas modificações Operacionais nos levaram a atingir recordes nas movimentações históricas do TEFER, como segue:

- Carregamento de 369 autos com 10.486,900 tons em 27/10/99, em 02 períodos contra 369 autos com 9.537,480 tons feitos por Codesp - 03/09/97 em 03 períodos.

- Descarga de navios :
  - TEFER 2- 13.000,000 tons, em 24 hrs, em Out. 2000, contra 9.567,359 tons em 1998 – produto - enxofre;
  - TEFER 1- 8.251,800 tons, em 24 hrs, em 1999, contra, 8.023,900 tons, em 1994 – produto - fertilizantes.
- Carregamento de vagões em 01 mês - 38.506,000 tons em Novembro/2000, contra 34.338,000 tons em Novembro/1995.

Conscientização dos trabalhadores:

- A mão de obra disponível no Porto de Santos, conforme a lei 8.630, obrigatoriamente é requisitada via Ogmo (estivadores, conferentes, operadores de pás carregadeiras e Guindastes etc...), funcionários da Codesp, e da Força Supletiva do Sintraport, não tem compromissos com resultados assumidos pela Codesp e Fertimport, sendo necessário um trabalho de conscientização com muito treinamento;
- Estão sendo implantados, usos de uniformes, EPI´s, cursos de especialização para pessoal técnico, bem como conscientização das atividades que possam vir a poluir o Meio Ambiente (evitar queda de produtos no Mar nas operações de descarga e ou limpeza do Pier, reaproveitamento e recondicionamento de óleos lubrificantes, trapos etc...).

I.II. Terminais de Granel Sólido no Brasil e no Mundo

I.II.I. Terminais no Brasil - Terminal da Ultrafertil - TUF

Fig.. 2 – Terminal Marítimo da Ulttrafértil - TUF

I.II.I.I. Terminal Marítimo da Ultrafértil - TUF

- Terminal Portuário de uso privativo misto, fora do porto organizado de Santos e alfandegado pela Secretaria da Receita Federal.
- Localização: Ilha do Cardoso, em Santos, São Paulo, em área 545.411m^2.
- Descrição: tem 164 metros de pier acostável, com capacidade para atracar navios com até 229 metros de comprimento, 33 metros de boca, calado máximo de 11 metros (36 pés) e 40.000 toneladas de carga.
- Capacidade de recebimento de amônia: 500 toneladas/hora
- Sistema de descarga de granéis sólidos: 10.000 toneladas/dia.
- Capacidade de Armazenagem de Amônia: 20.000 toneladas a –33° C.
- Capacidade de Armazenagem de Fertilizantes: 60.000 toneladas.
- Capacidade de Armazenagem a Céu Aberto: 60.000 toneladas.

Fig.. 3 – Fábrica da Ultrafertil - TUF

a) Histórico:

- A Ultrafertil foi constituída em 1965, com a participação da Philips/PS Petroleum e do Grupo Ultra além de entidades financeiras internacionais.
- Em maio de 1974 a Petrobrás adquiriu o controle e o manteve até o leilão de desestatização ocorrido em junho de 1993, quando o seu controle acionário foi adquirido pela Fosfertil.
- Em novembro de 1995, visando a simplificação administrativa e minimizar a carga tributária se deu a incorporação pela Goiasfertil S.A., empresa controlada integralmente pela Fosfertil e dedicada a extração de rocha fosfática em Catalão-Goiás.
- Após a incorporação a Goiás Fertilizantes S.A. - Goiasfertil assumiu a razão social Ultrafertil S.A.
- A Fosfertil é detentora de 99,99% do capital social da Empresa.

I.II.I.II. Meio Ambiente

Um dos objetivos da Ultrafertil é estar em sintonia com a meio ambiente, assim desenvolveu uma Política de Preservação do Meio Ambiente em consonância com as mais modernas tecnologias disponíveis no mercado em acordo com os padrões estabelecidos pelos órgãos de controle ambiental. A Ultrafertil possui áreas especializadas para a proteção do meio ambiente em todos os seus complexos industriais e de mineração, assim como no terminal marítimo.

Por intermédio de um planejamento ordenado de ações, são investidos milhares de dólares anualmente no tratamento de efluentes líquidos industriais; abatimento de gases nitrosos com peróxido de hidrogênio; controle e monitoramento de acidez da estação de tratamento; drenagem, tratamento e coleta de sólidos de efluentes do terminal marítimo, entre outros. A automação industrial também tem papel de destaque na preservação do meio ambiente, já que permite a modernização dos sistemas de controle de processo das unidades produtivas. Utilizando-se de instrumentação eletrônica digital, reestudam os riscos dos respectivos processos e intertravamentos de segurança, com significativo aumento da confiabilidade do controle operacional da produção. Com novas tecnologias e modernos equipamentos, as empresas aperfeiçoam a qualidade dos produtos, ganham maior produtividade e preservam o homem e o meio ambiente.

Políticas de Segurança, Saúde e Meio Ambiente.

a) Proteção Ambiental

- Realizar avaliação prévia de aspectos ambientais em novas atividades, bem como minimizar a geração de resíduos, efluentes e emissões através do contínuo aperfeiçoamento dos processos e de práticas que possibilitem sua reciclagem, reutilização e disposição final adequada, de maneira a prevenir danos ao meio ambiente e à comunidade.

b) Segurança e Saúde no Trabalho

- Transmitir e fazer cumprir medidas preventivas de segurança e saúde inerentes às suas atividades.

c) Segurança do Processo

- Analisar e prevenir os riscos potenciais envolvidos nas operações existentes, nas modificações de instalações, equipamentos, condições operacionais e na desativação de plantas de processos.

d) Transporte e Distribuição

- Manter as atividades de transporte e distribuição executadas por empresas previamente qualificadas pela Ultrafertil ou por ela validadas, comprometidas com as normas de segurança vigentes.

e) Gerenciamento do Produto

- Gerenciar os produtos fornecidos a seus clientes, orientando quanto às condições de armazenamento e manuseio, evitando-se riscos ao meio ambiente e ao bem estar da comunidade.

f) Diálogo com a Comunidade e Preparação e Atendimento a Emergências

- Manter diálogo com a comunidade sobre suas atividades e seus produtos, bem como preparativos de atendimento a emergências.

I.II.II. Terminais de Granel Sólido no Mundo

Podemos citar nas pesquisas feitas em visitas de pessoal técnico, em terminais de Graneis Europeus (EBS Bulk, em Rotterdam - Netherlands; e Terminal Verbrugge, no Porto de Terneuzen, na Holanda) diferentes tipos de equipamentos, logísticas e até a venda do serviço com procedimentos totalmente diferentes dos utilizados atualmente no Brasil..

Acreditamos que as diferenças que apontaremos abaixo, deverão deixar de existir assim que a iniciativa privada assuma os terminais.

Sendo assim, comentamos alguns dos pontos observados:

❖ Maiores informações sobre o Terminal Verbrugge vide página 68.

I.II.II.I. Comercialmente: Oferecer Pacotes de serviços aos clientes

Os serviços oferecidos são apresentados em pacotes de serviços, para atuarem com Importação e Exportação/ Carga e Descarga, inclusive com produtos diferenciados (Granel, Produtos Florestais e Carga SECA, por exemplo em Verbrugge).

Produtos oferecidos:

a) Atividades Portuárias - Operação Portuária, Armazenagem, Agenciamento e Afretamentos de outros modais (rodoviário, ferroviário e aquaviário).

b) Suportes – Disponibilidade on line oferecida ao Cliente como:

- Disponibilidade de dados via EDI;
- Utilização de código de barras, com transmissão de dados via Rádio freqüência.

c) Transportes:

- Comunicação móvel via rádio com os caminhões;
- Computadores de bordo;
- Auto identificadores.

I.II.II.II. Número de funcionários

Face ao nível de automação dos equipamentos instalados no Terminal de Terneuzen, somadas as leis trabalhistas locais, um terminal que opera 3 milhões de tons/ano para Granel, entre Importação e Exportação, opera com a seguinte equipe de funcionários:

- 06 operadores de Guindastes;
- 03 operadores de shiploaders;
- 11 operadores;
- 04 supervisores de turmas;
- 04 Coordenadores (média gerencia);
- 03 ADM/ Controladores;
- 05 suporte técnicos (elétrica e mecânica);
- Total - 36 funcionários.

Destacar o profissionalismo e o nível de educação.

I.II.II.III. Operações Logísticas e Equipamentos:

Os equipamentos disponíveis para as operações portuárias, são disponibilizados de formas mais flexíveis e eficientes, como por exemplo, em \Rotterdam são oferecidos Guindastes instalados em Piers normais e em flutuantes com torres de pesagem, que agilizam a descarga do navio.

Outra proposta, é a de "tetos removíveis" nos armazéns de descarga, e descarregadores instalados nesta área, Pier / Armazém, dispensando as esteiras no trajeto. Os guindastes de "giro" descarregam diretamente no Armazém Shoebox, sendo flexibilidade a palavra de ordem nos terminais visitados, está também foi observada nos sistemas de armazenagem. Os terminais devem possuir estruturas horizontais (armazéns para todo tipo de carga) e silos (para cargas do tipo "free flow"). Inovações como o tijolo "lego", usados para dividir áreas dos armazéns, sendo rapidamente montados e desmontados, armazéns do tipo "Shoebox" com o teto removível e dispensam o uso de moegas e esteiras, são inovações importantes.

As áreas onde esta instalado o terminal, estão interligadas e conectadas a auto estradas, ferrovias.

Os equipamentos disponíveis nos terminais citados são flexíveis, permitindo que os mais diversos tipos de graneis sólidos sejam movimentados. Um equipamento do

tipo sugador pneumático limitaria as opções, já que este é próprio para trigo e assemelhados. Evidentemente, em virtude do porte e do histórico dos terminais visitados, alguns equipamentos especializados foram observados, mas estes são minoria.

Essa flexibilidade é alcançada com o uso de equipamentos com grab (de vários tamanhos, adequados a cada peso específico), que realizam o chamado movimento de quadratura, evitando o giro. Os equipamentos do tipo "pórtico" (GRAB GANTRY CRANE) e "Kangaroo" foram os que mais se destacaram, os equipamentos do tipo pórtico são montados no cais, enquanto os Kangoroos podem ser montados em flutuantes. Equipamentos do tipo guindastes com giro montados em flutuantes foram observados, mas principalmente para a descarga em moegas equipadas com balança, as quais descarregam para barcaças.

Informação: Chave da Operação

Observou-se que a chave para a eficiente operação do terminal é a informação. A sala de controle no terminal E.B.S., vide figuras 14 e 15 que ilustram este fato. Todas as operações podem ser acompanhadas lá através de sensoreamento remoto, controlando o fluxo que passa por cada uma das esteiras e balanças de fluxo.

Importante também foi a observação ao trato com o meio ambiente. É reconhecido que a legislação ambiental brasileira é uma das mais severas do mundo. Comparando-se um Terminal como o E.B.S., vide figuras 4, 5, 6 e 7, com um Terminal moderno como o da Ultrafértil, vide figura 2, nota-se neste último uma acentuada vantagem na limpeza, inclusive dispondo de muito mais equipamentos de proteção do que o primeiro. O Terminal Verbrugge, vide figuras 10 e 11, no entanto, mesmo dispondo de equipamentos como a proteção anti-poeiramento e sendo consideravelmente moderno e eficiente, não se equipará em proteção ambiental ao da Ultrafértil.

Evidentemente, a cultura / formação (que se está tentando implantar gradativamente na mão-de-obra portuária brasileira) do "mantenha limpo evitando sujar" é responsável por essa "não necessidade" de Leis severas / terminal eficiente eficiente / termina limpo.

Fotos dos terminais acima citados como exemplos de atualidades no mundo.

Fig.. 4 – Visão do Terminal E.B.S de Rotterdam

Fig.. 5 – Visão do Terminal E.B.S de Rotterdam

Fig.. 6 – Visão do Terminal 2 E.B.S de Rotterdam

Fig. 7 – Visão do Terminal E.B.S de Rotterdam

Fig. 8 – Visão do Terminal de Le Havre - Bulks

Fig.. 9 – Visão do Terminal de Le Havre

Fig. 10 – Visão do Terminal de Verbrugge

Fig. 11 – Visão do Terminal de Verbrugge, no Porto de Terneuzem

Fotos de demonstrativos operacionais dos terminais acima citados como exemplos de atualidades no mundo.

Fig.. 12 – Visão da separação do produto por um sugador de grande porte – Terminal E.B.S.

Fig.. 13 – Visão da separação do produto por um sugador de grande porte – Terminal E.B.S.

Fig.. 14 – Visão da Sala de Controle do Terminal E.B.S., em Rotterdam

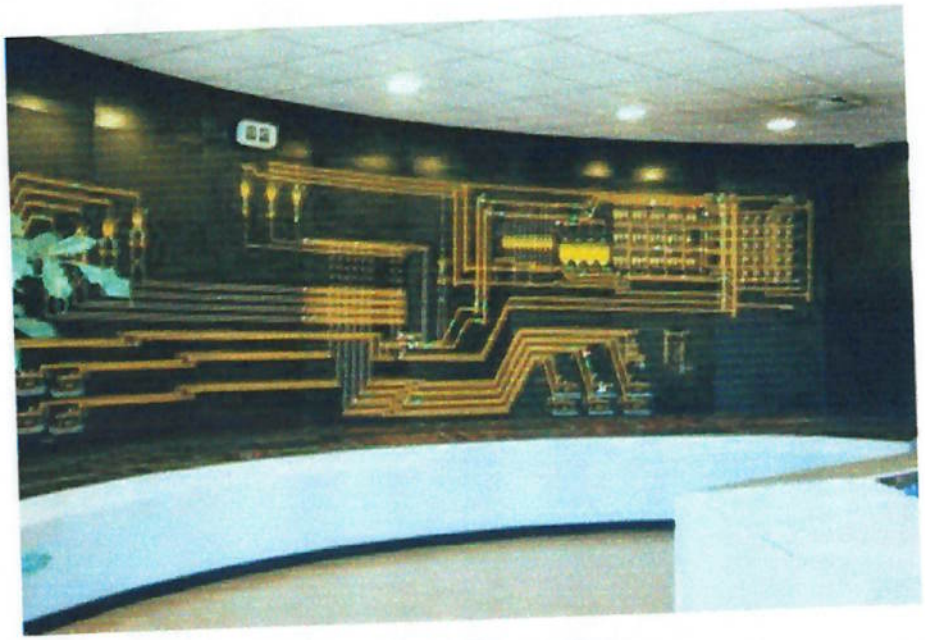


Fig.. 15 – Visão da Sala de Controle do Terminal E.B.S., em Rotterdam

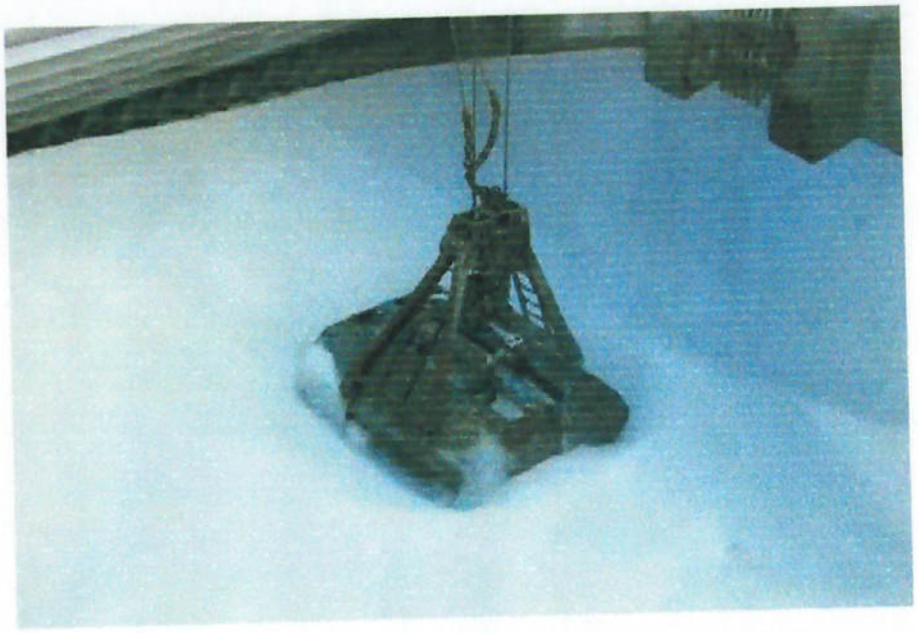


Fig.. 16 – Visão do Grab operando no porão de carga – Terminal E.B.S.

Fig.. 17 – Visão do Grab operando no porão de carga – Terminal E.B.S.

Fig.. 18 – Visão do Grab operando no porão de carga – Terminal E.B.S.

Fig.. 19 – Visão da operação de Guindastes com grab (36 tons) – Terminal E.B.S.

Fig.. 20 – Visão da operação de Guindastes de 36 tons. – Terminal E.B.S.

Fig.. 21 – Visão do Terminal Europort, em Rotterdam

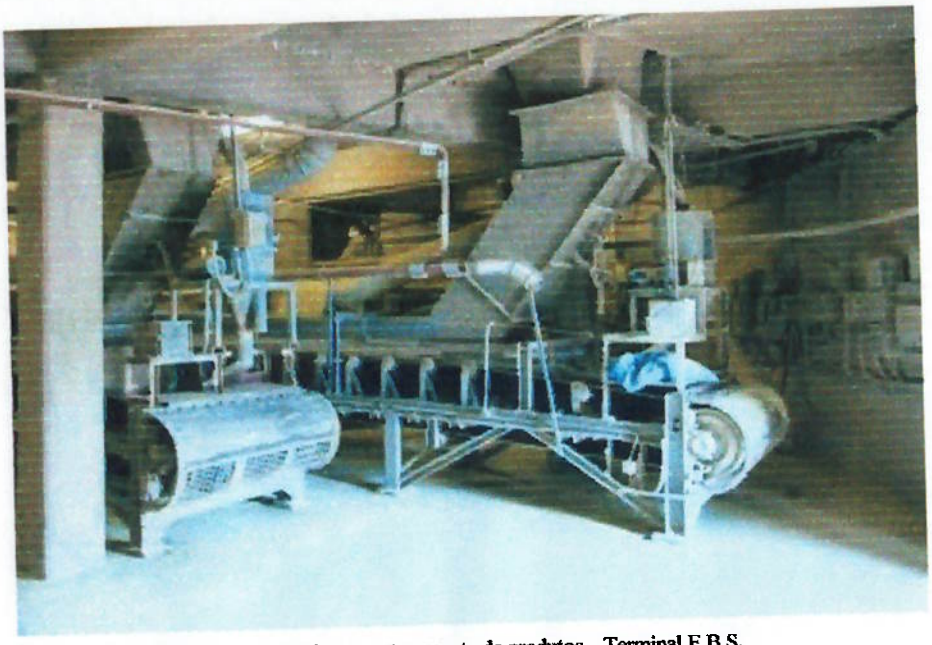


Fig.. 22 – Visão do sistema de esteiras para transporte de produtos – Terminal E.B.S.

Fig.. 23 – Visão da máquina de rechego no porão do navio – Terminal E.B.S.

Fig.. 24 – Visão do Grab operando no porão de carga – Terminal E.B.S.

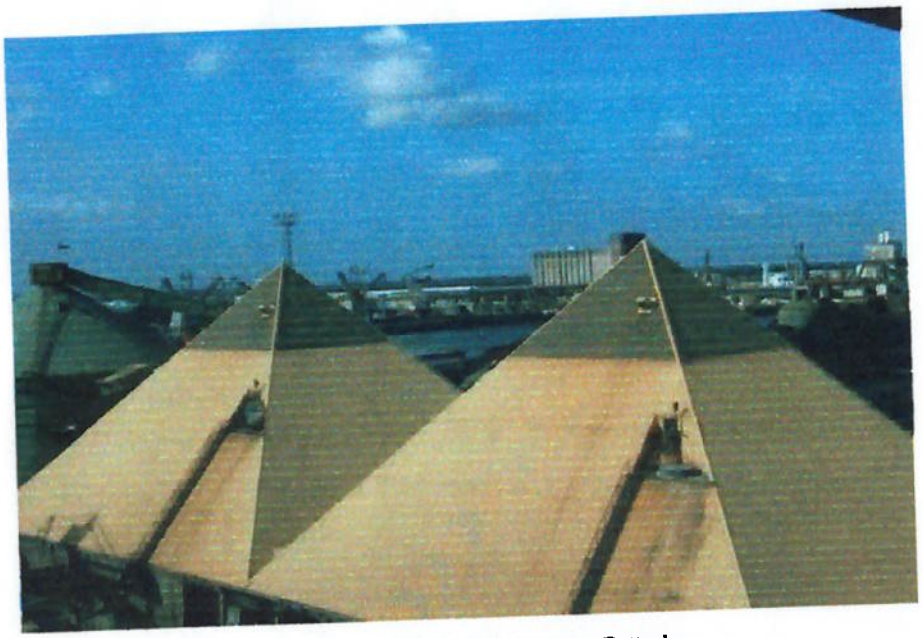


Fig.. 25 – Silo de armazenagem no Terminal de Laurenshaven, em Rotterdam

Fig.. 26 – Visão do descarregamento de grãos com equipamento pneumático – Terminal E.B.S.

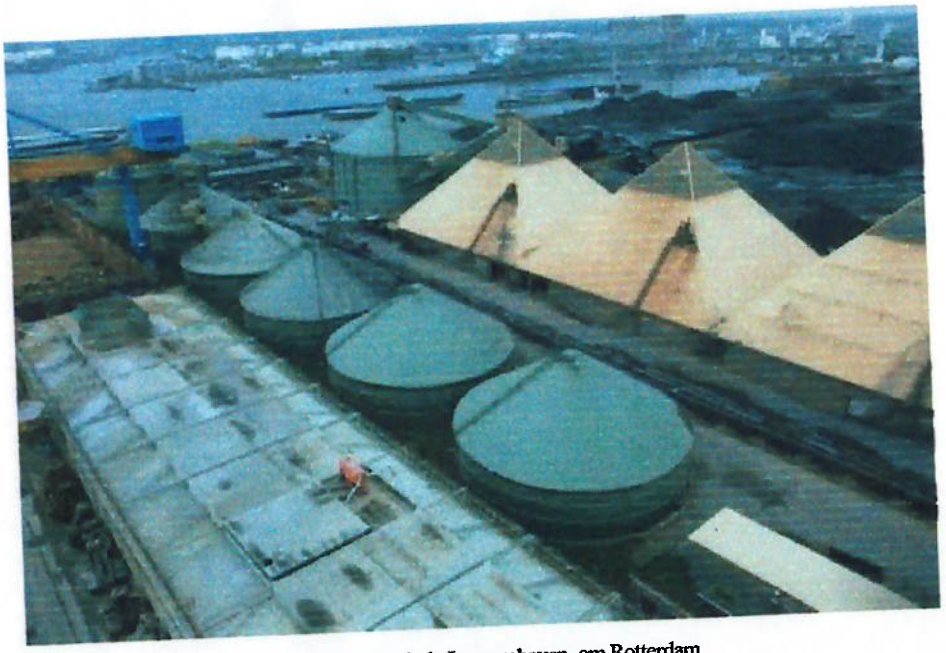


Fig.. 27 – Silo de armazenagem no Terminal de Laurenshaven, em Rotterdam

Fig.. 28 – Visão da máquina de rechego no porão do navio – Terminal E.B.S.

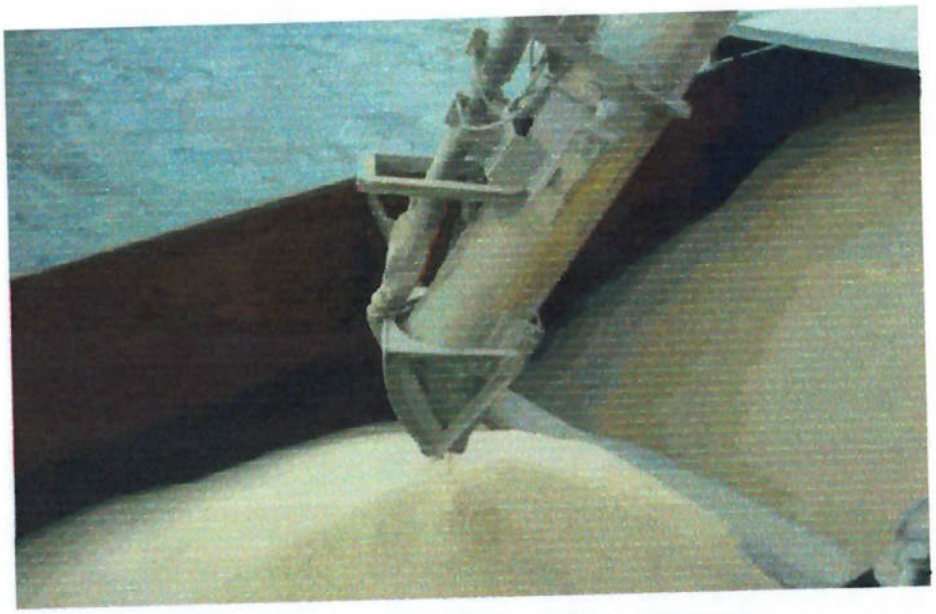


Fig.. 29 – Visão do carregamento de grãos em barcaça com proteção anti-poeiramento

Fig.. 30 – Ponte de Balança de Fluxo

Mapa de localização do Porto de Terneuzem e do Porto de Santos

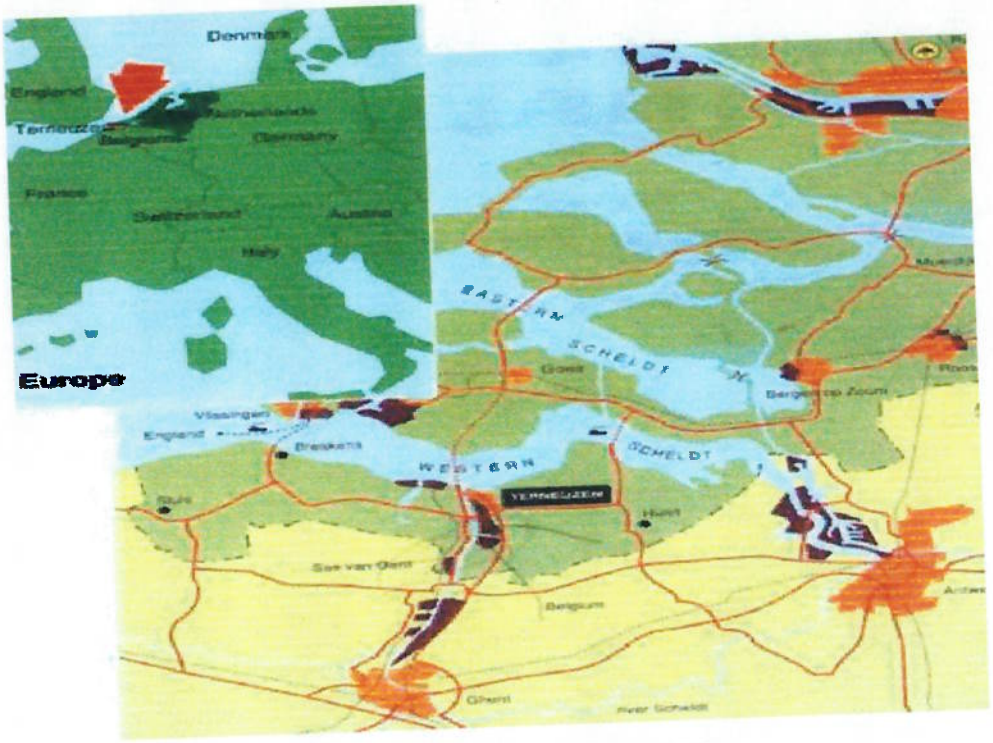


Fig.. 31 – Mapa de localização do Terminal de Verbrugge, no Porto de Terneuzem

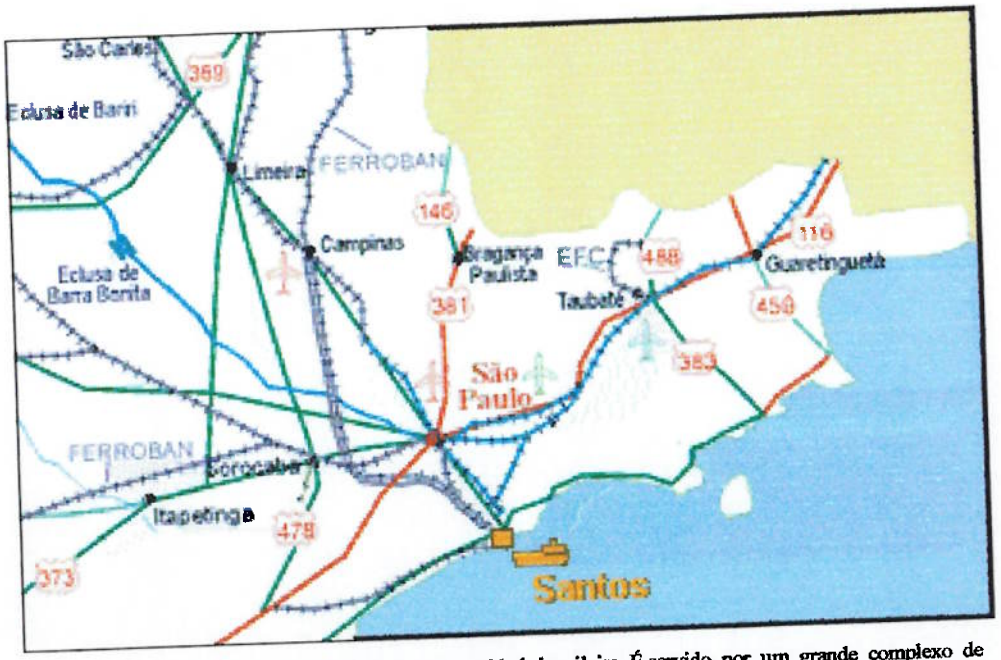


Fig.. 32 – Santos dista 65 km de São Paulo, a maior cidade brasileira. É servido por um grande complexo de transporte e, num raio de 100 km, dois aeroportos internacionais complementam a intermodalidade

I.III. Objetivos – Modelo de automação do TEFER

Buscamos apresentar neste trabalho, um projeto modular de automação de terminais de Graneis (especificamente - Fertilizantes), que opera a descarga do produto por correias transportadoras, gerando massa de dados para ferramenta gerencial.

Atividade de projetar, implantar e manter uma automação, seja ela para apenas uma máquina ou para uma planta de fábrica, ou terminal portuário, com várias maquinas e motores em operações sincronizadas, encontra dificuldades pela necessidade de atender à Interoperabilidade entre os circuitos operantes, sensores, ou instrumentos de medição a serem tratados pelo sistema central de controle de supervisão.

A diversidade de possíveis soluções, por muitas vezes é enganosa, pois quando o sensor é adequado, não pode ter condição de ser conectado ao sistema de controle.

A solução final passa a ser a utilização de outros circuitos auxiliares, para toma-los compatíveis uns com os outros, e assim, serem finalmente ligados aos controladores.

Em outros casos, um instrumento de medição disponibiliza suas informações com protocolo proprietário, não dispondo os "drivers" para o controlador escolhido e já adequado às demais tarefas.

Além da interoperabilidade, essa tecnologia também revoluciona os conceitos tradicionais de automação, pois descentraliza todo o controle do processo. Cada elemento atuador ou sensor trabalha de forma independente, ao mesmo tempo em que estão integrados as atividades dos outros sensores e atuadores.

Este projeto opera com altos índices de confiabilidade e produtividade, compatíveis com as atuais tendências de globalização da economia.

COMENTÁRIOS SOBRE A PLATAFORMA A SER UTILIZADA – LONWORKS

Atualmente os Terminal de Fertilizantes - TEFER não são automatizados, e devido a sua peculiaridade, como terminal de Graneis Sólidos, esta instalado em áreas acima de 150.000 m^2. Outra problemática enfrentada, é a oxidação de todos e qualquer tipo de metal, devido ao tipo de produto movimentado - Enxofre e Adubo em geral, levando principalmente os cabos elétricos e conexões a gerarem problemas de condutividade.

Face o exposto, buscamos uma tecnologia que atendessem aos seguintes pré requisitos:

- Circuitos independentes, a fim de não paralisar o controle, para eventuais problemas elétricos / eletrônicos.- Total INTEROPERABILIDADE;
- Baixo custo de instalação;
- Baixo custo de manutenção;
- Facilidade de manuseio, buscando menor custo do corpo técnico.

Face o exposto, apontamos a tecnologia LONWORKS, como a indicada para este tipo de aplicação, visto que é uma excelente plataforma para o desenvolvimento de redes de controles com topologia aberta, com aplicações em automação industriais, predial, residencial, e comercial, podendo operar os mais variados meios físicos (mídias).

Com uma vasta gama de ferramentas operacionais, viabiliza o controle e supervisão de uma rede implementada ou em implantação.

Esta tecnologia proporciona o controle total da rede, distribuindo e associando a ela, circuitos customizados com atividades individuais, sejam eles sensores ou atuadores, sendo todos dotados de inteligência própria e autonomia com prerrogativas de monitoração e controle.

Podemos então definir como Administração de REDE processo de instalação, onde os circuitos sensores e atuadores inteligentes (denominados – NÓS), são interligados e programados para trabalharem em conjunto numa mesma rede de controle, e manutenção de todo o este sistema.

É importante distinguir os conceitos de Administração da REDE, que não devem ser confundidos com a monitoração e controle das funções diversas e conjuntas propostas pela rede.

Administração de rede engloba todo o processo de instalação, interligação e configuração dos vários nós para que operem em conjunto e da melhor forma.

Uma ferramenta específica para a Administração de Rede, não atua diretamente na troca de aplicações, mensagens ou variáveis de rede, e também não é necessária ficar conectada permanentemente na rede para que esta opere.

Diferentemente das ferramentas e sistemas utilizados para controlar e monitorar os NÓS (sensores e atuadores) dessa rede, uma vez que estes são dispositivos ativos e participam normalmente na operação do sistema, sendo sua conexão permanente indispensável para seu funcionamento.

Uma única ferramenta pode combinar as funções de instalação e manutenção e depois pode permanecer para monitorar e controlar, ou pode ser usada como dispositivo em separado.

Sendo uma solução de projeto, para o embasamento de aplicações em redes de controle, a administração da rede é uma consideração chave para a determinação de cada ferramenta e componentes LONWORKS.

No Capitulo III, estaremos comentando e exemplificando as tarefas exigidas para instalação de uma rede baseada na tecnologia LONWORKS, bem como um breve resumo das várias maneiras de realiza-las.

Capitulo II

II.I Modelo atual de funcionamento

Os Terminais de Fertilizantes que atuam no Porto de Santos – TEFER e ULTRAFERTIL, não estão automatizados totalmente.

O TEFER , esta com o seu sistema de Balanças totalmente automatizada e interligada.

Como foi explanado no "histórico do Terminal", o modelo atual de funcionamento, mantém as determinações feitas por Codesp, e mantidas atualmente pela Fertimport, atuando como um – CAIS PÚBLICO –, ou seja, qualquer Operador Portuário – escolhido pelo Importador do produto, e que esteja com suas obrigações com a Codesp, trabalhará nas instalações do Porto de Santos, e especificamente no TEFER.

Sendo assim, nesta explanação, estaremos dividindo as operações no TEFER, em 02 áreas :

Área 01 - Operações no Pier / Descarga do Produto

Consiste nas atividades de :

- Planejar a chegada do Navio e tomar as providências necessárias para sua atracação e operação, disponibilizando – calado, Mão de Obra para atracação, e condições/ espaço para descarrega-lo nos Armazéns, em locais chamados de – Celas, Boxes, ou diretamente no caminhão (Descarga Direta), via esteiras (Pier 2), ou através do Grab abrindo diretamente em cima do caminhão..
- Disponibilizar equipamentos de Terra (Guindastes , esteiras, balanças de fluxo, Armazéns) para a descarga do navio, conforme requisição do Operador Portuário.

Documentação exigida para Fiel Depositário, para iniciar a descarga e ou a liberação do produto na retaguarda.

- Averbação da carga- Despachante / Fiscal Receita federal;
- Documentos de importação (BL, manifesto, DI's liberadas);
- Comprovante de Importação / pagamento de Armazenagem;
- Requisição de retirada e Nomeação de transportadora (Rodovia);
- Requisição de retirada e nota de expedição (Ferrovia);
- Liberação do local para armazenagem- Armazéns / Celas;
- Cadastro no sistemas de pesagens;
- Controle de produto armazenado para Navio (Através Armazém / Celas);
- Controle de Cobrança de Armazenagem p/ período através do saldo da carga.

Área 02 - Operações de Retaguarda

Consiste nas atividades de:

- Manter contato com os Importadores a fim de programar quantidades de produtos a serem retirados no próximo período, e o recebimento e aprovação da documentação necessária.

- Com esta informação, deverá ser requisitada a quantidade de avulsos necessários para operar os equipamentos de entrega:

  - Pás Carregadeiras,
  - Tratores,
  - Carregador de vagões.

- Acompanhar e fiscalizar a entrada dos caminhões no Terminal, a sua pesagem, o seu carregamento no Armazém , o enlonamento do caminhão, a pesagem e libera-lo para a saída do Terminal.

❖ Vide documentos anexo:

- Previsão de retirada de produto solicitado pelo importador;
- Mapa de armazenamento de produtos – diário (Para orientação e planejamento de trabalhos, bem como informação a Importadores);
- Requisição de Mão de Obra Avulsa - OGMO;
- Ficha de entrega de produto para comprovante de importação;
- Boletim de serviços em Armazéns (discriminação de carga retirada p/ Navio/ Comprovante de Importação);
- Acompanhamento de paralisação de equipamentos;
- Registro de pesagem de vagões;
- Ficha de acompanhamento de retirada para ferrovia;
- Transferência de dados para emissão de Nota Fiscal;
- Demonstrativo de entrega diária de produto – Rodovia;
- Demonstrativo de entrega diária de produto – Ferrovia;
- Mapa com movimentação diária de entrega para modais Rodoviário e Ferroviário.

II.I.I. BALANÇAS

O TEFER opera com 04 balanças rodoviárias com capacidade de 80,000 tons/un, e 02 balanças Ferroviárias p/ 150,000 tons/un.

As mesmas estão automatizadas, e interligadas entre si, e o Posto Fiscal e o Gerenciador (fiel), podendo o auto entrar e sair para balanças diferentes.

Abaixo explanamos o seu funcionamento:

a) Componentes de hardware:

- Concentrador:
  - Microcomputador;
  - Modems para comunicação.

- Posto de pesagem:
  - Microcomputador;
  - PLC;
  - Módulo de pesagem (balança);
  - Modem;
  - Sensores de posição;
  - Sirene;
  - Semáforo.

- Geral:
  - Rack para posto de pesagem;
  - Cablagem;
  - Rack para modems.

b) Componentes de software:

- Concentrador:
  - Programa que administra banco de dados de pesagens;
  - Programa de comunicação via modems.

- Posto de pesagem:
  - Programa supervisor de pesagens (no microcomputador);
  - Lader (Programa) de controle de dispositivos (sensores, sirene e semáforo) e pesagem (no PLC);
  - Programa de comunicação via modem.

- Geral:
  - Programa gerenciador do sistema de pesagens (relatórios, etc.)

c) Processo de pesagem

No posto de pesagem:

- Função do PLC:

O Lader do PLC monitora fisicamente a entrada e saída do caminhão sobre a plataforma de pesagem, analisando os sensores que estão posicionados na entrada e saída da plataforma e peso instantâneo durante a entrada do caminhão. Através deste monitoramento o Lader consegue verificar se houve alguma irregularidade no posicionamento do caminhão sobre a plataforma ou na entrada e saída do mesmo. Isso é

possível devido a Interceptação dos sensores pelas rodas (eixos) do veículo e da curva de pesagem durante a entrada e saída do mesmo, ou seja, é contado quantos eixos adentram a plataforma de pesagem e quantos saem no final da pesagem, também é verificado o sentido que o veículo está se deslocando. Simultaneamente é verificado o peso de cada eixo que adentra a plataforma, e é comparado com o peso de cada eixo que sai da plataforma de pesagem. Caso seja verificada alguma distorção neste processo, o Lader informará ao microcomputador que a pesagem foi inválida devido a alguma irregularidade.

- Função do microcomputador:

O programa do microcomputador cuida de toda a interface homem-máquina, interagindo com o operador para entrada de dados e informações sobre a pesagem em andamento. Também é sua função gerenciar a pesagem controlando o PLC e a comunicação com o Concentrador de dados, ou seja, busca os dados necessários a pesagem no Concentrador (dados do veículo, informações históricas do motorista, tara se já pesou vazio, etc.) e envia os dados da pesagem, também gerencia as atividades do Lader e suas respostas.

No Concentrador de dados:

- Função do microcomputador:

Armazena os dados de pesagens recebidos dos postos de pesagens no banco de dados principal e pesquisa e informa todas as requisições de dados solicitadas pelo microcomputadores dos postos de pesagens. Armazena dados de pesagens, veículos, motoristas, histórico de irregularidades, etc.

Gerenciador do sistema

Através do banco de dados formado pelo Concentrador, realizada relatórios e gráficos que permitem o gerenciamento do sistema, além de permitir a administração do sistema com cadastramento de operadores, senhas de acesso, dados de postos de pesagens, etc.

II.I.II. POSTO FISCAL

É o local de controle onde todos os dados da Carga a ser retirada, da transportadora, Importador e do veículo, são adicionados ao sistema.
Fisicamente também são notadas as placas do auto, as condições de limpeza da carroceria (através de espelhos direcionados ao interior da carroceria), os dados da minuta, inclusive se a carga esta totalmente liberada para o carregamento.

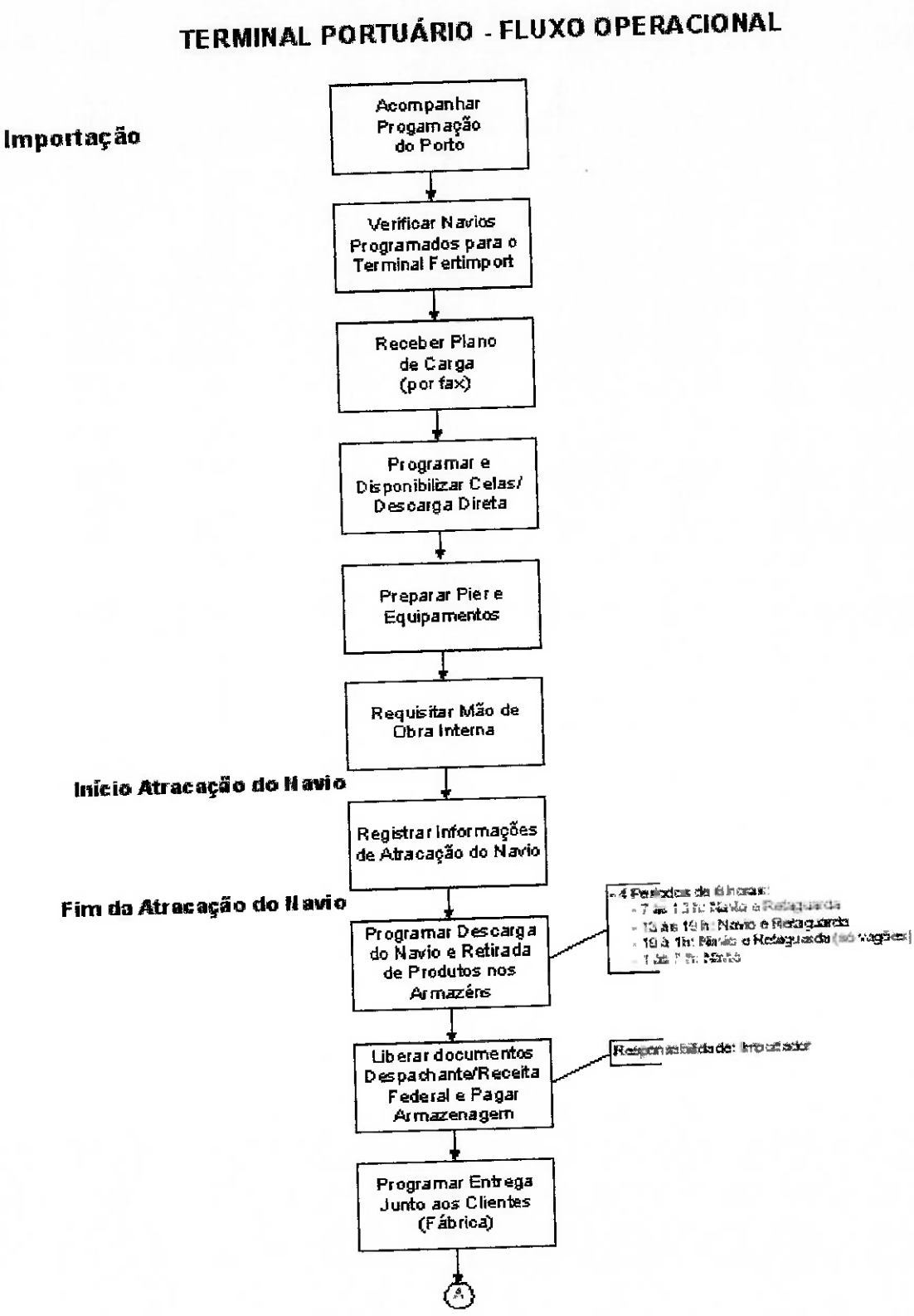
Inseridos os dados e com a vistoria OK, o auto dirigi-se a balança que estiver desocupada, e após a parada do auto em cima da plataforma da balança, os dados

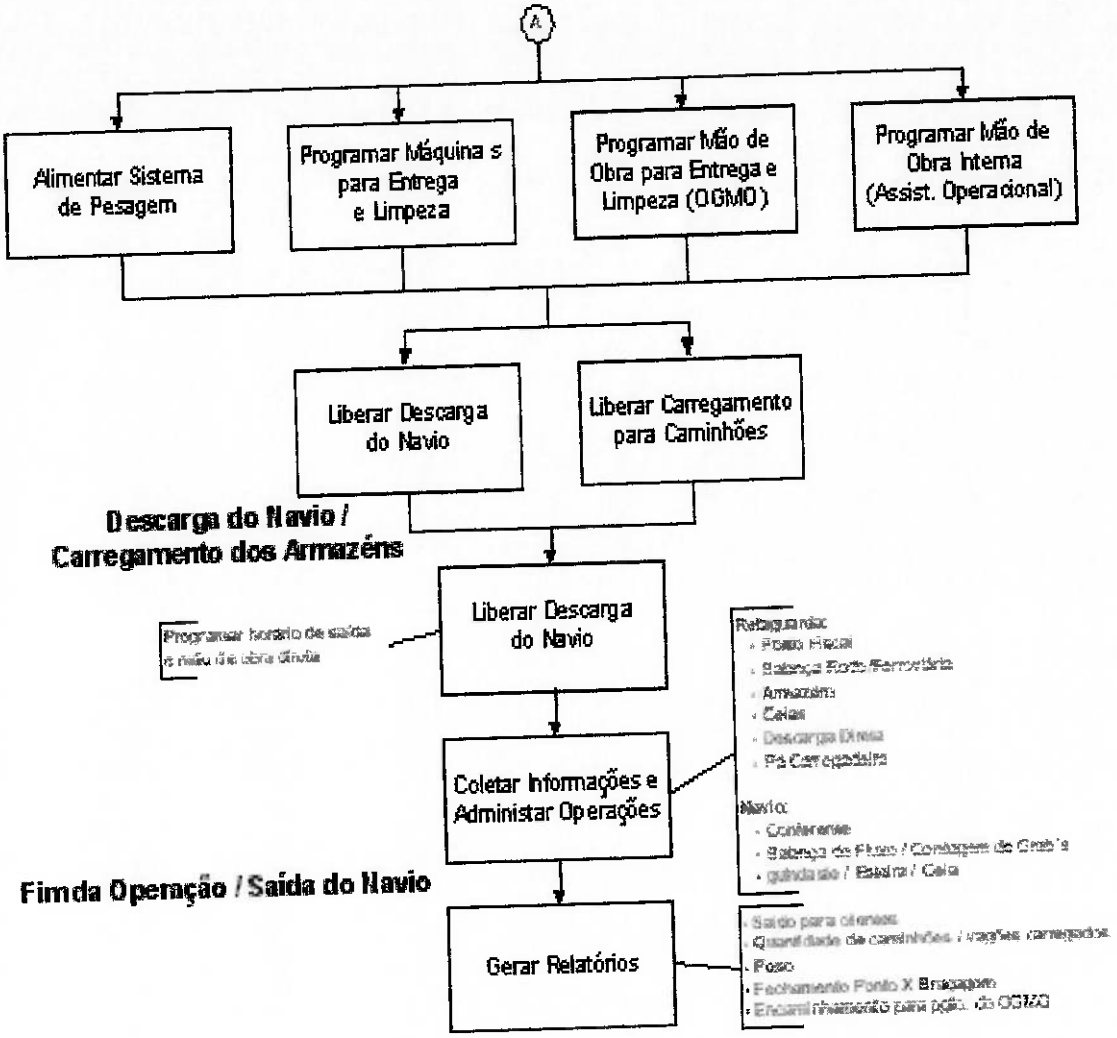
imputados no PF, são agora apontados em um bilhete de Pesagem, que será impresso juntamente com o peso bruto do auto.

Estes equipamentos (micros, CLP's e sistema), foram comprados após a entrada do novo Gestor do Terminal, bem como os procedimentos de segurança.
Após a pesagem o auto dirigi-se para o Armazém, onde deverá ser carregado e novamente pesado, para emissão do Bilhete que dará origem a Nota Fiscal.

- Vide documentos anexos apresentando as telas utilizadas, e os documentos emitidos.

II.II. Codesp – TEFER – Lay out do fluxo documental (modelo de informação)





Comentário – Fluxo Operacional – Terminal Portuário

1) Acompanhar programação do Porto

A Codesp, e as Agências de Navegação, emitem boletins de previsão de atracação e estadia de navios no Porto de Santos.

Nestes boletins são informados:

- Nome do navio e procedência;
- Data de chegada e previsão de saída;
- Berço que o navio deverá atracar;
- Tipo da carga e o total manifestado;
- Operador portuário nomeado e a Agência protetora do navio.

2) Verificar Navios Programados para o Terminal –TEFER

Com as condições acima, acompanhar a movimentação dos navios nomeados para o Terminal.

3) Receber o plano de Carga

A Agência envia a disposição e o tipo da carga depositada por porão.

4) Programar disponibilidade de Celas/ Descarga Direta

Com base na data de chagada do navio, e o tipo de carga, o terminal inicia a disponibilidade de espaço para a descarga do navio.

5) Preparar Pier e equipamentos:

Terminal deve providenciar a disponibilidade dos equipamentos a serem utilizados – Guindastes, esteiras, Pás carregadeiras, bem como os mesmos deverão estar devidamente limpos, sem resíduos de produtos de outras descargas.

Quanto ao Pier, deverão ser inspecionadas as: defenças, cabeços de amarração, condições do piso e de limpeza em geral.

6) Inicio da Atracação :

Após definida a data e hora da atracação (em função da tábua de mares e da fila de navios), o terminal prepara a equipe de atracação composta por 08 homens, que ficam a disposição para realização de tal tarefa.

Esta operação é acompanhada de pessoal da fiscalização Codesp, do operador portuário/ terminal .

7) Registrar informações da atracação do Navio:

Representante da fiscalização do porto, e do terminal, registram os horários de colocação dos cabos de amarração do navio, o seu registro no Porto de Santos, e acompanha a arqueação do navio.

ATIVIDADES DE RETAGUARDA :

1) Programar a Descarga do Navio e retirada de produtos nos Armazéns:

Terminal, deverá apontar ao operador portuário, os locais dentro do Armazém, em que a carga irá ser depositada, bem como nos casos de entrega de produtos aos fabricantes, devem ser confirmadas as quantidades junto ao fabricante.

2) Liberar documentos – despachante / DRF, de produtos nos armazéns:

Quando a carga "toca" no chão do armazém, o fiel lhe concede um aviso de presença de carga no Terminal, que posteriormente, deverá ser liberada pelo despachante nomeado pelo dono da carga (Importador), para liberá-la.

3) Programar retirada com as fábricas:

Com objetivo de otimizar a entrega dos produtos, o Terminal contacta as fabricas e programa o volume a ser retirado, bem como serve de base para a devida programação de Mão de Obra junto ao Ogmo.

4) Alimentar o sistema de pesagem:

Os n.°s de DI - (Declaração de Importação), devem ser registrados na memória do sistema, para que possa funcionar e controlar o sistema;

5) Programar máquinas de limpeza:

Com base nas informações dos importadores/fabricas, devemos agendar Mão de Obra junto ao Ogmo bem como equipamentos (Pás carregadeiras), para entrega deste material.

6) Gerar relatórios de :

- Entrega para armazém;
- Saldos;
- Fechamento de braçagens;
- Fechamento do navio ao término do produto.

Capítulo III

III.I Descrição dos equipamentos atuais, nível de automação nos equipamentos e o supervisório administrativo

a) Guindastes:

- Takraf, com capacidade para 300 tons/hr., peso total- 239,554 tons, equipados com Grab´s, Não automatizado.

b) Esteiras:

- Pier 1 – correias transportadoras de graneis de 24“, em estruturas metálicas suspensa (móveis e fixas);
- Pier 2 - correias transportadoras de graneis de 42“, em estruturas metálicas suspensa , equipados com raspadores em razão do trabalho com enxofre.

Controladas por painel de comando central e bloqueios manuais, não automatizados.

c) Balanças de fluxo:

- 04 un instaladas no Pier 1, marca Schenk, com capacidade de 500 tons/hr., não automatizada.
- Pier 2- Não possui balanças de fluxo.

d) Balanças Rodoviárias:

- 04 un marca Toledo para 80 tons , automatizadas e interligadas p/ sistema *

e) Carregador de vagão:

- 04 un , construção local, equipados com esteiras de 24”, acionados por motores de 25 HP, com capacidade de 300 tons/hr, não automatizada.

f) Armazéns

- 06 un ,com capacidade de 30.000 tons, dividido para 21 celas de capacidade de 1500 tons/un . O empilhamento é feito p/ esteiras fixas e móveis (04 un) ligados ao Pier 1 (Armazéns 1 e 3) ;

- e por Tripper com esteira de 42" ligados ao Pier 2, (Armazéns 2, 4, 6 e Pátio de Enxofre).

Obs. Nenhum equipamento nos armazéns é automatizado.

❖ Para o escoamento/entrega do produto estocado, são utilizadas 12 Pás Carregadeiras, c/ capacidade de 3,5 m^3 .

Sistema de automação das Balanças Rodoviárias, Posto fiscal e Posto do Fiel :

III.II Descrever tecnologias de mercado

- Allen Bradley , Rockwell , Reliance Eletric – São fornecedores de equipamentos e sistemas mais implementados atualmente nos terminais de graneis automatizados, como – Cargil, Coopersucar .
- São ferramentas de software , para a supervisão e controle e aquisição de dados de proceso, para plataformas Windows 95 e Windows NT, que provem ao usuário o estado da arte em tecnologia, tais como controles Active X, em telas gráficas, além de total integração c/ produtos Microsoft.
- Para a migração dos dados, são utilizados CLP, A.I, que permitem importar diretamente de seus arquivos, os dados fornecidos p/ equipamentos monitorados.

a) Coletores de dados :

- Outro equipamento utilizado nos terminais portuários, são os coletores de dados acoplados a radio frequência. Normalmente utilizados nos sistemas de balanças,
- Coletam os dados já cadastrados do caminhão, e transmitem os dados ao sistema central, quando o auto é pesado na balança , eliminando a presença do balanceiro.
- No carregamento de vagões , este equipamento também é utilizado, na coleta de dados do vagão .

Sistema Microled – automação de balanças rodoviárias e ferroviárias :

Capítulo IV

IV.I. Modelo Proposto

A Produção Brasileira de Fertilizantes, concentra-se no período de Agosto a Novembro e totaliza 16 milhões de toneladas, sendo que no Porto de Santos circulam entre os terminais Tefer e da Ultrafértil, 4 milhões de toneladas.

Dentro desta necessidade, buscamos apresentar um projeto modular de automação de terminais de graneis (especificamente - fertilizantes), que opera a descarga do produto por correias transportadoras, gerando massa de dados para ferramenta gerencial.

Viabilização do projeto com total INTEROPERABILIDADE

A atividade de projetar, implantar e manter uma automação, seja ela para apenas uma máquina ou para uma planta de fabrica ou terminal portuário, com várias maquinas e motores em operações sincronizadas, encontra dificuldades pela necessidade de atender à interoperabilidade entre os circuitos operantes, sensores ou instrumentos de medição a serem tratados pelo sistema central de controle de supervisão.

A diversidade de possíveis soluções, por muitas vezes é enganosa, pois quando o sensor é adequado, não pode ser conectado ao sistema de controle. A solução final passa a ser a utilização de outros circuitos auxiliares, para torna-los compatíveis uns com os outros e assim, serem finalmente ligados aos controladores.

Em outros casos, um instrumento de medição disponibiliza suas informações com protocolo proprietário, não dispondo de drivers para o controlador escolhido e já adequado ás demais tarefas.

Além da interoperabilidade, essa tecnologia também revoluciona os conceitos tradicionais de automação, pois descentraliza todo o controle do processo. Cada elemento atuador ou sensor trabalha de forma independente, ao mesmo tempo em que estão integrados as atividades dos outros sensores e atuadores. Gerando desta feita o conceito de nó inteligente. Este projeto opera com altos índices de confiabilidade e produtividade, compatíveis com as atuais tendências de globalização da economia.

Toda definição em automação requer a priori, um profundo conhecimento do sistema em utilização para a mais segura e objetiva conduta operacional. Para tanto estivemos em campo, catalogando dados de engenharia para que constatássemos a sua capacidade de movimentação e estocagem de produtos em sua atual área física.

Este catalogo tem por finalidade, embasar toda e qualquer atitude de automação voltada a este projeto. Nos cabe lembrar que o mesmo traz ao leitor uma visão global facilitadora da estrutura portuária.

Apresentaremos a composição deste catalogo em forma de planilhas, plantas e suas memórias de cálculo em forma de anexos para não avolumarmos o corpo principal deste conteúdo.

Passaremos a expor nosso estudo , para aplicação da tecnologia Lon Works em nossa proposta de automação.

Lon Works – pesquisa da tecnologia.
Descrição, Aplicação, Vantagens, Desvantagens e Componentes

IV.I.I. Descrição do Barramento

A tecnologia LonWorks fornece uma solução para muitos problemas de projeto, construção, instalação e manutenção de redes de controle cujo tamanho pode variar de 2 a 32000 dispositivos conectados através de par trançado, linha de transmissão, cabo de fibra óptica, cabo coaxial, RF ou infravermelho. Pode ser usada em qualquer lugar - de supermercado à plataforma de petróleo, de foguetes aos veículos utilitários, de residências aos arranha-céus. O controle de uma rede LonWoks é distribuído. Dispositivos de controle inteligentes chamados nós, comunicam entre si usando um protocolo comum. Cada nó na rede contém uma inteligência embutida que implementa o protocolo, distribui o processamento de cargas e efetua as funções de controle. Com as funções de controle distribuídas, o desempenho e a confiabilidade dos sistemas que utilizam tecnologia LonWorks são consideravelmente aumentadas. Além disso, cada nó inclui uma interface física que acopla o nó microcontrolador com o meio de comunicação. Um nó típico, numa rede de controle LonWorks, executa tarefas simples. Dispositivos como sensores de proximidade, chaves, detetores de movimento, relés e controladores de motores podem ser nós na rede. A tecnologia LonWorks é um sistema aberto, permitindo combinações de componentes de diferentes fabricantes e, permitindo também, adicionar novas funções de controle com um custo mais baixo. A tecnologia LonWorks possui um protocolo chamado LonTalk que implementa as sete camadas do modelo OSI - Modelo de Referência para Interconexão de Sistemas Abertos e possui mecanismos que impedem a modificação acidental ou intencional. Inclui ainda, outras características tais como: funções de reconhecimento (acknowledgement), comunicação peer-to-peer, prioridade na transmissão, detecção de mensagens duplicadas, evita colisões, retransmissão automática, detecção e correção de erros, padronização e identificação do tipo de dados. É um protocolo aberto que permite a qualquer companhia colocá-lo no processador que deseja. Isto significa que aplicações que requerem processadores de 16 ou 32 bits não necessitam mais de programa de interface para o microprocessador. Esse protocolo está sendo analisado pela Associação de Indústrias Eletrônicas afim de ser recomendado como um padrão para automação residencial. Além disso, esse protocolo é parte do American Society of Heating, Refrigeration, and Air-Conditioning Engineers´s BACnet control standard for buildings. Isto é conhecido como ANSI/ASHRAE 135-1995. Apesar da possibilidade de implementar o protocolo LonTalk num processador genérico, a Echelon desenvolveu o Neuron Chip que é mais apropriado para aplicações de controle por várias razões: o Neuron chip é composto por três processadores de 8 bits onde dois deles são otimizados para executar o protocolo e o terceiro para aplicações dos nós. O Neuron chip incorpora watchdog timers, 35 tipos de controladores de dispositivos, um sistema operacional em tempo real distribuído, três tipos de memória, possui um vetor de 48 bits acessível via software que garante um endereço disponível quando da instalação de um nó. O protocolo LonTalk possui alta confiabilidade, pois garante que a informação foi transmitida e recebida com sucesso. Garante a integridade dos dados porque não usa paridade nem checksum, mas sim, controle por CRC. Os transceivers, equipamentos

utilizados na interligação dos nós com o barramento, são capazes de corrigir e detectar erros evitando a retransmissão. O protocolo LonTalk utiliza CSMA p-persistente preditivo com opção de prioridade e detecção de colisão. Esta tecnologia supera os inconvenientes das técnicas tradicionais de CSMA.

IV.I.II. Áreas de aplicação

A tecnologia LonWorks é utilizada em:

- Automação residencial e predial;
- Automação dos serviços de utilidade pública (gás encanado por exemplo);
- Automação industrial;
- Transporte.

Descrição de algumas características de redes LonWorks para controle de residências e gerenciamento de energia.

Antes de surgir o barramento Fieldbus as residências eram equipadas com dispositivos de controle como por exemplo: portão automático, alarme, circuito interno de TV mas estes dispositivos não estavam interligados em rede. LonWorks é uma tecnologia de controle de residências e edifícios capaz de integrar os diversos dispositivos num único sistema além de possibilitar a interconexão de produtos de diferentes fabricantes. Esse barramento é uma solução flexível, poderosa, de arquitetura aberta, com tempos de resposta rápidos e com custo relativamente baixo. Observe a seguinte situação: "Imagine acordar com uma voz suave, com música ao fundo falando que já é hora de você se levantar. Quando você entrar no banho o sistema de aquecimento começa a funcionar fazendo com que sua cafeteira já comece a preparar o seu café na cozinha. Enquanto você estiver tomando o café da manhã na cozinha a televisão ativa seus e-mails e os lê para você através do sintetizador de voz. Se você entrar no seu carro elétrico e perceber que esqueceu de recarregar as baterias perceberá que este problema já foi detectado e automaticamente as baterias já foram recarregadas durante a noite. Se você por acaso não gostasse de chegar cedo ao trabalho porque pela manhã o prédio onde você trabalha está sempre escuro, frio e ligeiramente assustador e descobrisse que a partir de agora assim que você estacionar o seu carro no parque de estacionamento as luzes do prédio onde você trabalha automaticamente se acenderão, seu computador será ligado automaticamente e os seus e-mails acessados. Se você descobrisse que não será mais necessário trabalhar no chão de fábrica como já era de costume, mas sim num confortável escritório e certo de que todas as atividades realizadas em chão de fábrica estão sendo realizadas eficientemente, além de segurança, gerenciamento de gasto de energia nos horários de pico e automação de muitas outras atividades". A situação descrita acima já é possível graças a tecnologia LonWorks que já se encontra presente no mercado.Com o uso de sistemas de controle inteligentes para residências é possível controlar:

1) Iluminação:

É possível controlar luzes em qualquer cômodo da casa em qualquer hora do dia

2) Sistemas de Ar Condicionado:

Os locais da casa onde você se encontra são refrigerados e o sistema será desligado automaticamente quando você sair.

3) Aquecimento:

Enquanto você dorme o controlador do volume diminui o aquecimento e quando você acordar pela manhã ele automaticamente deixa a sua casa numa temperatura ligeiramente morna que é mais agradável.

4) Segurança:

Sistema de alarme contra roubos da sua casa reconhece pessoas estranhas mas admite que as pessoas possuidoras de uma senha como encanadores ou outros prestadores de serviços possam entrar sem problemas.

5) Irrigação:

Os irrigadores são ligados automaticamente quando o gramado está precisando de água e são desligados quando chove.

6) Gerenciamento de energia:

Nos horários de pico você pode utilizar a tecnologia LonWorks para desligar alguns dos aparelhos elétricos da sua casa de acordo com a prioridade que você mesmo estipular evitando assim desperdício de energia.

7) Entreterdimento:

De qualquer cômodo da sua casa você poderá controlar a Televisão, o CD e o vídeo cassete através de um controlador remoto fácil de usar.

IV.I.III. Vantagens e Desvantagens

VANTAGENS:

1) Rapidez no desenvolvimento de projetos:

LonWorks permite desenvolver um sistema desde o início em menos de um ano causando um aumento da renda pois seu sistema entra mais cedo no mercado.

2) Baixo custo:

LonWorks é uma das alternativas de mais baixo custo dentro deste segmento segundo opinião de usuários.

3) Interoperabilidade:

Qualquer produto ou sistema baseado na tecnologia LonWoks pode se comunicar com outro produto ou sistema que também tem esta tecnologia não importando se são de fabricantes diferentes.

4) Modularidade:

É possível adicionar aos poucos dispositivos à sua rede de controle LonWorks.

5) Boa performance:

A velocidade da rede é aumentada pois o processamento é distribuído.

6) Boa Confiabilidade:

Cada ponto da rede possui inteligência para processar as informações no mesmo local onde estas são adquiridas, evitando que haja concentração em um único nó.

7) Disponibilidade:

Existe no mercado uma variedade de fabricantes usando tecnologia LonWorks.

8) Padronização:

As redes baseadas na tecnologia LonWorks foram reconhecidas pela ANSI (American National Standards Institute) como sendo verdadeiramente uma arquitetura aberta.

DESVANTAGENS:

1) O uso da tecnologia LonWorks tem seu uso limitado à redes de controle que não requerem taxas de transmissão superiores a 1.25 Mb/s e tempos de resposta através da rede de 7-13 ms e permite somente comunicação entre equipamentos LonWorks.

IV.I.IV. Componentes disponíveis no Mercado

A Echelon, que é a criadora da tecnologia LonWorks, é a principal fornecedora de produtos desta tecnologia. Possui uma linha de mais de 75 produtos que inclui todos os equipamentos necessários para desenvolvimento, fabricação, instalação e manutenção de redes LonWorks. Alguns dos equipamentos são:

- LonWorks Transceivers;
- Gateways e Interfaces para redes LonWorks;
- Roteadores LonWorks;
- Ferramentas de Serviços de Rede LonManager;
- Ferramentas de Desenvolvimento.

Já a Motorola e a Toshiba fabricam com exclusividade e vendem os Neuron Chips utilizados nos nós das redes LonWorks. Alguns dos componentes disponíveis no mercado serão melhor descritos abaixo para uma melhor compreensão:

1) KIT DE I/O MULTI-FUNCIONAL LONBUILDER modelo 2800:

O kit de I/O multi-funcional LonBuilder é uma coleção de dispositivos de I/O para o LonBuilder Developer's Workbench. Estes dispositivos de I/O apresentam display numérico com 4 dígitos, sensor de temperatura, transdutor de pressão, dois push buttons e dois LEDs. Os dispositivos I/O são fisicamente montados no módulo I/O Gizmo 2, o qual pode ser usado para testar aplicações em LonWorks. O Gizmo 2 é conectado ao Emulador Neuron LonBuilder utilizando uma placa de interface e cabo de interface de I/O, sendo que ambos estão incluídos no kit. Os dispositivos Gizmo 2 fornecem hardware que podem ser usados em nós LonWorks. Muitos programas em C são incluídos com o kit a fim de fornecer soluções de engenharia para uma variedade de problemas de I/O. O Gizmo 2 é também usado para construir um nó capaz de simular um gerador de testes para todos os outros nós.

Descrição:

O kit de I/O multi-funcional inclui os seguintes componentes:

a) Gizmo 2: Um módulo com 8 Neurons Chips compatível com dispositivos de I/O;

b) Placa de interface de aplicação (AIB): Uma placa de expansão para LonBuilder que fornece acesso ao Neuron Chip de I/O, comunicações, clock externo, reset e sinais de serviços. Um conector DB-25 na frente do AIB fornece acesso fácil para estes sinais. As comunicações, I/O e sinais de serviço são protegidos através de diodos de proteção ESD;

c) Cabo de interface de I/O de aplicação: É um cabo que conecta os sinais de I/O do AIB para o Gizmo 2;

d) Discos de exemplos de programação de I/O multi-funcional:
Exemplos que demonstram uma variedade de objetos de I/O que dão suporte ao Neuron C.

2) KIT DE INTERFACE DE APLICAÇÃO LONBUILDER modelo 27810:

O kit de interface de aplicação LonBuilder (AIK) é uma interface de I/O para LonBuilder que simplifica o desenvolvimento e testes de I/O externas e hardware de comunicação. O AIK dá suporte também aos testes dos circuitos dos nós. Permite também testar e fazer debug dos produtos utilizando uma poderosa ferramenta de Neuron C debugger.

Descrição:

O AIP Kit inclui os seguintes componentes:

a) Placa de interface de aplicação (AIP):
Uma placa de expansão para LonBuilder que fornece acesso para I/O do Neuron Chip, comunicações, clock externo, reset e sinais de serviço. Um conector DB-25 na frente do AIB permite fácil acesso a estes sinais. As comunicações, I/O e sinais de serviço são protegidos com diodos de proteção ESD;

b) Cabo de interface de aplicação (AIC): Cabo que conecta a AIB ao hardware externo. Apresenta imunidade ao ruído;

c) Interface de aplicação do módulo (MAI): A MAI substitui os módulo de controles do nó usuário;

d) Adaptador de interface de aplicação (AIA): Uma placa de interface que é compatível com adaptadores Neuron 3120 e 3150 disponíveis pela Emulation Technologies.

3) KIT DE DESENVOLVIMENTO DE LNS PARA WINDOWS modelo 34303:

O kit de desenvolvimento de LNS para Windows fornece softwares requeridos para aplicação LNS em Windows NT e Windows 95. Estas aplicações podem ser locais, ou seja, na mesma máquina do servidor ou aplicações remotas através da rede LonWorks.

O Kit inclui os seguintes componentes:

- Servidor de Objeto LCA (OCX);
- Servidor de Dados LCA;
- NSS for Windows server e database manager;
- Drivers de rede para o PCNSI e PCC-10;
- Visual Basic e código fonte ANSI C;
- Um ano de modelo 91000-0 LonSupport para LNS/LCA.

Descrição:

a) Servidor de Objetos LCA (OCX):

O Servidor de Objetos LCA converte objetos LNS (nós, roteadores, etc) gerenciados pela NSS para Windows num padrão de objetos OLE. Representando objetos LNS como objetos OLE, o Servidor de Objeto LCA simplifica os desenvolvimentos de aplicativos para Windows 95 ou NT. Os controles OLE são de alta performance, com 32 bits, com objetos programáveis em linguagens independentes que podem ser usados com uma variedade de ferramentas de desenvolvimento, incluindo desenvolvimento de aplicativos rápidos em Visual Basic. O servidor de objeto fornece também uma ferramenta central capaz de compartilhar informações e objetos entre componentes e ferramentas múltiplas.
O servidor de objeto pode também ser visto como uma nova linguagem para ferramentas LonWorks baseado numa automação OLE. Esta linguagem é facilmente programável numa variedade de ambientes, incluindo Visual Basic e Visual C++.

b) Servidor de Dados LCA API:

Assim como o Servidor LonManager DDE, o Servidor de Dados LCA API é um instrumento de alta performance para monitoração de sistemas e controle usando os serviços e dados fornecidos pelo NSS para Windows.
Usando o servidor de dados, as aplicações dos usuários podem observar valores das variáveis da rede e mandar mensagens e podem mudar os valores das variáveis da rede ou enviar mensagens para efetuar operações da rede. Para simplificar aplicações dos usuários, o servidor de dados converte dados da rede em um formato de strings que podem ser exibidas.

c) NSS para Windows:

O NSS para Windows fornece serviços de alto nível para instalação de redes, manutenção, monitoramento e controle para redes LonWoks com um único ou com múltiplos canais com mais de 32385 nós. O NSS para Windows admite usuário local assim como usuários remotos (usuários em diferentes nós da rede). A interface de programação é a mesma para ambos os usuários e eles tem acesso aos mesmos dados e serviços NSS. O NSS para Windows é acessado por host cujo sistema operacional seja Windows 95 e Windows NT usando o servidor de objeto LCA e servidores de dados. Para dar suporte aos usuários remotos, o NSS para Windows fornece serviços de gerenciamento que permite aos clientes remotos acessarem o sistema e o monitor de outros usuários remotos. O NSS fornece serviços de gerenciamento de diretórios que permite aos usuários remotos acessarem serviços de aplicações específicas das quais eles necessitam.

4) KIT DE DESENVOLVIMENTO PARA CONSTRUÇÃO DE REDES LONBUILDER:

Função: Contém ferramentas e componentes necessários para desenvolver nós e redes LonWorks.

O kit inclui:

a) LonBuilder Developer's Workbench: o qual se subdivide em 3 ferramentas de desenvolvimento:

- Sistemas de desenvolvimento de nós múltiplos;
- Gerenciamento de rede;
- Analisador de protocolo.

4 Transceivers FTM-10 que permite integrar os nós criados pelo LonBuider Developer's Workbench com nós de uma rede externa.

c) Servidor DDE permite criar interfaces gráficas para o usuário das redes LonWorks usando aplicativos do Windows que possuem capacidade dinâmica de troca de dados (DDE).

d) Placa PCNSI de interface entre redes LonWorks e o PC.

e) Um ano de assistência técnica.

Descrição:

a) Software LonBuilder:

É o software usado para criar nós e redes LonWorks. Inclui:

- Um editor, um compilador, um debugger usado para criar e corrigir os programas aplicativos para o Neuron Chip;
- Gerenciador de rede usado para instalar e configurar os nós durante o desenvolvimento da rede;
- Um analisador de protocolo usado para monitorar o desenvolvimento da rede e interpretar suas atividades, permitindo debugar a rede.

Vantagem:

- Redução no tempo de desenvolvimento da rede.

b) Hardware de estação de desenvolvimento (Development Station Hardware):

Esta estação é um rack expansível que inclui 2 nós LonWorks, um deles para gerenciamento de rede e outro para análise de protocolo. Inclui processador LonBuilder que se comunica com uma rede externa ou com dispositivos de potência. Esta estação de desenvolvimento simplifica o debug de redes de controle e suporta até 6 emuladores.

c) 2 Neuron Emuladores:

Este par de nós são usados com o Neuron C debugger para rodar e fazer o debuger nos programas do Neuron C e para testar os transceiver e as entradas e saídas.
O emulador e o Neuron C debugger são usados para setar os breakpoints, modificar as variáveis e a aplicação da rede através de dois programas executados em paralelo. Eles podem se comunicar entre si , com o gerenciador da rede, com o analisador de protocolo através da topologia de rede incluida na estação de desenvolvimento.

d) Cabo e Adaptador de interface Lonbuilder:
O adaptador de interface é uma placa compatível com barramento ISA de 8bits pode ser instalado em qualquer computador compatível com um IBM PC. Este adaptador possibilita uma conexão com taxa de 10Mbps entre um PC e a estação de desenvolvimento.
Este adptador reduz o investimento total pois permite que os projetista usem seus PC's com as ferramentas Lonbuilder.
Um cabo de 3 metros é usado para conectar o adpatador de interface à estação de desenvolvimento.

e) Roteador Lonbuilder:

quando instalado na estação de desenvolvimento e usado com 2 transceivers este roteador conecta a topologia de rede existente na estação desenvolvimento com qualquer tipo de rede externa. Pode-se configurar os transceivers para ligar estação de desenvolvimento a canais de comunicação diferentes.

f) Transceivers Lonworks:

O kit de desenvolvimento Lonbuilder inclui quatro adaptadores SMX Lonbuilder e quatro FTM-10 Transceiver. Estes quatro adaptadores SMX e os quatro transceiver (que são compatíveis com o adaptador SMX) são instalados respectivamente, dois nos neurons emulators, um na estação de desenvolvimento, e um no analisador de protocolo.

g) kit Gizmo 3 Lonbuilder:

Este kit inclui uma coleção de dispositivos de entrada/saída úteis para desenvolver aplicações baseado na tecnologia Lonworks, uma placa de interface para aplicações Lonbuilder e um cabo para conectar os dois. Esta interface de aplicação deve ser instalada no emulador. Os dispositivos Gizmo 3 já possuem hardware que pode ser usado para desenvolver uma variedade de nós.
Diversas amostras de progrmas para o Neuron C estão incluídos no kit para fornecer soluções para problemas de entrada/saída.
Este kit também pode ser usado para criar um nó que age como um gerador de testes dos outros nós.

h) Kit de interface de aplicação:

Este kit inclui uma placade interface de aplicação e cabo usados para conectar oas entredas e saídas e o tranceiver ao emulador Neuron. Para permitir o teste em circuito da parte de hardware do nó, o kit inclui a interface do módulo de aplicação Lonbuilder para nós baseados nos módulos de controle Lonworks; e um adaptador de interface de aplicação Lonbuilder, para usar com o adaptador Neuron 3150 ou 3120 que serve para nós que não são baseados nos módulos de controle Lonworks.

i) Servidor DDE Lonmanager:

Este servidor facilita a criação de uma interface gráfica para o usuário numa rede Lonworks. Usando este servidor qualquer aplicativo do Windows que permita troca de dados dinâmica (DDE) pode monitorar e mudar o valor das variáveis da rede e monitorar e enviar mensagens.

j) Placa de Interface para PC PCNSI:

Esta placa se constitui num barramento ISA que promove uma interface entre o PC e a rede Lonworks, interface esta de alta performance capaz de transmitir 228 pacotes por segundo.Os PC´s podem ser usados para fazer um controle, monitoração e gerenciamento centralizado da rede Lonworks. O PCNSI usa o transceiver SMX para fazer a interface para qualquer meio de comunicação compatível com Lonworks.

Como usar o Lonbuilder Developer´s Workbench:

O Lonbuilder Developer's Workbench é usado para criar nó sob medida para sua aplicação. Na verdade, estes nós são dispositivos Lonworks sob medida que combina um Neuron Chip, I/O, transceivers para comunicação , e uma aplicação que interage tanto com o dispositivo de I/O local quanto com outros nós da rede Lonworks. A figura seguinte ilustra os componentes de um nó sob medida típico:

Usando as ferramentas do Lonbuilder, um projetista pode criar estes nós com um mínimo de treinamento e sem ter que inventar ferramentas ou aprender complexos protocolos de rede. As ferramentas Lonbuilder são usadas com os componentes Lonworks OEM como descrito a seguir:

- Escrever e compilar código das aplicações para o Neuron Chip usando Neuron C:

  Neuron C é uma linguagem de alto nível baseada em ANSI C . Escrever códigos para comunicar com outros nós na rede Lonworks requer somente duas linhas de código: uma para declarar uma variável na rede e outra para atribuir um valor a esta variável. O protocolo Lontalk permite que estas variáveis sejam compartilhadas entre qualquer grupo de nós da rede Lonworks. Interface com dispositivos de hardware complicados se torna tarefa simples. Por exemplo, controlar um triac necessita de apenas duas linhas de código: uma para declarar um dispositivo triac de I/O e a outra para atualizar o valor do atraso do triac. O software Lonbuilder inclui um editor que facilita a entrada de código. O complilador Neuron C compila o código da aplicação e como compilador e editor são integrados, os erros de sintaxe são automaticamente colocados em destaque no editor junto com as mensagens de erro do compilador.

- Implantar e testar as aplicações em multiplos nós da rede Lonworks: O software Lonbuider inclui um gerenciador de projeto que cria automaticamente um mapa de memória para todos os nós, carrega estes nós e começa a executar a aplicação-tudo isto com um único comando. Este software pode carregar o software da aplicação usando uma interface de alta performance ligada ao hardware Lonbuilder, ou seja da rede para os nó sob encomenda.

- Usando o Neuron C e o neuron emulador debugger para fazer o debug dos softwares das aplicações.

- Neuron C debugger permite ao projetista parar o programa em qualquer ponto, executar o programa passo a passo, e simbolicamente ver, modificar e monitorar as variáveis no programa.

LONBUILDER NEURON EMULADOR

O Lonbuilder Neuron Emulator é um processador que pode ser instalado no hack da estação de desenvolvimento. O emulador é a primeira ferramenta usada para desenvolver nós Lonworks. Ele é constituído pelo Neuron Chip 3150 com 64 Kbytes de RAM para simular o hardware e fazer o debug do software. O emulador permite ao projetista executar um software independente da procedência do hardware, assim o software pode ser desenvolvido antes mesmo do hardware estar disponível.

Cada Emulador pode acomodar até duas placas de expansão para testar o protótipo de I/O e o hardware do transceiver.

O Neuron emulador LonBuilder inclui um Neuron Chip 3150 com 64kbytes de memória RAM para código e dados e outra RAM de 64kbytes de controle. A RAM de controle é usada pelo Neuron C Debugger para setar breakpoints e detectar acessos à memória fora do mapa de memória da aplicação. Dois conectores de expansão permitem a comunicação do Neuron Chip e dos dispositivos I/O com dispositivos externos de I/O através dos transceivers. Outro transceivers é incluído para conectar o emulador com a rede da estação de desenvolvimento LonBuilder.

Um oscilador configuravel via software fornece o clock para o Neuron Chip. Este clock pode ser configurado para trabalhar em 625 kHz, 1,25 MHz, 2,5 MHz, 5 MHz ou 10 MHz.

Na seqüência, estaremos incorporando todo o levantamento de campo efetuado no Tefer, ao qual denominamos Catálogo.

Levantamento do sistema de transportadores e guindastes do Píer 1 & 2

TAG'S	LOCAL	LARGURA	CENTRO A	ELEVAÇÃO	VELOCIDADE	MATERIAL	TONELA-	TENSÃO	COBERTURA		METRAGEM
		(POL)	CENTRO	(GRAUS)	(M/SEG)		DAS/	(N/MM)			TOTAL (M)
			(M)				HORA		TIPO	CALIBRE	C/EMENDA
G-243	GUINDASTE 01	36"	9.45	0	1,22	ADUBO	300	6.1	330-B	3/16"X1/16"	22.50
G-247	GUINDASTE 02	36"	9.00	0	1,22	ADUBO	300	6.1	330-B	3/16"X1/16"	22.50
G-248	GUINDASTE 03	36"	9.00	0	1,22	ADUBO	300	6.1	330-B	3/16"X1/16"	22.50
G-250	GUINDASTE 04	36"	9.00	0	1,22	ADUBO	300	6.1	330-B	3/16"X1/16"	22.50
G-251	GUINDASTE 05	36"	9.00	0	1,22	ADUBO	300	6.1	330-B	3/16"X1/16"	22.50
T-01	SETOR 01	24"	33.81	0	2,91	ADUBO	300	6,3	220-B	3/16"X1/16"	70.50
T-03	SETOR 01	24"	33.81	0	2,91	ADUBO	300	6.3	220-B	3/16"X1/16"	70.50
T-02	SETOR 01	24"	68.55	0	2,93	ADUBO	300	6.3	220-B	3/16"X1/16"	139.50
T-04	SETOR 01	24"	68.55	0	2,93	ADUBO	300	6.3	220-B	3/16"X1/16"	139.50
T-05	SETOR 01	24"	133.00	0+8+0	2,47	ADUBO	300	21.8	220-B	3/16"X1/16"	275.00
T-06	SETOR 01	24"	133.00	0+8+0	2.47	ADUBO	300	21.8	220-B	3/16"X1/16"	275.00
T-07	SETOR 01	24"	133.00	0+8+0	2.47	ADUBO	300	21.8	220-B	3/16"X1/16"	275.00
T-08	SETOR 01	24"	133.00	0+8+0	2.47	ADUBO	300	21.8	220-B	3/16"X1/16"	275.00
T-09	SETOR 01	24"	7.20	0	2,96	ADUBO	300	5,2	220-B	3/16"X1/16"	17.50
T-10	SETOR 01	24"	7.20	0	2,96	ADUBO	300	5,2	220-B	3/16"X1/16"	17.50
T-11	SETOR 01	24"	7.20	0	2,83	ADUBO	300	5,4	220-B	3/16"X1/16"	17.50
T-12	SETOR 01	24"	7.20	0	2,83	ADUBO	300	5,4	220-B	3/16"X1/16"	17.50
T-13	SETOR 01	24"	27.20	0	2,93	ADUBO	300	5,6	220-B	3/16"X1/16"	57.50
T-15	SETOR 01	24"	27.20	0	2,93	ADUBO	300	5,6	220-B	3/16"X1/16"	57.50
T-17	SETOR 01	24"	27.20	0	2,93	ADUBO	300	5,6	220-B	3/16"X1/16"	57.50
T-19	SETOR 01	24"	27.20	0	2,93	ADUBO	300	5,6	220-B	3/16"X1/16"	57.50
T-14	SETOR 01	24"	89.80	0	2,93	ADUBO	300	6,8	220-B	3/16"X1/16"	184.00
T-16	SETOR 01	24"	89.80	0	2,93	ADUBO	300	6,8	220-B	3/16"X1/16"	184.00
T-18	SETOR 01	24"	89.80	0	2,93	ADUBO	300	6,8	220-B	3/16"X1/16"	184
T-20	SETOR 01	24"	89.80	0	2,93	ADUBO	300	6,8	220-B	3/16"X1/16"	184

Tabela 1 - Levantamento do sistema de transportadores e guindastes do Píer 1 & 2

Continuação do levantamento de transportadores e guindastes do Píer 1 & 2

TAG'S	LOCAL	LARGURA	CENTRO A	ELEVAÇÃO	VELOCIDADE	MATERIAL	TONELA-	TENSÃO	COBERTURA		METRAGEM
		(POL)	CENTRO	(GRAUS)	(M/SEG)		DAS/	(N/MM)			TOTAL (M)
			(M)				HORA		TIPO	CALIBRE	C/EMENDA
T-21	SETOR 01	24''	89.26	0	3,4	ADUBO	300	5,9	220-B	3/16''X1/16''	183.00
T-22	SETOR 01	24''	89.26	0	3.4	ADUBO	300	5.9	220-B	3/16''X1/16''	183.00
T-23	SETOR 01	24''	89.26	0	3.4	ADUBO	300	5.9	220-B	3/16''X1/16''	183.00
T-24	SETOR 01	24''	89.26	0	3.4	ADUBO	300	5.9	220-B	3/16''X1/16''	183.00
T-25	SETOR 01	24''	7.20	0	2,92	ADUBO	300	4,9	220-B	3/16''X1/16''	17,5
T-26	SETOR 01	24''	7,2	0	2,92	ADUBO	300	4.9	220-B	3/16''X1/16''	17,5
T-27	SETOR 01	24''	7,2	0	2,92	ADUBO	300	4.9	220-B	3/16''X1/16''	17,5
T-28	SETOR 01	24''	7,2	0	2,92	ADUBO	300	4.9	220-B	3/16''X1/16''	17,5
T-29	SETOR 01	24''	26.79	0	3	ADUBO	300	5,5	220-B	3/16''X1/16''	57.50
T-31	SETOR 01	24''	26.79	0	3	ADUBO	300	5,5	220-B	3/16''X1/16''	57.50
T-33	SETOR 01	24''	26.79	0	3	ADUBO	300	5,5	220-B	3/16''X1/16''	57.50
T-35	SETOR 01	24''	26.79	0	3	ADUBO	300	5,5	220-B	3/16''X1/16''	57.50
T-30	SETOR 01	24''	82.85	0	2,98	ADUBO	300	6.7	220-B	3/16''X1/16''	184.00
T-32	SETOR 01	24''	82.85	0	2,98	ADUBO	300	6,7	220-B	3/16''X1/16''	184
T-34	SETOR 01	24''	82.85	0	2,98	ADUBO	300	6,7	220-B	3/16''X1/16''	184
T-36	SETOR 01	24''	82.85	0	2,98	ADUBO	300	6,7	220-B	3/16''X1/16''	184
BIGURRILHO	SETOR 01	24''	18,55	0	0,23	ADUBO	300	9,5	220-B	3/16''X1/16''	40.00
G-244	GTE 01 - SETOR 02	36''	9,1	0	2,63	ENXOFRE	800	7,1	330-B	3/16''X1/16''	22
G-242	GTE 02 - SETOR 02	36''	9,1	0	2,63	ENXOFRE	800	7,1	330-B	3/16''X1/16''	22
G-241	GTE 03 - SETOR 02	36''	9,1	0	2,63	ENXOFRE	800	7,1	330-B	3/16''X1/16''	22
G-252	GTE 04 - SETOR 02	36''	9,1	0	2,63	ENXOFRE	800	7,1	330-B	3/16''X1/16''	22
G-254	GTE 05 - SETOR 02	36''	9,1	0	2,63,	ENXOFRE	800	7,1	330-B	3/16''X1/16''	22
T - 01	SETOR 02	42''	73,36	0	2,63	ENXOFRE	800	14,5	330-B	3/16''X1/16''	150,5
T - 02	SETOR 02	42''	73,21	0	2,49	ENXOFRE	800	10,5	330-B	3/16''X1/16''	150.50
T - 03	SETOR 02	42''	97,58	6	2,1	ENXOFRE	800	22,3	330-B	3/16''X1/16''	206
T - 04	SETOR 02	42''	103,68	6	2,4	ENXOFRE	800	23,1	330-B	3/16''X1/16''	214.00
T - 05	SETOR 02	42''	127,55	7	2,3	ENXOFRE	800	31,2	330-B	3/16''X1/16''	266

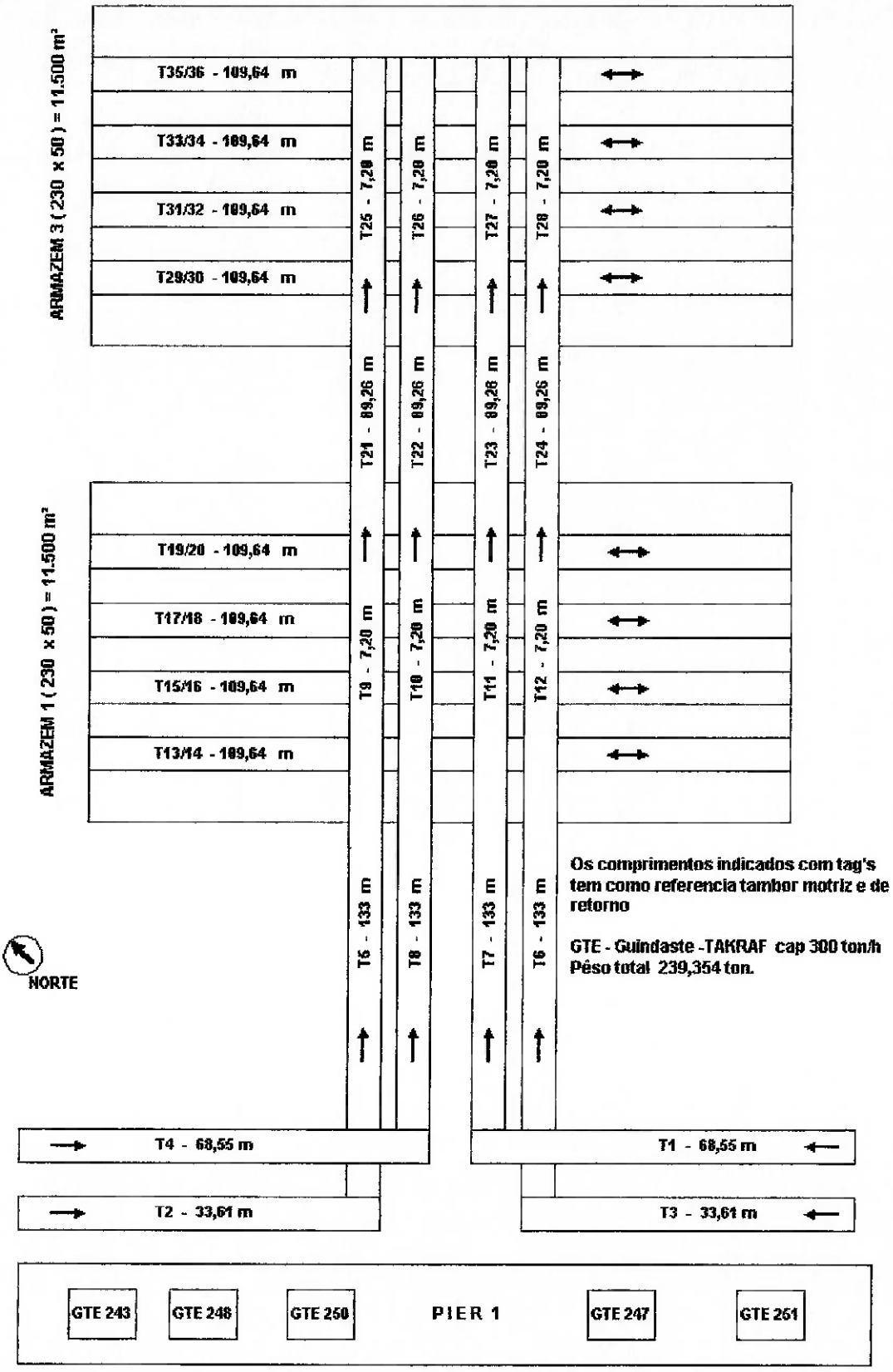
Tabela 2 - Continuação do levantamento de transportadores e guindastes do Píer 1 & 2

Finalização do levantamento de transportadores e guindastes do Píer 1 & 2

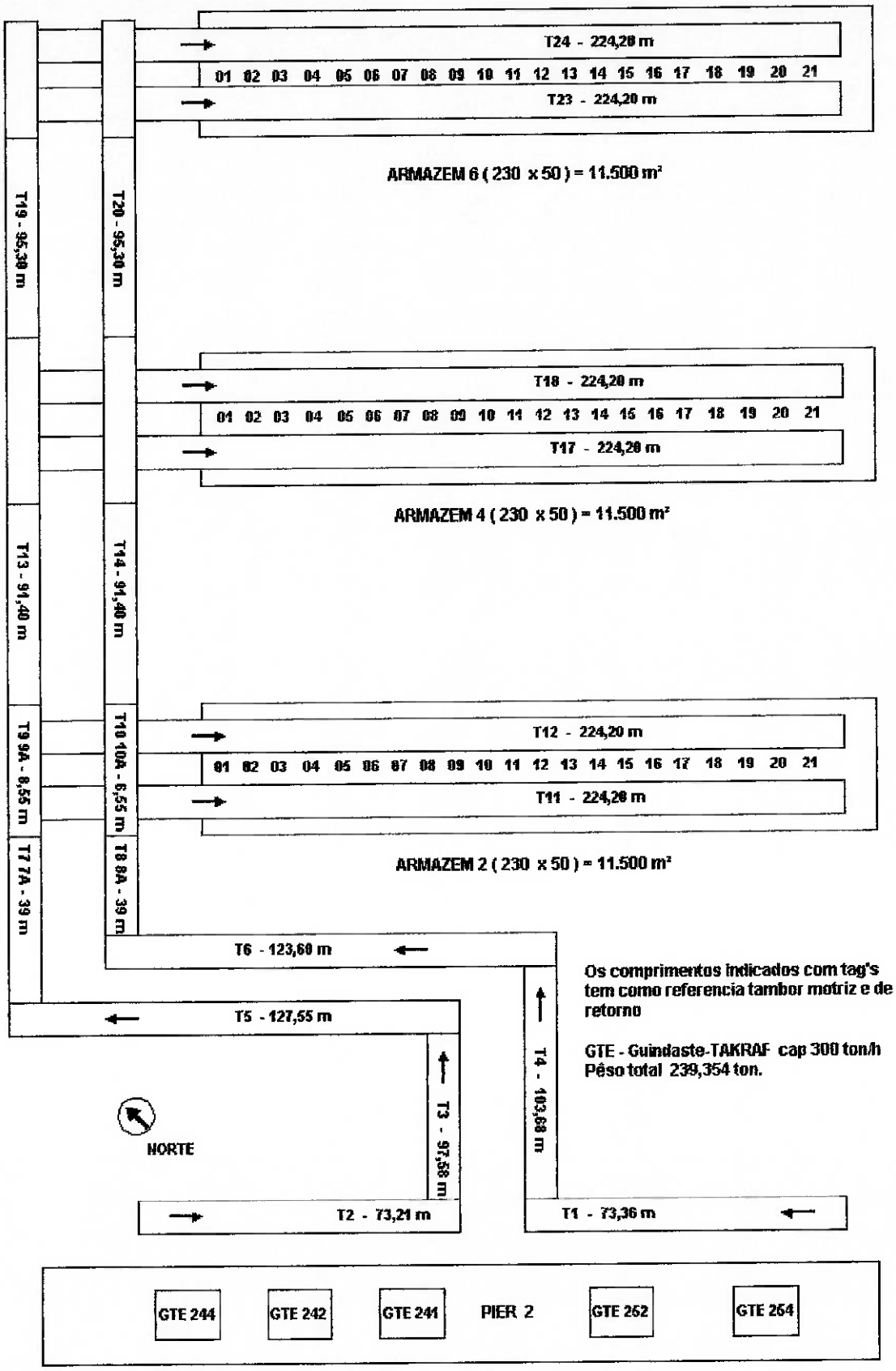
TAG'S	LOCAL	LARGURA (POL)	CENTRO A CENTRO (M)	ELEVAÇÃO (GRAUS)	VELOCIDADE (M/SEG)	MATERIAL	TONELADAS/ HORA	TENSÃO (N/MM)	COBERTURA		METRAGEM TOTAL (M)
									TIPO	CALIBRE	C/EMENDA
T - 06	SETOR 02	42''	123,6	7	2,38	ENXOFRE/ADUB	800	29,5	330-B	3/16''X1/16''	259
T - 07	SETOR 02	42''	39	7	2,52	ENXOFRE/ADUB	800	14,3	330-B	3/16''X1/16''	87
T - 08	SETOR 02	42''	39	7	2,52	ENXOFRE/ADUB	800	14,3	330-B	3/16''X1/16''	87.00
T - 09	SETOR 02	42''	6,55	0	2,52	ENXOFRE/ADUB	800	5,6	330-B	3/16''X1/16''	17
T - 10	SETOR 02	42''	6,55	0	2,42	ENXOFRE/ADUB	800	5,6	330-B	3/16''X1/16''	17.00
T - 11	SETOR 02	42''	224.20	0+14+0	2,39	ADUBO	800	28,7	330-B	3/16''X1/16''	466
T - 12	SETOR 02	42''	224,2	0+14+0	2,39	ADUBO	800	28,7	330-B	3/16''X1/16''	466
T - 13	SETOR 02	42''	91,4	4	2,39	ENXOFRE/ADUB	800	18,1	330-B	3/16''X1/16''	194
T - 14	SETOR 02	42''	91,4	4	2,39	ENXOFRE/ADUB	800	18,1	330-B	3/16''X1/16''	194
T - 15	SETOR 02	42''	6,5	0	2,8	ENXOFRE/ADUB	800	5,7	330-B	3/16''X1/16''	17.00
T - 16	SETOR 02	42''	6,5	0	2,8	ENXOFRE/ADUB	800	5,7	330-B	3/16''X1/16''	17
T - 17	SETOR 02	42''	224,2	0+14+0	2,39	ADUBO	800	28,7	330-B	3/16''X1/16''	466
T - 18	SETOR 02	42''	224,2	0+14+0	2,39	ADUBO	800	28,7	330-B	3/16''X1/16''	466
T - 19	SETOR 02	42''	95,3	3	2,39	ENXOFRE/ADUB	800	16,7	330-B	3/16''X1/16''	201
T - 20	SETOR 02	42''	95,3	3	2,39	ENXOFRE/ADUB	800	16,7	330-B	3/16''X1/16''	201
T - 21	SETOR 02	42''	6,4	0	2,5	ENXOFRE/ADUB	800	5,5	330-B	3/16''X1/16''	17.00
T - 22	SETOR 02	42''	6,4	0	2,5	ENXOFRE/ADUB	800	5,5	330-B	3/16''X1/16''	17.00
T - 23	SETOR 02	42''	224,2	0+14+0	2,39	ADUBO	800	28,7	330-B	3/16''X1/16''	466
T - 24	SETOR 02	42''	224,2	0+14+0	2,39	ADUBO	800	28,7	330-B	3/16''X1/16''	466
T - 25	SETOR 02	42''	27,4	8	2,61	ENXOFRE/ADUB	800	12,7	330-B	3/16''X1/16''	64
T - 26	SETOR 02	42''	27,4	8	2,61	ENXOFRE/ADUB	800	12,7	330-B	3/16''X1/16''	64
CT-CAR - T 11	SETOR 02	42''	2,85	0	2,4	ADUBO	800	5,6	330-B	3/16''X1/16''	
CT-CAR - T 12	SETOR 02	42''	2,85	0	2,4	ADUBO	800	5,6	330-B	3/16''X1/16''	
CT- CAR - T 17	SETOR 02	42''	2,85	0	2,4	ADUBO	800	5,6	330-B	3/16''X1/16''	
CT - CAR - T 18	SETOR 02	42''	2,85	0	2,4	ADUBO	800	5,6	330-B	3/16''X1/16''	
CT-CAR - T 23	SETOR 02	42''	2,85	0	2,4	ADUBO	800	5,6	330-B	3/16''X1/16''	
CT - CAR - T 24	SETOR 02	42''	2,85	0	2,4	ADUBO	800	5,6	330-B	3/16''X1/16''	

Tabela 3 - Finalização do levantamento de transportadores e guindastes do Píer 1 & 2

TEFER - Terminal de Fertilizantes - Pier 1



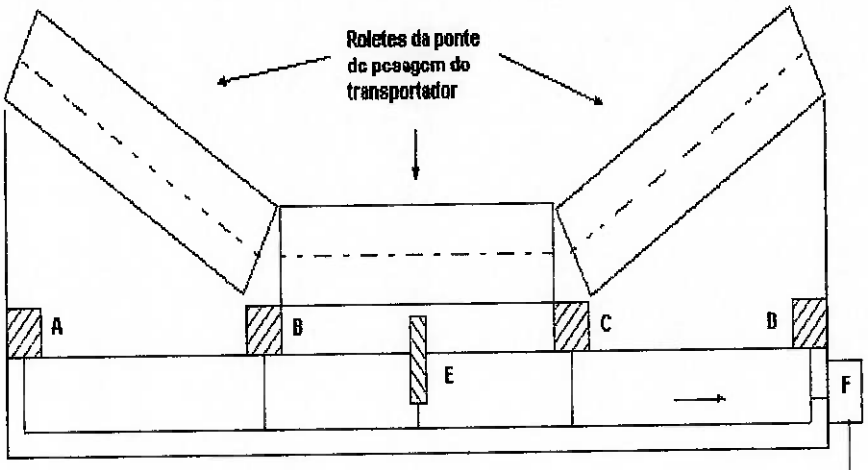
TEFER - Terminal de Fertilizantes - Pier 2



Os comprimentos indicados com tag's tem como referencia tambor motriz e de retorno

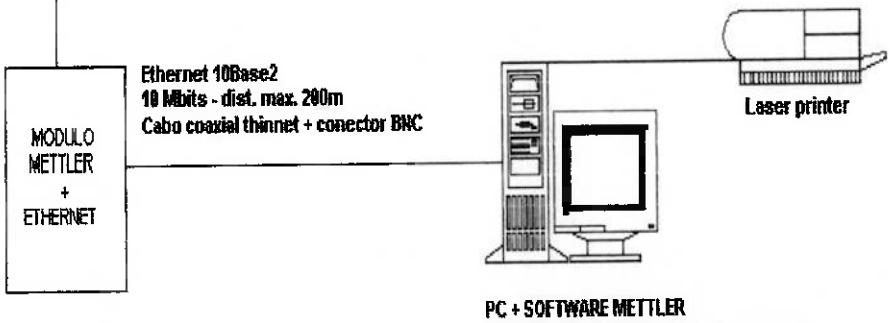
GTE - Guindaste-TAKRAF cap 300 ton/h
Pêso total 239,354 ton.

DIAGRAMA DE INTERLIGAÇÃO DO SISTEMA DE PESAGEM CONTINUA



A , B, C, D = Celula de carga com sinal de saida de 2mV - IP 65
E = Gerador de pulsos 2 pulsos/mm
F = Caixa de junção eletrica

Cabo 10 x 28 AWG Cobre
PE + shieldado + dreno



PC + SOFTWARE METTLER
GERANDO MASSA DE DADOS DA PESAGEM

Descrição técnica alguns módulos de automação.

Modulo AI-10

É composto por 02 entradas analógicas de 16 bits e podem monitorar 0 – 20 mA, 0 – 10 V e entradas resistivas de 100Ω à 15kΩ.

Operam com tensão entre 16 à 30 VAC ou VDC, mantendo-se a mesma tensão de alimentação dos sensores.

Modulo AO-10

É composto por 02 saídas analógicas de 12 bits e podem controlar 0 – 20 mA, 0 – 10 V atuadores.

Operam com tensão entre 16 à 30 VAC ou VDC, mantendo-se a mesma tensão de alimentação dos atuadores.

Modulo DI-10

É composto por 04 entradas digitais podem monitorar contatos secos ou entradas de voltagens 0 – 32 VDC, mantendo status por led's independentes.

Operam com tensão entre 16 à 30 VAC ou VDC, mantendo-se a mesma tensão de alimentação dos sensores.

Modulo DO-10

É composto por 04 saídas digitais configuráveis por DIP switch.

Operam com tensão entre 16 à 30 VAC ou VDC, mantendo-se a mesma tensão de alimentação dos atuadores.

Modulo DIO-10

É composto por 02 entradas digitais e 02 saídas a rele. As entradas digitais podem monitorar contato seco ou entradas de tensão em 31 VDC.

As duas saídas a rele mantém continuamente 2 A e pico de 6 A , 30 VAC ou VDC.

Cada posição é habilitada por DIP swith Manual /Desl/Automático.

Operam com tensão entre 16 à 30 VAC ou VDC, mantendo-se a mesma tensão de alimentação dos atuadores/sensores.

Para que possamos representar graficamente nosso projeto em rede LonWorks, iremos utilizar uma analogia com circuitos a rele propiciando ao leitor uma facilidade no algoritmo de programação e lógica de operação.

De posse da planta dos transportadores do Tefer Píer 2 poderemos observar a seqüência de construção de uma dos lados do sistema, como segue:

Píer 2 → Armazém 2
T2→T3→T5→T7A→T9A→(T12 ou T11) → moega de 01 à 21

Pier 2 → Armazém 4
T2→T3→T5→T7A→T9A→T13→T15→(T18 ouT17) → moega de 01 à 21

Pier 2 → Armazém 6
T2→T3→T5→T7A→T9A→T13→T15→T19 →T21→(T24 ouT23) → moega de 01 à 21.

Onde T significa Transportadora.
Iremos desenvolver nosso projeto na rota Píer 2 → Armazém 2

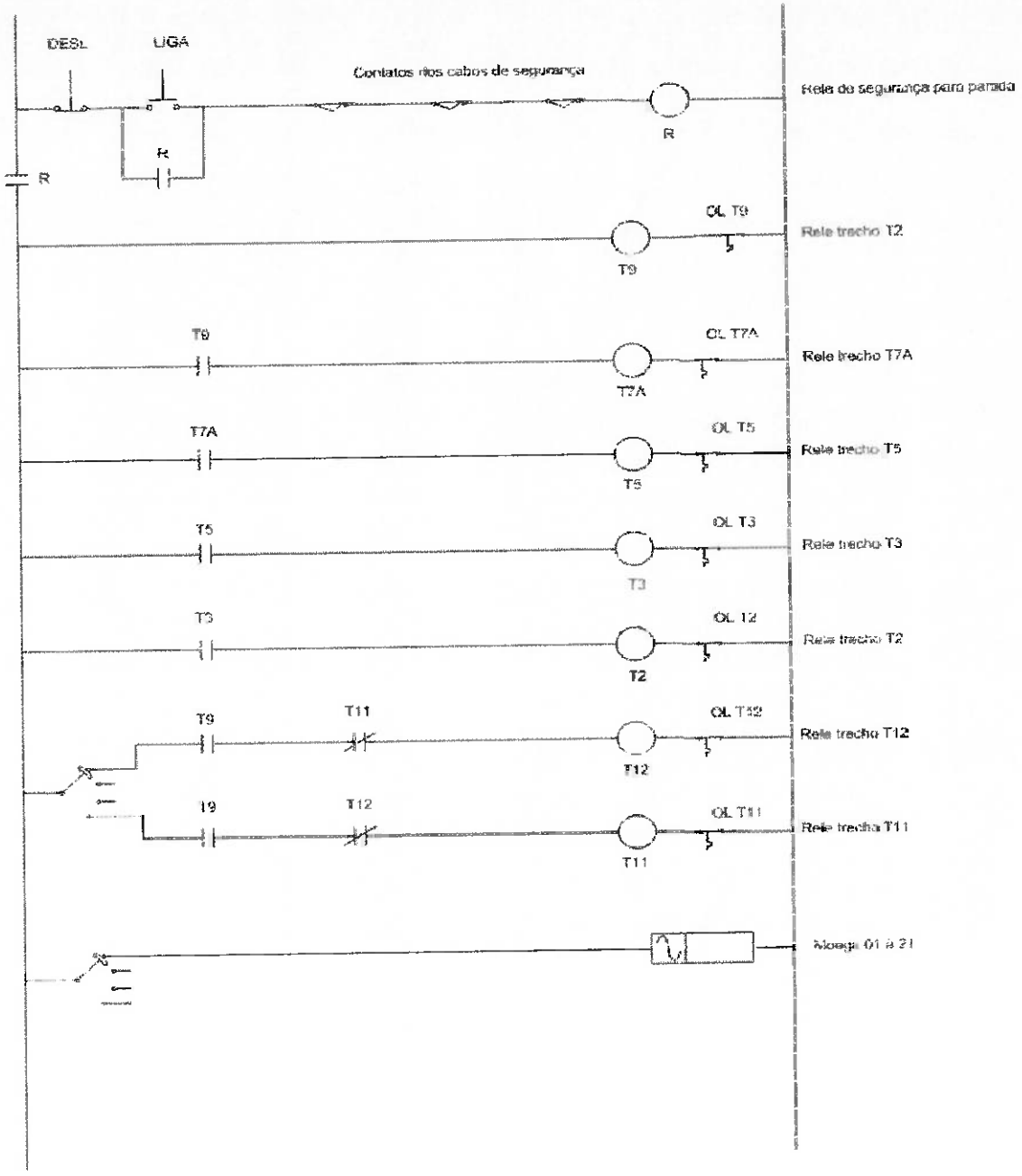
Com esta definição poderemos comentar a lógica de funcionamento do sistema de transportadores, durante o processo de descarga de graneis.

Para que o sistema possa entrar em operação todos os seus componentes devem estar habilitados, para que possam ser energizados impondo assim o inicio da operação de descarga pelos transportadores a comando do operador.

Temos que comentar, que obedecendo a seqüência de construção, em caso de parada de um transportador todos os anteriores, pararão. A menos no caso de acionamento do cabo de segurança ao longo de todos os transportadores, o sistema para por completo, para averiguação do ocorrido.

O esquema em lógica por rele exemplifica:

Diagrama elétrico do sistema Pier 2 - Armazem 2



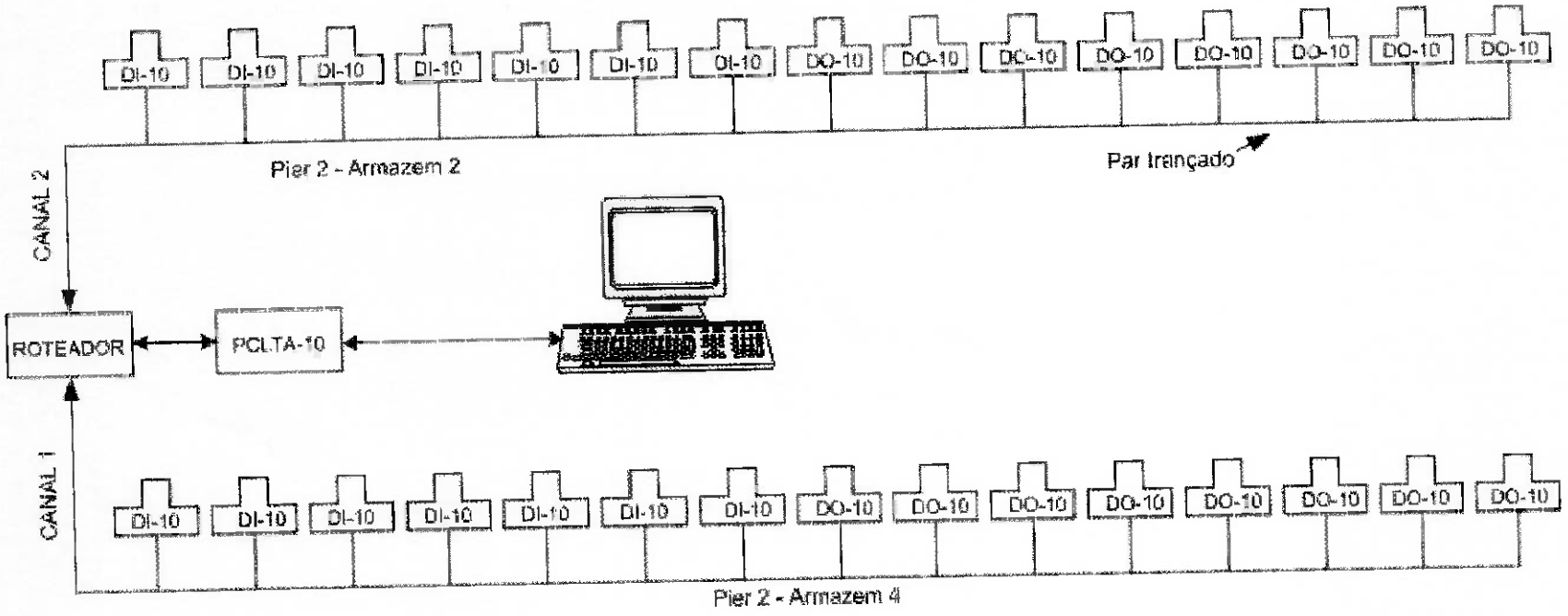
Analisando o diagrama acima temos:

O comando de ligar, permitido pelos contatos dos cabos de segurança não atuados (fechados), dando inicio a operação do sistema, sendo o anterior partindo o posterior. Está lógica de circuito, nos permite na ocorrência de falhas ou atuação dos dispositivos de proteção, desligarmos somente os transportadores em falhas e seus anteriores. Com auxilio de temporizadores, podemos temporizar as partidas posteriores, somente após os motores anteriores estarem em regime nominal de rotação.

Do diagrama temos, 29 relês (T9..T11 e moega 01 ...21) o que em representação LonWorks serão pontos de *saídas digitais* da rota e 28 contatos secos oque nos serão pontos de *entradas digitais*.

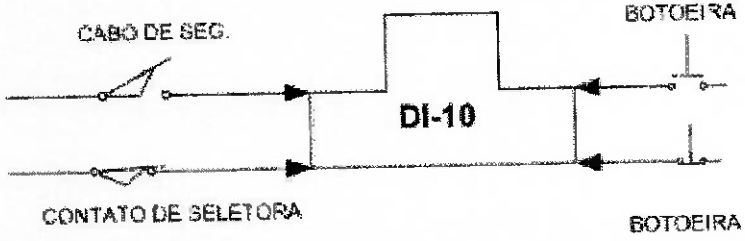
Podemos agora identificar, a economia na aplicação da Tecnologia LonWorks, pois todos os sensores (contatos dos cabos de segurança, alinhadores de esteira) e atuadores (solenóides), não necessitarão de cabos elétricos até o painel de comando e sim até um modulo LonWorks (entradas ou saídas digitais, no caso em estudo) interligando por sua vez com um cabo par trançado estes módulos ao painel central. Favorecido ainda pela inteligência dos Neuron chips dos módulos.

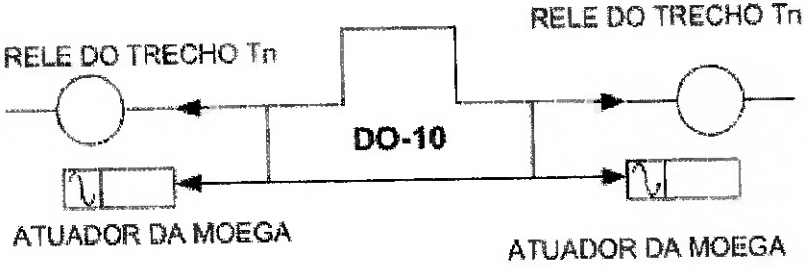
Para melhor visualização do descrito e da migração de tecnologia, iremos mostrar o mesmo circuito em Tecnologia LonWorks com seus módulos DI-10 e DO-10, já descrito anteriormente. Lembrando que todo o entertravamento é desenvolvido com a aplicação do software.



Estamos representanddo duas rotas de transportadores, conectados a um roteador que pela interface a um PC.

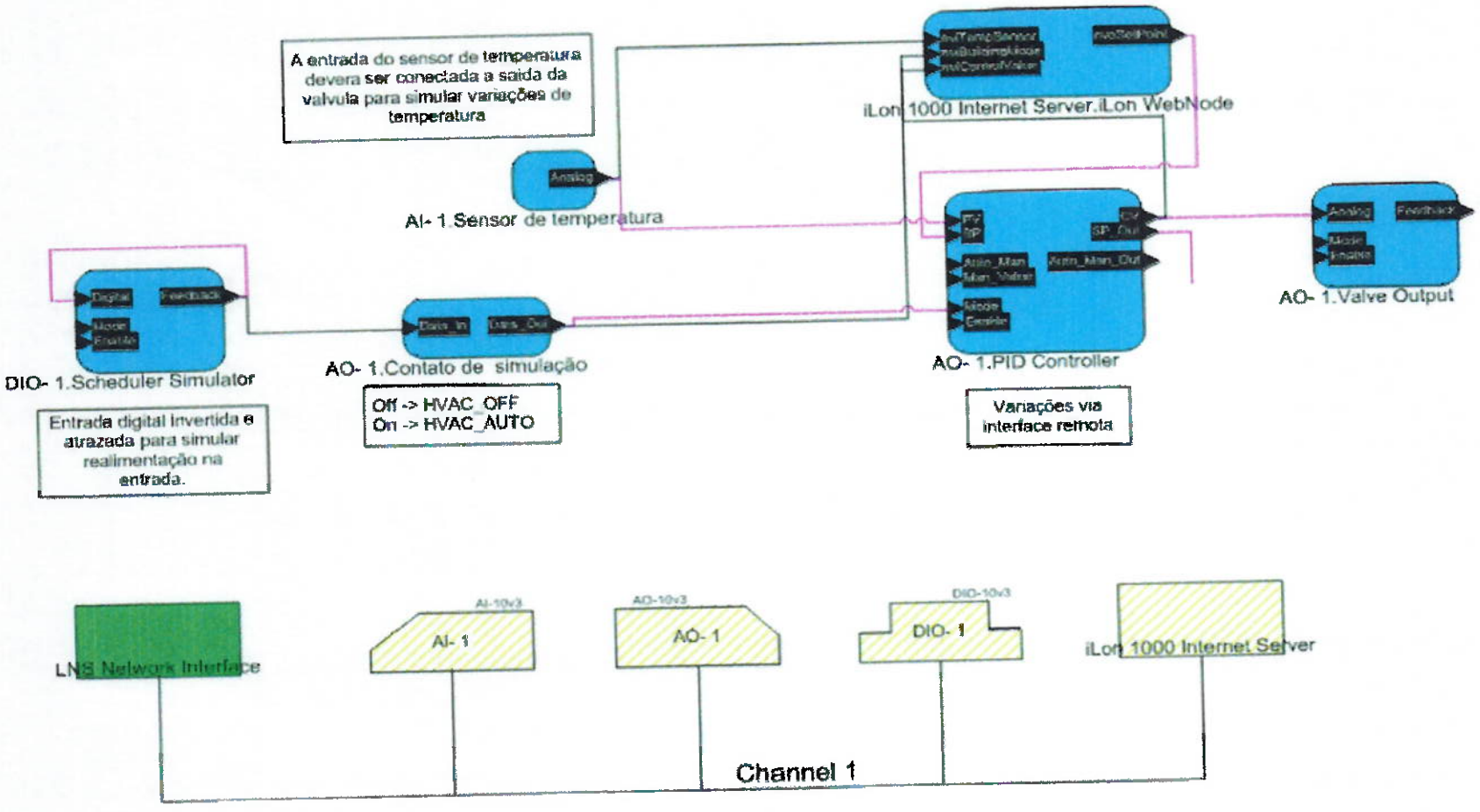
Cada modulo DI-10, recebe 04 saidas
Cada modulo DO-10, externa 04 saidas





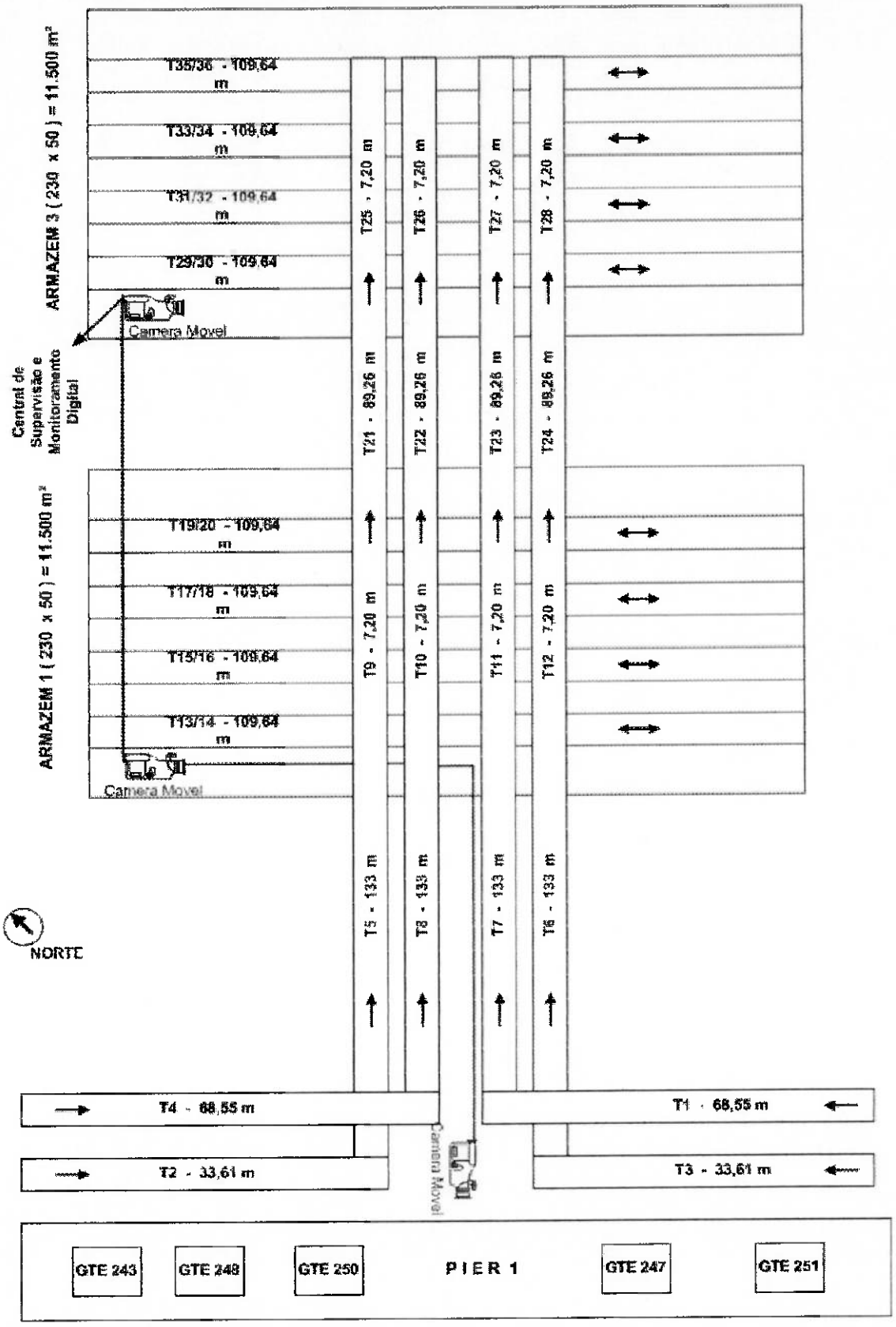
A

Representação de um modulo da rede em LonWorks



Vista em planta Tefer do sistema de monitoramento

Posicionamento do Sistema de monitoramento digital do Tefer



Capitulo V

V.I. Conclusão

No modelo proposto, entendemos que a mudança de tecnologia de automação e controle do sistema de descarga de graneis trará um excelente incremento de qualidade e confiabilidade no gerenciamento operacional, sendo esse modelo uma ferramenta de administração do terminal.

Estamos propondo, é claro, um passo teórico embasado em observações de campo e vivências do cotidiano operacional do Tefer. A coluna mestra desse modelo está focada no sistema de balanças que irão retratar por intermédio do software especifico, *quanto e de onde recebemos e para quem e como distribuímos* . Todo o restante do sistema de automação estará voltado à segurança intrínseca dos operadores e do processo de descarga de granéis.

Quando propomos a supervisão por câmeras de vídeo interna e externamente ao Tefer, viabiliza-se um controle eficaz do movimento logístico em sua fase operacional e de manutenção, sendo esta, parceira indispensável para esse modelo.

Temos também plena consciência do grande desafio que existe em adaptar um terminal a uma nova tecnologia, lembrando que nada se fez quanto à chuva, que por sua vez, simplesmente, aborta determinadas operações de descargas, tendo esta ou outra tecnologia de automação implantada.

Ressaltamos a real necessidade da educação continuada neste processo, pois temos certeza que com a melhor formação de nossos colaboradores retornaremos com melhores índices de rentabilidade final, diminuindo também o retrabalho e o número de horas/ máquinas paradas.

Fica-nos a certeza, que não estamos no final e sim no inicio de grande trabalho em um terminal brasileiro.

ANEXOS:

DADOS ESTATÍSTICOS DO PORTO DE SANTOS

MOVIMENTO ANUAL (em toneladas)

	1.997	1.998	1.999
TOTAL	38.472.130	39.940.386	42.675.507
EXPORTAÇÃO	17.791.815	19.401.126	24.264.690
IMPORTAÇÃO	20.680.315	20.539.260	18.410.817

MOVIMENTO DE NAVIOS (em unidades)

	1.997	1.998	1.999
TOTAL	3.975	4.350	4.018

MOVIMENTO DE CONTÊINERES (em unidades)

	1.997	1.998	1.999
TOTAL	580.592	564.948	546.972

PRINCIPAIS MERCADORIAS MOVIMENTADAS (em toneladas)

MERCADORIA	1.997	1.998	1.999
AÇÚCAR (E)	2.378.348	3.668.613	6.965.010
CAFÉ (E)	702.750	467.898	524.463
SUCO DE LARANJA (E)	1.175.488	932.353	1.053.110
SOJA EM GRÃO (E)	1.866.622	2.089.040	2.560.863
FARELOS (E)	1.318.186	1.700.687	2.157.873
PAPEL (E)	194.825	210.966	311.625
TRIGO (I)	1.006.624	1.537.145	1.713.105
SAL (CABOTAGEM)	555.109	662.726	690.142
ADUBO (I)	2.402.846	1.304.185	1.880.331
CARNE (E)	84.851	94.732	144.043
GLP (E e I)	1.061.862	976.421	989.041
ÓLEO DIESEL (E)	530.178	1.407.940	1.369.445

E = Cargas de Exportação
I = Cargas de Importação

CARACTERÍSTICAS GERAIS DO PORTO DE SANTOS

Área

	Total	7.765.100 m2
Área do Porto	Margem Direita	3.665.800 m2
	Margem Esquerda	4.099.300 m2

Cais

	CODESP	53
Número de Berços	Terminais Privativos	11

Local	**Extensão**	**Profundidade**
Total	13.013 m	Entre 5.0 e 13.5 m
CODESP	11.600 m	Entre 6.6 e 13.5 m
Terminais Privativos	1.413 m	Entre 5.0 e 13.5 m

Armazéns

Área Total (incluindo silos)	499.701 m2

Pátios

Área Total	981.603 m2

Tanques

255 tanques	585.111 m3

Dutos

Ilha do Barnabé, Saboó e Alemoa	55.676 m

Linhas Férreas

Ferrovia	186.784 m
Outras (guindastes, portêineres, transtêineres)	13.217 m
Total	201.183 m

Linhas Férreas

Descrição	Efetivo
Com vinculo	1.991

COMPARATIVO DE RENDIMENTO OPERACIONAL

Local	Natureza da Carga	Produtividade t/navio x dia				Variação %			
		1.996	1.997	1.998	1.999	97/96	98/97	99/98	99/96
Alamoa (4 berços)	LG (diversos)	7.960	8.300	9.624	9.251	4,3	16,0	(3,9)	16,2
Ilha Barnabé (2 berços)	LG (diversos)	1.900	2.761	3.677	3.687	45,3	33,2	0,3	94,1
Saboó (5 berços)	LG (sucos) SG (diversos)* CC CC(ro/ro) CG(ro/ro)	3.480 1.170 3.610 3.200 1.850	4.035 1.857 4.428 4.234 2.034	3.279 1.573 5.636 5.257 2.608	5.272 2.130 4.539 4.107 1.847	15,9 58,7 22,7 32,3 9,9	(18,7) (15,3) 27,3 24,2 28,2	60,8 35,4 (15,5) (21,9) (29,2)	51,5 82,1 25,7 28,3 (0,2)
Valongo ao Arm. 12 (6 berços)	CG	530	625	697	797	17,9	11,5	14,3	50,4
Arm. 12^ ao 23 (9 berços)	CG CC SG (sal) SG (açúcar) SG (trigo)	550 3.180 2.020 1.470 1.120	693 3.348 2.404 3.217 1.449	944 5.289 2.279 6.167 1.816	1.536 5.638 4.089 10.587 1.873	26,0 5,3 19,0 118,8 29,4	36,2 58,0 (5,2) 91,7 25,3	62,7 6,6 79,4 71,7 3,1	179,3 77,3 102,4 620,2 67,2
Frigorifico à Mortona (5 berços)	CG SG (trigo)	550 -	697 -	849 -	1.050 2.471	26,7 -	21,8 -	23,7 -	90,9 -
Arm. 29 ao 35 (12 berços)	CG CG (ro/ro) CC (ro/ro) CC (por/cont) CC (conv.)	745 2.510 2.750 3.300 1.640	1.071 3.234 3.844 3.775 2.428	1.156 2.868 4.090 4.777 2.678	1.152 3.122 5.383 4.693 2.399	43,8 28,8 39,8 14,4 48,0	7,9 (11,3) 6,4 26,5 10,3	(0,3) 8,9 31,6 (1.8) (10,4)	54,6 24,4 95,7 42,2 46,3
Ferry Boat ao Arm. 39 (6 berços)	SG (div.) **	4.550 4.490	6.839 5.351	5.904 9.134	7.194 9.106	50,3 19,2	(13,7) 70,7	21,8 (0,3)	58,1 102,8
Tecon (3 berços)	CC	4.500	5.654	6.830	9.436	25,6	20,8	38,2	109,7
Tefer (2 berços)	SG (diversos)	3.100	3.215	2.962	3.401	3,7	(7,9)	14,8	9,7

Fonte: Sistema de Informações Operacionais 0 SIOP
Companhia Docas do Estado de São Paulo – CODESP

Legenda:
LG = líquidos a Granel
SG = Sólidos a Granel
CG = Carga Geral
CC = Contêiner
SG (diversos):
* Carvão, cimento, enxofre, adubo, trigo, barrilha e minérios
** Grãos, pellets, trigo e açúcar

Comparativo do Movimento Físico	Mil toneladas				Variação %			
	1.996	1.997	1.998	1.999	97/96	98/97	99/98	99/96
Total do Porto (milhares de toneladas)	36.339	38.472	39.940	42.675	5,87	3,82	6,85	17,43

Melhores Movimentos Anuais de Enxofre	
Ano	Tonelagem
1.997	1.353.887
1.998	1.351.254
1.999	1.266.751
1.996	1.146.968
1.995	1.114.845

Melhores Movimentos Anuais de Adubo	
Ano	Tonelagem
1.997	2.402.846
1.996	2.255.109
1.987	2.206.682
1.998	2.065.189
1.994	2.043.269

Forma como o Terminal de Verbrugge disponibiliza suas atividades na Internet
Fonte: endereço eletrônico http:\\www.verbrugge.nl

Verbrugge Terminals

Verbrugge Terminals operates its port terminal in the port of Terneuzen, the Netherlands. Its main specialisations are forest products (1.4 mm ton p.a.) and dry bulk (appr. 1.5 mm tons p.a.) The service package includes:

- stevedoring
- warehousing
- agencies
- chartering by road, rail, water
- towage and salvage
- in-house customs services

The Port of Terneuzen

Situated in the estuary of the river Scheldt, the port of Terneuzen hosts many multinational companies.

Ideally located to serve as a European hub, the port offers an advanced infrastructure, excellent connections to the major markets, and, most importantly, no bottlenecks

Handling the bulk of the world

Verbrugge Bulk Terminal: Your European Hub!

The Verbrugge Bulk Terminal was constructed in 1986 and is still one of Europe's most modern terminals. The focus has been on adding value to your products; screening, breaking, crushing and bagging services are offered in addition to stevedoring and warehousing services. Over the years, great strides have been made in dust prevention.

Verbrugge is well equiped to arrange through-transportation of your products using all existing modes:

- by road, using its own fleet
- by rail, loading block trains using its new loading facility
- by water, both inland and short sea using its own chartering departments.
- Cutting-edge in-house information services are provided.
- Offering a total logistical package enables us to integrate your supply chain and thereby be highly competitive.

Handling Capacity

Berth Capacity	Length	254m
	Width	34m
	Draught	12.40m

Vessel discharge capacity	1200 tonnes/hour
Vessel loading capacity	600 tonnes/hour
Truck/rail loading capacity	4000 tonnes/shift
2 gantry cranes with reach of 38m. each	

Product Treatment

Screening simultaneous undersize and oversize (0 to 10mm)	150 tonnes/hour
Breaking	200 tonnes/hour
Crushing	150 tonnes/hour
Bagging (FIBC's)	300 tonnes/day

Dry Bulk Storage Capacity

Up to 30 different commodities in covered warehousing	230.000 tonnes
Open storage	70.000 tonnes

Port Related Activities

Ships agency services are provided

Verbrugge Marine handles close to 2000 sea-going vessels per year. In addition, in-house customs services are provided.
For more information you can contac:

Verbrugge International B.V.
att. Jo Vandeputte
P.O. Box 5
4530 AA Terneuzen
The Netherlands
Tel.:+31 (0)115 – 646330
Fax.:+31 (0)115 – 646380
Telex: 55023
E-mail: jo.vandeputte@verbrugge.nl

Bibliografia

- Guia Internet de Conectividade – Editora Senac SP.

Referência de pesquisas:

- TEC Tecnologia Engenharia e Comércio de Componentes Eletrônicos – São Paulo
- Histórico administrativo da Fertimport
- Departamento Técnico da Godyear
- Departamento Técnico da Corbras
- Departamento Técnico Tomé Engenharia

Endereços eletrônicos:

- www.echelon.com - Pesquisa da tecnologia
- www.lonmark.com - Pesquisa do protocolo
- www.portodesantos.com.br - Pesquisa sobre o Porto de Santos
- www.verbrugge.nl - Pesquisa sobre o Terminal Verbrugge
- www.ebsbulk.nl - Pesquisa sobre o Terminal E.B.S. Bulk

TOLEDO

9270
BALANÇA INTEGRADORA

Para Materiais a Granel

- Mineração, siderurgia, pedreira, papel e celulose
- Armazéns, portos e cooperativas
- Usinas de açúcar, destilarias de álcool e outras

9270 BALANÇA INTEGRADORA TOLEDO

A Balança Integradora Toledo Modelo 9270 é adequada para transportadores de correia de 16 a 72 polegadas e capacidades de até 6000 t/h (*). Consiste de três componentes básicos: ponte de pesagem, gerador de pulsos e painel de controle.
A ponte de pesagem, de concepção modular, possui uma ou duas células de carga, dependendo da largura do transportador de correia. Em aplicações a velocidades elevadas ou mais exigentes quanto à exatidão, até quatro pontes de pesagem podem ser dispostas em série num mesmo transportador de correia.
As células de carga, em virtude do inovador projeto da ponte de pesagem, reagem apenas às forças verticais transmitidas pelo rolete de pesagem (correspondentes à carga de material na correia) e nunca às forças de atrito entre rolos e correia, forças laterais e cargas descentradas.
Em operação, os sinais das células de carga (peso) e do gerador de pulsos (velocidade da correia transportadora), posicionado nas proximidades da ponte de pesagem, são utilizados pelo painel de controle para obtenção do fluxo de material passante que, integrado em relação ao tempo, resulta na indicação da quantidade de material transportado.

(*) Capacidades diferentes, sob consulta.

CARACTERÍSTICAS TÉCNICAS

PONTE DE PESAGEM

- Totalmente eletrônica: utiliza uma ou duas células de carga, dependendo da largura do transportador de correia.
- Transferência direta, sem alavancas, da carga de material na correia para as células de carga: respostas rápidas às forças verticais e às variações instantâneas de carga e, o mais importante, com excelente repetibilidade.
- Construção robusta: deflexão estrutural desprezível.
- Modular: até quatro pontes de pesagem podem ser instaladas em série com um único Painel de Controle, proporcionando também grande flexibilidade em caso de necessidade futura de maior exatidão.
- Versátil: um futuro aumento significativo na capacidade do transportador implica somente na troca das células de carga por outras de capacidade nominal superior.
- Instalação: simples, entre as longarinas do transportador, requerendo apenas quatro furos passantes.
- Nivelamento/Alinhamento: fáceis, precisos e permanentes com buchas de ajuste e parafusos de trava, dispensando calços e assemelhados.
- Perfil baixo: reduzido espaço necessário entre a correia de carga e a de retorno.
- Danos por sobrecarga: nenhum; células de carga protegidas por limitadores mecânicos.
- Grampos ajustáveis: facilidade de fixação do rolete de pesagem.
- Buchas, grampos, porcas e parafusos zincados: maior resistência à corrosão.
- Excelente estabilidade operacional: área de acúmulo de pó/material prejudicial à operação limitada à ocupada pelo próprio rolete de pesagem, e inexistência de pontos que, com a queda do material transportado, possam vir a provocar o travamento da ponte de pesagem.
- Manutenção *Zero*: inexistência de alavancas, elementos móveis sujeitos a desgaste como cutelos, coxins e munhões, e de limitação de movimento como varões e guias paralelas.
- Trava de segurança: proteção das células de carga durante o transporte ou manutenção do transportador de correia.
- Pré-montagem e testes de fábrica: tempo de instalação reduzido ao máximo.

CÉLULAS DE CARGA

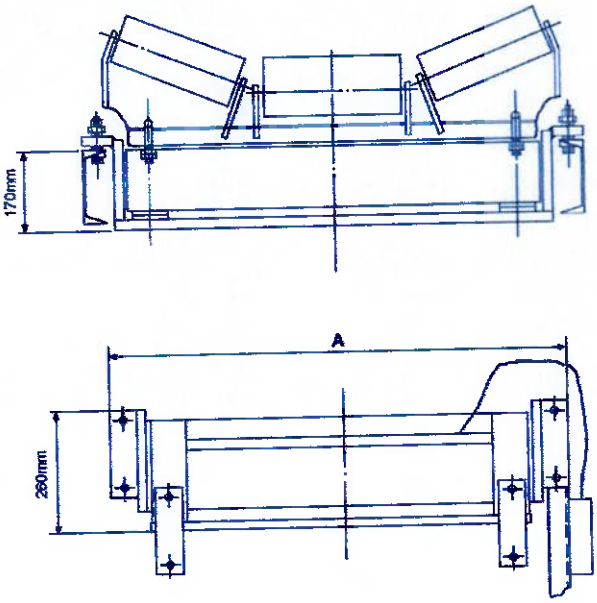
- Flexibilidade: disponíveis em várias capacidades nominais e dimensionalmente idênticas.
- Faixa de utilização: em geral, 15% a 50% da capacidade nominal destinada à indicação do peso liquido, graças à elevada gama de capacidades.
- Sinal de saída: 2 mV/V @ capacidade nominal.
- Erro combinado: 0,02% da capacidade nominal (inclui os efeitos combinados de histerese, não linearidade e repetibilidade).
- Agressividade ambiental: excelente comportamento. Construção em aluminio anodizado.
- Grau de proteção: IP-65 (standard). Opcionalmente, com grau de proteção IP-67.

GERADOR DE PULSOS

- Instalação: contrapeso e polia em contato com a parte inferior (limpa) da correia de carga.
- Tipo: óptico eletrônico. Sem partes móveis sujeitas a desgaste.
- Pulsos/revolução: 900 PPR (2 pulsos/mm). Elevada resolução.
- Agressividade ambiental: excelente comportamento. Construção em Zamak com eixo de aço inoxidável e mancais com rolamentos blindados.
- Grau de proteção: IP-65.

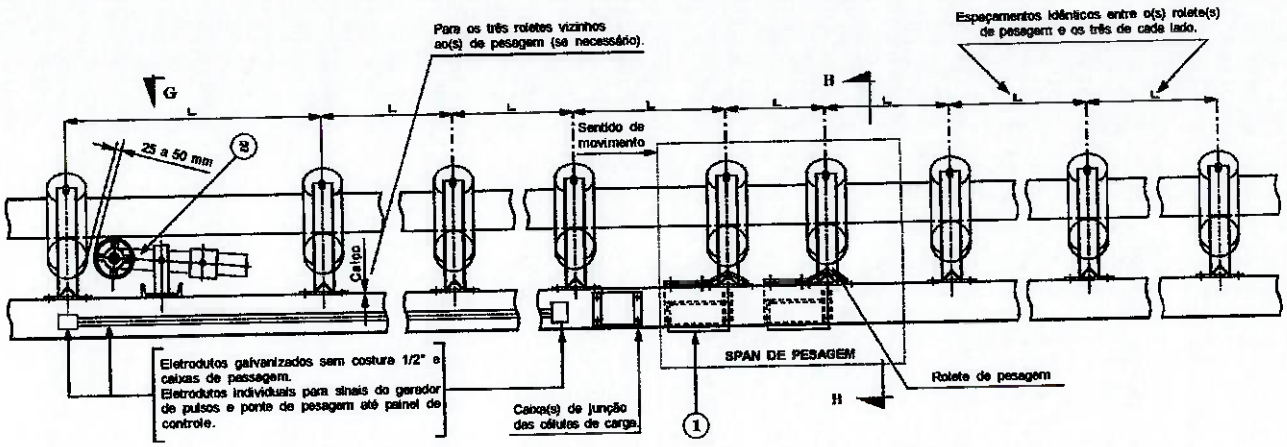
PONTE DE PESAGEM - DIMENSÕES

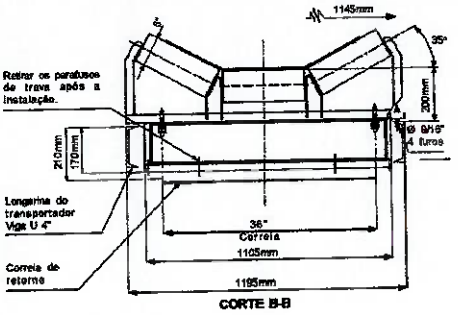
Veja também o desenho ao lado



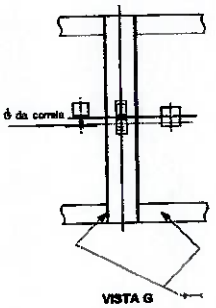
LARGURA DA CORREIA	A (mm)	PESO DE EMBARQUE (kg)
16"	680	135
18"	750	140
20"	800	145
24"	902	155
30"	1055	170
36"	1208	185
42"	1359	200
48"	1512	215
54"	1664	230
60"	1817	245
72"	2121	280

Valores aproximados

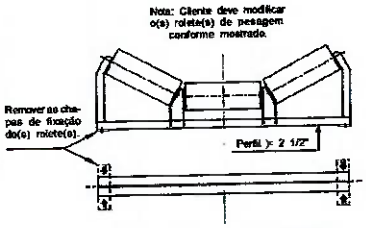




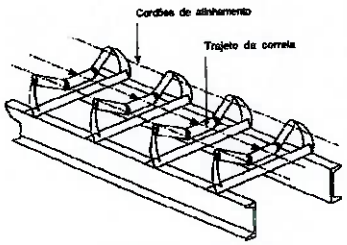
CORTE B-B



VISTA G



INSTRUÇÕES PARA MODIFICAÇÃO DO(S) ROLETE(S) DE PESAGEM



ALINHAMENTO DOS ROLETES

NOTAS

1 O local da instalação da balança deve ser isento de vibração.
2 Instalar a balança a uma distância tal, do ponto de carregamento, que o material já se encontre acomodado na correia.
A balança deve ser instalada no trecho do transportador onde o tensionamento da correia é menor, e distante de mecanismos causadores de variações de tensão.
3 Retirar a cobertura do transportador (e/ou chapa de proteção da correia de retorno) para permitir a instalação da(s) ponte(s) de pesagem. Retirar também para a unidade geradora de pulsos, se necessário.
4 A unidade geradora de pulsos deve ser instalada centralmente sob a correia de carga e na posição indicada no desenho (antes ou após o(s) rolete(s) de pesagem).
5 Não desmontar a balança para a instalação.
6 A distância mínima do topo da longarina do transportador à correia de retorno não deve ser inferior a 185 mm.
7 O cliente deve calçar os três roletes de cada lado do(s) rolete(s) de pesagem, se necessário. O topo do rolo central destes seis roletes e do(s) de pesagem devem estar num mesmo plano. Não devem ser utilizados Offset rollers. Utilize cordões de alinhamento.
8 A perpendicularidade do(s) rolete(s) de pesagem em relação à direção de deslocamento da correia deve estar dentro de ± 1/2°.
9 O espaçamento máximo do perfil de atravessamento da correia não deve exceder a 0,5 mm.
10 Não alterar o comprimento dos cabos das células de carga.
11 Para manutenção da calibração é recomendado que, após aferição inicial, os seguintes itens sejam firmemente soldados:
. O suporte do gerador de pulsos às longarinas do transportador.
. Os três roletes de cada lado do(s) rolete(s) de pesagem às longarinas do transportador.
12 A estrutura do transportador na região da balança, no mínimo 6 metros antes e após a mesma, deve ser contínua e rígida o suficiente para que a deflexão relativa entre o(s) rolete(s) de pesagem e seus adjacentes seja eliminada sob todas as condições de carregamento. Como regra prática, a deflexão entre dois roletes adjacentes quaisquer, dentro da zona crítica, não deve exceder a 0,6 mm, quando sob carga.
13 Toda zona crítica da balança deve estar adequadamente protegida contra vento, chuva, etc.
14 Mesmo sem carga, a correia deve manter pleno contato com o(s) rolete(s) de pesagem e com os três de cada lado do(s) mesmo(s).
15 Os roletes da zona crítica devem possuir excentricidade máxima inferior a 0,2 mm T.I.R. com o(s) de pesagem, ainda, balanceado(s) dentro de 0,11 Newton metro.

DADOS GERAIS

MATERIAL TRANSPORTADO	Pedra britada
PESO ESPECÍFICO	2,0-2,3 ton/m³
TEMPERATURA	Ambiente
ÂNGULO DE REPOUSO	30°-35°
CAPACIDADE MÁXIMA	660 t/h
CAPACIDADE NORMAL	500 t/h
CAPACIDADE MÍNIMA	340 t/h
PORCENTAGEM DE ENCHIMENTO	50%
VELOCIDADE DA CORREIA	1 m/s
COMPRIMENTO DA CORREIA	78 m
PESO LINEAR DA CORREIA	18,1 kg/m
PESO DO ROLETE	27,5 kg
ESPAÇAMENTO "L" ENTRE ROLETES	1.300 mm
ESTICADOR	Gravidade vertical
INCLINAÇÃO DO TRANSPORTADOR	15 10'

2	1	GERADOR DE PULSOS		V2M- 6067800
1	2	PONTE DE PESAGEM		V1M- 6067794
ITEM	QUANT	DESCRIÇÃO		DESENHO N.
EMISSÃO				
DES.	Lucio	15/11/94	TOLEDO DO BRASIL INDÚSTRIA DE BALANÇAS LTDA.	
PROJ.	Alberto	7/6/94		
APROV.	José	8/6/94	DESENHO DE LAYOUT	
LIBER.		/ /	BALANÇA INTEGRADORA MODELO 9270/2	
ESCALA				
QUANT.	—			
LISTA N.	CONJ. GERAL N.	O.S.	A.F. 13209	DESENHO N. V1M-6067802 REV. 0

FORNECIMENTO TOLEDO

9270 BALANÇA INTEGRADORA TOLEDO

PAINEL DE CONTROLE

- Unidade microprocessada com interface alfanumérica de fácil programação e operação. Três displays fornecem todas as indicações ao usuário de como o Sistema Transportador de Correia & Balança 9270 está operando, e facilitam a entrada de parâmetros via teclado.
- Calibração dirigida, com instruções e valores calculados indicados ao operador em displays, em até *6 Regiões de Calibração*.
- Opção de captura automática do *Zero Dinâmico* para cargas inferiores a 2% ou 4% FS.
- Opção de *Não Totalização* para cargas inferiores a 2% ou 4% FS.
- Proteção total das constantes de calibração e totais acumulados na falta de energia elétrica: sistema de back-up com autonomia de 4.000 horas.
- Displays numéricos em LED com 5 dígitos para indicação da vazão instantânea e 6 dígitos para indicação parcial de material totalizado. Display LCD alfanumérico de cristal líquido com 2 linhas de 16 caracteres para visualização do total de material transportado (12 caracteres), velocidade instantânea da correia, mensagens de alarme, menu de programação/seleção e outros.
- Saídas seriais para interligação a microcomputador ou impressora, e ligação em rede multiponto com outros equipamentos Toledo.
- Saídas digitais via relés de estado sólido para indicação de "Alarme" e "Setpoint Atingido".
- Saída analógica em tensão ou corrente correspondentes à vazão instantânea.
- Saída pulsada para totalização remota de material transportado.
- Com impressora acoplada, emissão de relatórios com data, hora, fatores de calibração, totais acumulados, taxas de fluxo, mensagens de alarme e outros.

TOLEDO - ALTA TECNOLOGIA EM PESAGEM

BALANÇA INTEGRADORA 01

CAPACIDADE:	500,00 t/h	CORR DE ZERO:	0,010
SEÇÃO DE PESAGEM:	2,40 m	FATOR ZONA 1:	0,12345678901
COMPR. CORREIA:	100 m	FATOR ZONA 2:	0,12345678901
VELOCIDADE NOMINAL:	160,00 m/min	FATOR ZONA 3:	0,12345678901
TEMPO TOTALIZAÇÃO:	24 h	FATOR ZONA 4:	0,12345678901
ELIMINAR TOTALIZAÇÃO:	S	FATOR ZONA 5:	0,12345678901
AUTOZERO HABILITADO:	S	FATOR ZONA 6:	0,12345678901

MENSAGEM	DATA	HORA	TOTAL	TOTAL GERAL
INÍCIO OPERAÇÃO	20/11/94	13:30	0,00 t	360.240,30 t
CARGA < 20%	20/11/94	13:50	170,25 t	360.410,55 t
# AUTOZERO	20/11/94	13:58	183,70 t	360.424,00 t
OPERAÇÃO NORMAL	20/11/94	14:00	183,70 t	360.424,00 t
CARGA > 100%	20/11/94	14:13	291,40 t	360.531,70 t
SOBRECARGA	20/11/94	14:20	355,65 t	360.595,95 t
OPERAÇÃO NORMAL	20/11/94	14:27	425,80 t	360.666,10 t
# PREDET LIGADO	20/11/94	14:36	500,00 t	360.740,30 t
FINAL OPERAÇÃO	20/11/94	14:36	500,00 t	360.740,30 t

- Senha de acesso, religação automática, chave comutadora liga/desliga, sinaleiro indicativo de "Alarme" e outros.

A TOLEDO SEMPRE ESTÁ ONDE VOCÊ PRECISA

TOLEDO DO BRASIL INDÚSTRIA DE BALANÇAS LTDA.

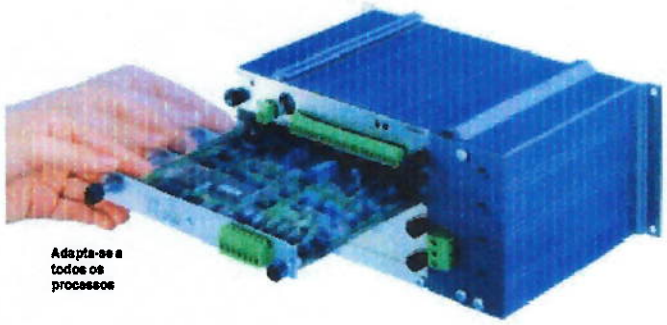
BELÉM, PA ..... TEL. (91) 233-4891 FAX (91) 244-0871
BELO HORIZONTE, MG ..... TEL. (31) 412-4188 FAX (31) 412-4306
CAMPINAS, SP ..... TELEFAX (19) 225-6666
CAMPO GRANDE, MS ..... TEL. (67) 741-1300 FAX (67) 741-1302
CURITIBA, PR ..... TELEFAX (41) 332-1010
FORTALEZA, CE ..... TEL. (85) 283-4050 FAX (85) 283-3183
GOIÂNIA, GO ..... TELEFAX (62) 202-0344
MANAUS, AM ..... TEL. (92) 635-0441 TELEFAX (92) 233-0787
PORTO ALEGRE, RS . TELEFAX (51) 337-2966
RECIFE, PE ..... TEL. (81) 339-4774 FAX (81) 339-6200
RIBEIRÃO PRETO, SP ..... TEL. (16) 626-4252 FAX (16) 626-5595
R.DE JANEIRO, RJ..TELEFAX (21) 590-5216
SALVADOR, BA ..... TELEFAX (71) 384-6618
SANTOS, SP ..... TEL. (13) 222-2365 FAX (13) 222-3854
S. J. DOS CAMPOS, SP ..... TEL. (12) 321-8077 FAX (12) 321-8198
SÃO PAULO, SP ..... TEL. (11) 6160-9117 FAX (11) 6915-7766

RUA DO MANIFESTO, 1183 - TEL. (11) 6160-9000 - CEP 04209-901 - SÃO PAULO - SP - BRASIL
site: www.toledobrasil.com.br

TBR-WIN-02a MAI/97

Arquitetura Aberta
Integra Dados de Pesagem na Automação de Processos

Terminal de Pesagem Poderoso e Fácil de Usar na Automação Industrial



Adapta-se a todos os processos

Integrar balanças em uma automação industrial deveria ser uma tarefa fácil, e não uma aventura.

No desenvolvimento do poderoso terminal industrial de pesagem Jaguar, o objetivo principal era facilitar o projeto, construção, instalação, start-up, operação e manutenção de sistemas que usam balanças. As necessidades, problemas e expectativas foram ouvidas e combinadas com a criatividade da Toledo. Resultado... um excelente instrumento de pesagem para seu sistema.

O terminal de pesagem Jaguar inclui novas e avançadas tecnologias, oferecendo a maior performance já atingida até hoje. E pode também ser combinado num sistema altamente integrado, compartilhando dados e recursos.

Como o mundo da automação está em constante mudança, o terminal de pesagem Jaguar incorpora um sistema de arquitetura aberta, que permite crescimento junto com a aplicação ou mudança de tecnologia. Com o terminal de pesagem Jaguar na integração de sistemas se obtém todos os benefícios desejáveis de um instrumento de pesagem... e com um valor extraordinário.

Aventuras podem ser divertidas, mas não quando se tem problemas a resolver e trabalho a fazer.

Rápida instalação em painel

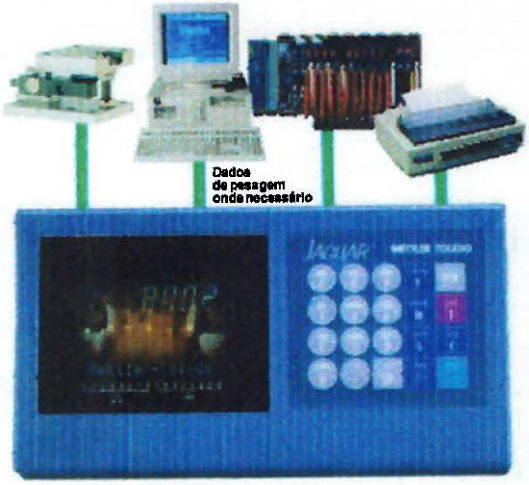
CARACTERÍSTICAS E BENEFÍCIOS

O terminal de pesagem Jaguar se conecta a mais tipos de balanças do que qualquer outro instrumento de pesagem, desde células de carga digitais da Mettler Toledo até balanças de alta precisão. Com células analógicas, proporciona a mais alta performance e a melhor rejeição a ruído disponível no mercado.

TraxDSP™ - Rejeição a Vibração

O TraxDSP é uma tecnologia exclusiva da Mettler Toledo para ambientes instáveis. Combina diversas inovações tanto em hardware como em software. No hardware inclui uma seção analógica extremamente limpa, um conversor analógico/digital de tecnologia exclusiva da Mettler Toledo, e um processador de alta velocidade exclusivo para filtragem digital. No software inclui filtros digitais próprios, super-rápidos e de multiestágio, que podem ser ajustados de forma a proporcionar a melhor combinação de filtragem e velocidade. Com o TraxDSP se consegue pesar em aplicações que antes se considerava impossível, como pesar com um misturador ou um agitador na balança.

TraxDSP para rejeição a vibração em fábricas e processos



Dados de pesagem onde necessário

VANTAGENS FÍSICAS

A construção do Jaguar objetiva proporcionar o máximo de uso no sistema. O painel frontal é construído de alumínio fundido, para evitar deformações devidas a condições ambientais extremas como temperatura elevada e operadores descuidados.
O grau de proteção do painel é NEMA 4.

A caixa é de alumínio extrudado, facilitando a montagem em painel com um simples rasgo retangular. Todos os conectores externos são removíveis, tipo borneira. Até mesmo olhais de fixação são disponíveis para amarrar os cabos no terminal, evitando forçá-los. Todas estas vantagens e características permitem uma instalação limpa e confiável em um sistema novo, ou na reforma de um existente.

O Jaguar pode ser configurado com ou sem painel do operador. Os sistemas que não requerem a indicação de peso o tempo integral, para cada balança, podem usar um chassis cego montado dentro do painel. Isto poupa espaço precioso no frontal do painel. A interligação em rede standard permite ao operador usar qualquer display/teclado de um Jaguar para interagir com as unidades cegas.

REDE

Todo Jaguar vem de fábrica com uma porta de rede standard que permite o compartilhamento de dados e recursos. Um par de fios standard padrão ANSI 878.1 conecta via ARCnet diversos terminais em rede. Esta característica permite que o terminal de pesagem Jaguar partilhe teclados/displays, possibilitando o acesso a todas as balanças na rede, de um terminal que possua teclado/display.

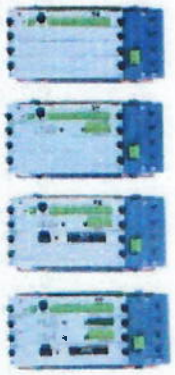
Também se pode receber dados de diversas fontes. A rede de Jaguar permite acesso a todas as portas seriais dentro da rede. Pode-se projetar, por exemplo, um sistema de dosagem de múltiplas balanças e gerar um relatório de dosagem de todas as balanças em um só terminal. Não é necessário software especial ou qualquer equipamento adicional.

Quatro Jaguar com chassis cego

ENTRADAS E SAÍDAS

Todo Jaguar possui entradas e saídas seriais e paralelas, e opcionalmente outras. Existem duas portas seriais completas bidirecionais, disponíveis em RS-232, loop de corrente, RS-422, ou RS-485. As duas portas podem fornecer dados em demanda ou contínuo, formatados livremente de acordo com as necessidades da aplicação.

O fornecimento standard inclui quatro entradas e quatro saídas discretas, para uso conforme as necessidades. As entradas podem ser usadas, por exemplo, para executar funções de teclado, efetuar tara automática, ou iniciar uma saída serial para impressora. E as saídas, para monitorar status, tais como setpoints, ou indicar estabilização da balança. Uma placa multifunção, opcional, acrescenta duas portas seriais adicionais, oito entradas e oito saídas.



1 ou 2 balanças
2 ou 4 portas seriais
8 ou 24 entradas e saídas
Allen Bradley RIO - Configuração de acordo com as necessidades

FLEXIBILIDADE

O terminal de pesagem Jaguar oferece a flexibilidade de adicionar ou remover displays e teclados, proporcionando interfaces homem/máquina, locais ou remotos. O backplane passivo permite adicionar opções de interface para outras balanças. Outros periféricos podem ser utilizados mediante uma variedade de placas opcionais. A arquitetura aberta do Jaguar permite adicionar opções de hardware e de software no campo, de forma que o terminal acompanhe as necessidades do processo.

Uso simples e intuitivo

A C Reference Implementation of the LonTalk® Protocol on the MC68360

Document Revision 1.7

Revision Date: July 15, 1998

Code Version 1.7

Adept Systems Incorporated

www.adeptsystemsinc.com

Boca Raton, FL 33428-4861

+1-561-487-1244 (tel) +1-561-487-8930 (fax)

smithsm@adeptsystemsinc.com adept@adeptsystemsinc.com

TABLE OF CONTENTS

1 OVERVIEW

This document describes the "C" language reference implementation of the LonTalk* protocol on the MC68360. The objectives of the reference implementation are described along with the approach taken in its development. The software architecture is described along with the relevant details associated with implementation on a 68360. The API for the reference implementation is also given.

1.1 Objectives of Reference Implementation

1.1.1 Support Open Protocol Specification— In the past the only available implementation of the LonTalk® protocol was on a Neuron® processor. Recently however, Echelon® Corporation has made the LonTalk protocol open for implementation on any processor. In an effort to facilitate more rapid implementation and porting to other processors, Echelon®* Corporation has contracted with Adept Systems to develop and document a working C language implementation on an MC68360 as the basis for a open protocol specification. LonTalk is under consideration for adoption as a standard protocol by several industrial control standards organizations. An essential requirement for many of these standards bodies is a complete open specification of the protocol. The existing LonTalk Protocol Specification document published by Echelon contains only pseudo-code and is not a complete self-contained specification. One of the goals of reference implementation is to expand and clarify the specification with the addition of working C code. The reference implementation was developed from a functional description only of the LonTalk protocol. No Neuron® C or Neuron assembly source code was provided for reference. Because of this "clean" approach to the reference implementation development, omissions, confusing descriptions, and errors in the original protocol specification became more apparent. As a result a list of clarifying questions and answers has been compiled for future reference.

1.1.2 Clarity and Understandability vs. Performance— Because the reference implementation's main purpose is to support an open protocol specification the emphasis was put on developing clear, correct, and understandable code as opposed to optimally performing code relative to speed and memory. The implementation minimizes dependence on the features of a specific development environment or operating system services or additional external hardware. For example, the reference implementation does not use a real time operating system (RTOS) but has built in its own scheduling and timing facilities. In every case where possible ANSI C language code was used instead of assembly. In fact, there is only one line of assembly code in the reference implementation. The behavior of the Neuron® processor is the "gold" standard for correct behavior of the implementation.

1.1.3 Readability— A set of coding conventions has been adopted that emphasize readability and documentation. The coding conventions help to maintain consistency throughout the code. The conventions used were based on a combination of recommendations from well known C style manuals and the development teams preferences.

1.1.4 Neuron® Memory Map Emulation— A virtual memory map of the Neuron Chip's 64k memory space is maintained in a global structure. This approach makes it easier for the reference

---

*. Echelon, LON, LONWORKS, LonBuilder, NodeBuilder, LonManager, LonTalk, LonUsers, Neuron, 3120, 3150, the Echelon logo, and the LonUsers logo are trademarks of Echelon Corporation registered in the United States and other countries. LonLink, LonResponse, LonSupport, LonMaker, and LonPoint are trademarks of Echelon Corporation.

implementation to replicate Neuron® chip facilities for reading and writing memory to set configuration parameters in response to network management messages.

1.1.5 Data Flow Architecture— The software design developed by Adept is driven by the objectives stated above. A data flow style architecture was selected where the layers execute in round robin fashion and pass information from layer to layer with global memory queues. The provides a clearer separation of functionality in the layers than would a function call stack thereby enhancing understandability.The data flow approach also simplifies the implementation by avoiding function blocking for timers to expire and/or complex function return handling. This approach also makes it easier to port the implementation to other processors and/or a multi-process RTOS where each layer could be its own process and communicates through either shared memory or message queues.

1.1.6 68360 Microcontroller— The microprocessor hardware selected by Echelon for the reference implementation is the Motorola MC68360 quad integrated communications controller. The 68360 has a 32 bit processor core and a multifunction communications processor module. The communications processor includes many of the functions needed to implement the media access control layer of the LonTalk protocol. Because of the 68360's built in support for several other protocols it has potential as a generic platform for building gateways between LonTalk and other protocols such as TCP/IP-Ethernet. It was hoped that no external hardware would be required to implement the LonTalk protocol on the 68360. As of this writing, however, several hardware design features of the 68360 are incompatible with the direct mode of the LonTalk protocol. This document describes the finished implementation of special purpose mode only.

1.1.7 Future Protocol Ports— One other requirement is that Adept Systems be positioned in the future to do ports of the LonTalk Protocol to other platforms or provide consulting services to others doing ports.

1.2 Development Team

Dr. Samuel Smith and Dr. Stanley Dunn founded Adept Systems in 1994 to commercialize intelligent, distributed control system technology and expertise developed at Florida Atlantic University (FAU). The FAU Advanced Marine Systems Group, under the direction of Dr. Smith and Dr. Dunn, has developed this technology and expertise over the past six years as the result of a multi-million dollar research program to develop autonomous underwater vehicles (AUV), sensor systems, and shipboard automation. A key factor in this successful research has been the creation of a modular, low-cost reconfigurable AUV architecture that uses the LONWORKS® technology.

For the purpose of this proposed work, Adept has assembled a very capable development team. The 3 primary members of this team are Samuel Smith, K. Ganesan, and Bryan Jacobson. In addition to the 3 principles members the development team will be supplemented by staff and graduate students at FAU who will be hired on a contract basis as needed.

Dr. Smith is the project leader. His primary technical participation is focused on the physical and link layers with emphasis on the processor and electronic hardware specific implementation issues. He has a Ph.D. in electrical and computer engineering and is experienced in embedded controllers, and has over 4 years experience with LONWORKS® development and 10+ years experience in C programming. Dr. Ganesan's primary technical participation will focus on the network, transport, and session layers in addition to the overall architecture. He has a Ph.D. in computer science and is experienced in networking, real time operating systems, and database design with

12+ years of experience in C programming. Mr. Jacobson's primary technical participation will focus on the presentation and application layers and in code integration, testing, and validation. He has an M.S. in Computer Science and is experienced in portability, portability testing and validation, and compiler15+ design with 15+ years experience in C language programming.

1.3 Development Tools

The major components to the development environment include an Arnewsh SBC360-1M development board, Software Development System's (SDS) CrossCode 68K C Compiler Suite and SingleStep C BDM Debugger.

The SBC360 has a 25 MHz MC68EN360 processor with 1 Mbyte of DRAM, on-board ROM monitor/debugger, Ethernet port, serial cable, BDM port, and documentation. The board requires an external 5 Volt power supply. The SDS development environment runs in Windows95/NT and includes a software simulator of the 68360. The SBC360 sells for $975. The SDS compiler does not include a make facility. We use the Gnu Software Foundation make utility. It can be obtained by ftp from ftp.prep.ai.mit.edu/make-3.75.tar.gz. A C compiler such as Visual C is needed to build the make utility. A pre-compiled version is provided with the reference implementation. The SDS compiler includes a limited text editor but will work with a more full featured editor such as EMACS. The CrossCode C compiler is $2000 and the SingleStep BDM debugger is $2500. The debugger comes with a cable to connect the BDM port to a PC parallel port. Contact information is listed below.

Arnewsh Inc.
P.O. Box 270352
Fort Collins, CO 80527-0352
Tel: 970-223-1616 Fax: 970-223-9573

Software Development Systems
815 Commerce, Suite 250
Oak Brook, Illinois 60521
Tel: 800-448-7733 or 630-368-0400 Fax: 630-990-4641

1.4 Development History

Because the reference implementation was developed from a functional description of the protocol specification, the first few weeks of the project involved extensive review and clarification of the protocol specification document through a combination of phone, email, and in person exchanges between Echelon and Adept [Echelon 95]. A history of the Q&A exchanges has been compiled. The initial software architecture was then designed. The coding tasks were divided into three groups, the MAC layer, the middle layers 2 – 6, and layer 7 including the API. Coding proceeded in parallel on these 3 groups. Modifications and revisions to the architecture were made as appropriate as understanding and the implementation matured. Before the MAC layer was finished it was desirable to test the completed parts of the middle layers. To do this the protocol stack was modified to allow multiple protocol stacks to run on a single processor with a software virtual channel connecting the stacks. This greatly sped the testing of the middle and upper layers of the reference implementation.

Clarifications on 68360 functionality were provided by one of the Motorola field sales representatives. At first study it appeared that there might be a few problems in implementing the direct mode on the 68360 without external hardware. As a result the initial development focused on

implementing the special purpose mode while waiting for help and clarification from Motorola. The goal was to implement a skeleton stack that supported at least one physical layer interface, one protocol service type and one message type. The technical challenge for the special purpose mode was to implement continuous transfers of frames between the 68360 and the transceiver. A combination of functionality in the transceiver and 68360 make is impossible to implement continuous transfers. Instead a non-continuous mode was implemented with the addition of some limited external hardware. Details are given below in Sec. 3. This allowed the upper layers to communicate with Neuron Chips and facilitated further development and testing of the upper layers. Once the basic special purpose mode was working, development resumed on the direct mode. Several discrepancies between the actual and documented function of the 68360 in transparent mode became problematic with regards to implementing direct mode. Development resumed on the channel access algorithm for special purpose mode which was subsequently completed.

1.5 Coding Conventions

The Coding conventions and examples are given below.

```
File Naming
    File names are all in lower case.
    The name reflects the purpose of its use.
    For example, network.c or session.c.
    To facilitate use with DOS systems, we will restrict our filename
    length to 8 characters.

.c and .h files
    Every .c file will have a corresponding .h file that represents
    the public interface except main.c that contains main.

    Every .h file has the following information:

    1) constant definitions (#define)
    2) macro definitions
    3) type definitions
    4) global variable declarations (i.e extern declarations)
    5) functions prototypes

    Only prototypes for functions that are visible outside of the .c file
    should be given the prototypes in the .h file.

    All functions which are used locally inside the .c file should be
    declared as static so that there is no conflict with same names
    from other files.

    Global variables used only in the .c file should be declared as static.
    Only those global variables that are declared in the .c file and visible
    outside should be given the extern declaration in the .h file.

    Each function prototype should be preceded by a comment that gives
    information about the proper usage of the function. The function
    prototype should be the same as the function heading in the .c file.
```

Every .h file should use #ifdef to avoid multiple inclusions.

```
#ifndef _NETWORK_BUFFERMGT
#define _NETWORK_BUFFERMGT
<body of file>
#endif
```

Also, when header files are included in .c files, use < > instead of double quotes. Use the compiler option -I to tell the compiler where to find the header files. This gives flexibility in relocating header files, if necessary.

File Creation

Restrict all lines in the file to no more than 70 characters, whenever possible.
Use tabs as you like. But for portability, replace tabs with spaces when saving files.

Every file should have a header and footer. The header should identify the following:

```
File Name:
References:     Reference to protocol spec sections or other papers
                that contributed to the code development.
Purpose:        The purpose of this file.
Note:           Any thing that does not fit anywhere above.
To Do:          Things to be done. Delete this as soon as it is done.
```

The footer should identify the end of the file.

```
/**********************sesenc.c**************/
```

Coding

Function Headings:

Every function should be preceded by decorated comment that gives the following information:

```
/*****************************************************************
 Function:
 Returns:
 Reference: Protocol Spec Reference
 Purpose:
 Comments:  Any thing else you want to say.
*****************************************************************/
```

Constants:

All upper case letters. Use _ to separate words

```
#define MAX_MSG_BUFFER_SIZE 500
#define MAX_MSG_COUNT        10
```

Parameterized Macros:

All upper case letters. Use _ to separate words. Use parenthesis for protecting arguments.

```
#define MAX(a,b) (((a) > (b))?(a):(b))
```

Types:

Capitalize each word in the type name.

```
        MsgOut
        BroadcastAddress

    Variables:
     Start with lower case and every word should be capitalized.
     No underscore is used.
        maxSoFar
        repeatCount
        priority
     For Pointer variables, use the suffix Ptr, if convenient.
     For Global  variables, use the suffic Gbl, if convenient.
        addrTblGbl
        msgTblPtr

    Formal Parameters:
     Use the following suffixes for all formal parameters of functions.
     This requirement is not absolutely necessary, but makes the code
     more readable. Follow this whenever possible.
        In      For Input Parameters passed by value
        Inp     For Input Parameters passed by reference
                (for efficiency. Especially structures)
        Out     For Output Parameters
        InOut   For Input/Output Parameters

    Functions:
     Capitalize every word including the first word.
        SendMessage
        ReceiveMessage
        NetVarUpdate
```

Indentation style

```
    We shall use 3 spaces for indentation of sub-statements for all statements.
    Use space after keywords such as if, for, while, switch.
    Use space before and after operands such as +, -, *, etc.
```

Functions

```
    The indentation style for functions looks as follows:

    <return-type> <fn-name>( <parameters> )
    {
       <declarations>

       <body>
    }

    <declarations>
        If more than one variable is declared in one declaration, put the
        variables on separate lines to facilitate comments for each variable.
        Indent all variables as well as comments.
        For example,

     double  miles,   /* Distance in miles */
             kms,     /* Distance in kms   */
             feet;    /* Distance in feet  */
```

```
    <paramters>
        Use suffixes Inp, In, Out, InOut as described earlier.
```

if

```
    if ( <expr> )
    {
       <then-part>
    }
    else if ( <expr> )
    {
       <else-part>
    }
    else
    {
       <else-part>
    }
```

Note: Even if there is only one statement in then or else parts, we shall use {} so that future changes are easier. The same goes for loops too.

while

```
    while ( <expr> )
    {
       <declarations>

       <body>
    }
```

do while

```
    do
    {
       <declarations>

       <body>
    }
    while ( <expr> );
```

for

```
    for ( <init> ; <condition> ; <update> )
    {
       <body>
    }
```

switch

```
    switch ( <expr> )
    {
       case <const>:
          <statments>
          /* Fall Through */
       case <const>:
```

```
        <statements>
        break;
      :
      :
    default:
        <statements>
}
```

Even if there is nothing to do in default case, we shall always have the default case. If there is nothing, just have a comment that says so. /* do nothing */.

Other Files

In addition to the .c and .h file pairs, we may have additional .h files that do not necessarily correspond to any .c file. The purpose of these files is to provide system wide definitions of types and constants.

End of Code Conventions

2 REFERENCE IMPLEMENTATION SOFTWARE ARCHITECTURE

This section explains the architecture and data structures used for implementing the reference implementation. The reader is assumed to have some reasonable understanding of the services offered by various layers of the protocol stack. Working knowledge of Neuron® C programs is also helpful.

The architecture for the reference implementation uses a data-flow model instead of a functional model due the nature of the interaction among the various layers. One of the reasons for this choice is its simplicity. The data-flow model helps avoid unnecessary complexities involved with blocking when function calls are used. The data flow model is also easier to partition and schedule in a real time limited resource implementation. Each layer has three major functions: *Reset, Send, Receive*. *Reset* is used on a node reset so that the layer can initialize and allocate its data structures. *Send* is used to process any outgoing packet(s) waiting for that layer. *Receive* is used to process any incoming packet(s) waiting for that layer. Some layers also have an additional function *Init* to perform initialization that is done only once during power-up.

The main round robin scheduler simply cycles through all the layers and the application program. In each cycle, the scheduler calls the Send function of each of the layers then calls the Receive function of each of the layers starting. The exact sequence is diagrammed in Fig. 2.1 . In each cycle, it also calls the Application Program. It is up to the Application Program to use the time slice in whatever way it wants to use it. For example, it can call one or more critical sections during that time slice. It is important that the application program does not take up too much time or else the layers will not have sufficient time to process the messages. The Send and Receive functions of each layer should process minimal amount of work and return back to the scheduler. The main scheduler calls the function NodeReset at start and whenever the node is reset. The function NodeReset in turn calls the Reset functions of each of the layers. In addition, the main scheduler also calls the function PHYIO that is responsible for checking button press events and LED control. The timing critical parts of the Link layer including the MAC sub-layer such the packet framing, encoding, decoding, and channel access algorithms are interrupt driven.

The information flow between the layers through the respective buffers is shown in Fig. 2.2. The reference implementation has been designed with minimal or no dependence on operating system details or machine architecture. In fact, the implementation does not use any operating system. The only code that depends on 68360 are the Interrupt Service Routines. Emphasis has been placed on readability more than on efficiency, even though every effort has been made to make code run efficiently. The code can be easily enhanced and ported to other systems provided the physical layer can be rewritten for the new architecture. Another important feature of the reference implementation is that it supports multiple stacks for possible future extension. Currently the multiple stacks run only in simulation mode in which there is no physical layer is involved. A stub function transfers packets between all the stacks. In fact, the simulation mode was used early in the development of the reference implementation to test the higher level code before the physical layer was available. It is not difficult to modify the code to handle one physical layer, but with several stacks running simultaneously on the 68360.

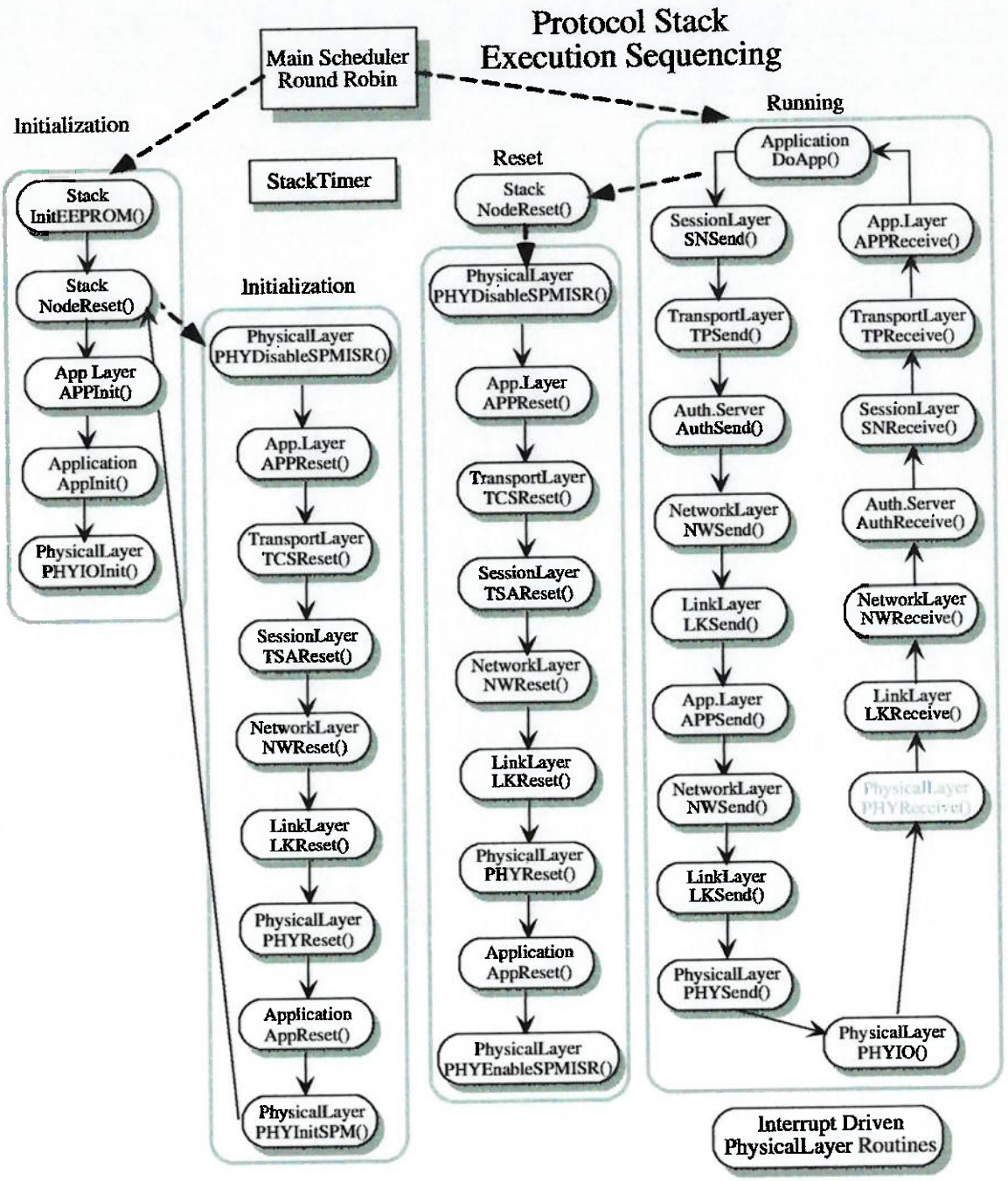


Figure 2.1 Main Scheduler Software Execution Sequencing

2.1 Queues

Queues are used as the primary data structure to store packets. There are generally three kinds of queues: *Input, Output, and OutputPriority. Input* queues are used to store incoming packets. *Output* queues are used to store non-priority packets. Finally, *OutputPriority* queues are used to store

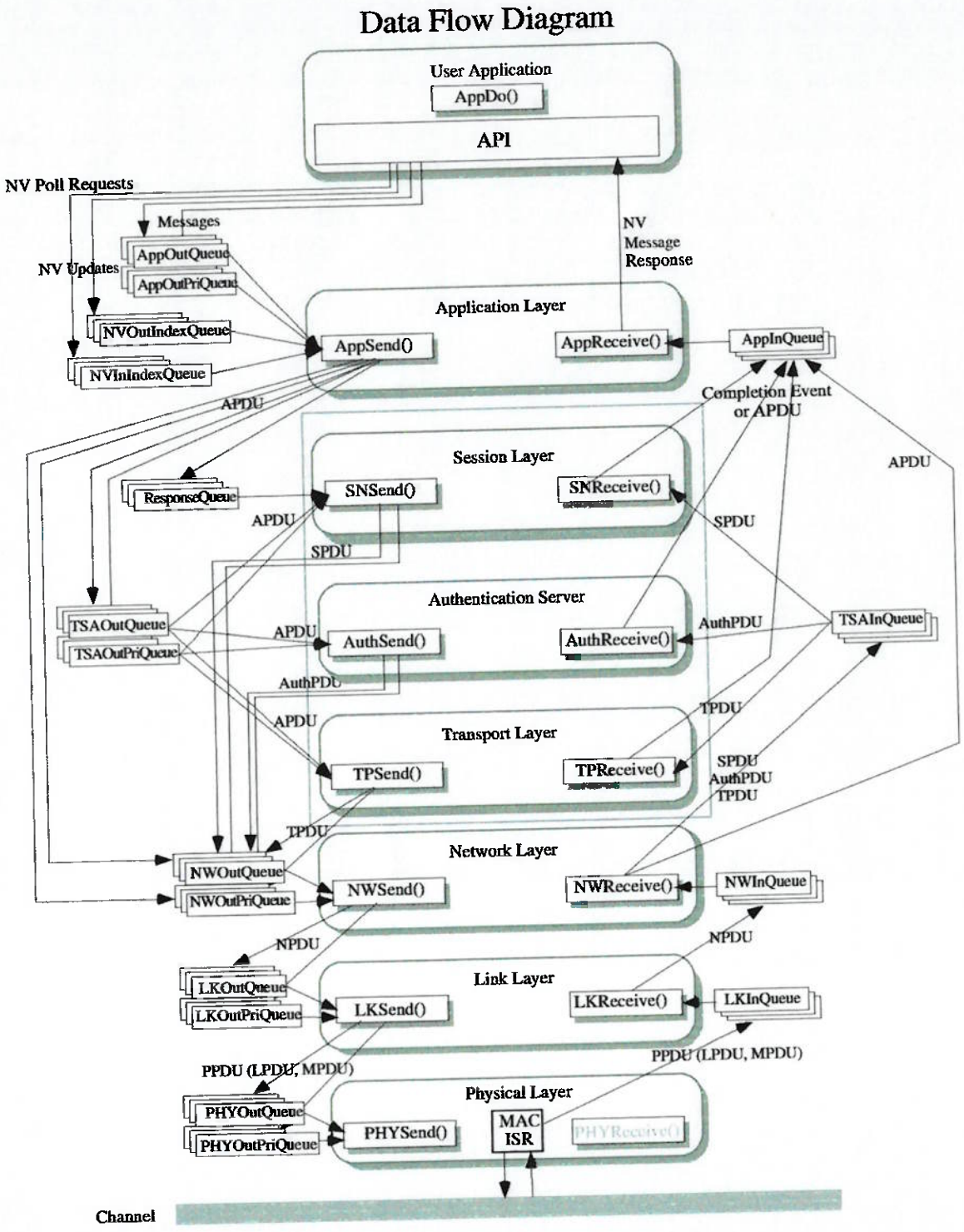


Figure 2.2 Data Flow Diagram

priority packets. In this section we explain the queues used by each of the layers and how they are used.

2.1.1 Application Layer— The application layer uses *Output* and *OutputPriority* queues to store explicit messages sent by the application program. If these queues are full, then the application can not queue up any more messages. Each item in the queue has two parts. The first part of the item is a fixed structure called APPSendParam. This structure has all the information regarding what to do with the packet such as whether the packet needs authentication, what service is used etc. These are the same information available in the MsgOut (or msg_out) structure except the data portion and the code. The second part of the queue item is the actual APDU which consists of the code and the data. If the data is too large to fit, the message is discarded and the application is notified with a completion event (i.e app layer calls the function MsgCompletes). Based on the priority of the message, the packet is queued up in the appropriate queue. If the queue is full, the message is discarded and the application is notified with MsgCompletes event.

The application layer uses an Input Queue for receiving incoming packets from the network as well as to receive transaction completion indications from the transport, session, and network layers. If the packet received from this queue is an indication event, then it might be for a transaction generated by the application layer itself (e.g. Network Variable Updates or Network Variable Polls) or for a transaction generated by the application program. Transactions generated by the application layer are given negative tags and transactions generated by the application program are given non-negative tags. The indication packet is appropriately processed either to give MsgCompletes event or NVUpdateCompletes event to the application. Some indications might be discarded such as the ones for Service Pin messages.

The application layer and the session layer share a dedicated queue for responses. Responses from the application program and the application itself are placed in this queue. This queue is used only for responses. The session layer checks this queue for any outstanding responses every time the SNSend is called. There is a good reason for having a dedicated queue for the response queue. The alternative to not using the queue is to place the response along with messages in the application layer's output queue and then transfer to the session layer's queue. This alternative will fail due to the following reason. Consider a scenario in which a node is busy sending a request message and thus the session layer is busy. Until the transaction is completed, the corresponding message from the session layer's output is not removed. If the response is behind this request message (or even worse well back in the queue) the response will not get delivered in time. When the application wants to send a response, it is directly placed in this queue by the application layer.

In addition to the above mentioned queues, the application layer uses a pair of queues to handle network variable updates and polls. The nvOutIndexQ is used to store the indices of network variables (priority or non-priority) that are scheduled to be sent out. Each item in the queue has the index of the network variable followed optionally by the value of the variable. If a network variable is declared as a synchronous variable, then the value field follows the index. Otherwise, only the index is stored. Every time APPSend function is called by the scheduler, it checks for network variable updates in this queue and sends appropriate nv messages. The nvInIndexQ is used to store the indices of network variables (priority or non-priority) polled by the application program. Once again, the APPSend functions checks this queue and sends appropriate nv request messages. These two pairs of queues are small compared to other queues. Due to the way the algorithm for handling network variables and aliases works, the same queue is used for both priority as well as non-priority variables (This includes any outgoing network variable updates or aliases). This is

equivalent to the way the Neuron chip implementation uses its application buffers. Thus, an outgoing priority message or netowork variable update could get sent down from the application layer after a outgoing non-priority message or network variable update. Using distinct priority and non-priority queues for the outgoing upper layers of the protocol stack would make the book keeping required to process completion events for a network variable and its aliases that do not share the same priority characteristic problematic. The effort needed to implement this is non-trivial and was not attempted on this implementation. At the Physical layer however there are two outgoing queues one priority and one non-priority. Each time the channel access algorithm runs it checks the priority queue first for pending packets. Priority messages or NV updates that arrive at the physical layer before a previous non-priority message or NV update has successfully completed a channel access attempt will get to access the network first.

Every time the scheduler calls the function APPSend, the application layer will try to send a priority message, a network variable update, a network variable poll, and a non-priority message. APPReceive function will receive only one message at a time. If the incoming message cannot be processed due to unavailability of resources, it might stay there until it can be processed.

2.1.2 Transport, Session, (and Authentication) Layers— The reference implementation does not support the tx_by_addr flag in the read only data structure which facilitates a node to send several outgoing transactions as long as they are to different destinations. Thus, there can be at most only one priority and one non-priority transaction that can be active at a given time. To avoid excessive usage of queues, the reference implementation uses a single set of *Output*, *OutputPriority*, and *Input* queues for the transport, session, and indeed the authentication layers. The authentication component is used by both session and transport layers. It is not really a separate layer, but for the sake of understanding of the queue structure, we can consider it to be a layer. The *Output* and *OutputPriority* queues are used to store outgoing APDUs meant for transport or session layer.

The transport layer uses the *Output*, and *OutputPriority* queues to process Acknowledged or Unacknowledged Repeated messages from the application layer. If the message at the head of the queue is not one of these services, the transport layer ignores that queue and does nothing. The priority queue is looked at before the non-priority queue. The item in the queue is not removed until the transaction is complete. The *Input* queue is used for receiving incoming TPDUs. The transport layer will process the incoming TPDU to take appropriate action. Once again, it is possible that the incoming message cannot be processed due to unavailability of resources and hence it might stay there until it can be processed.

The session layer is similar to the transport layer in the way the *Input*, *Output* and *OutputPriority* queues are used. In addition, the session layer examines the *Response* queue for any outgoing responses. The response is matched with the corresponding request in the receive record (discussed later) pool to determine the priority and the destination. If the corresponding network queue does not have space for the response, the response is left undisturbed in the *Response* queue. Currently, the reference implementation does not search for other responses in the same queue for possible transmission (e.g. it does not use the same priority). By nature of the properties of a queue, this operation will violate the way a queue should be used. A different data structure may be more appropriate for this type of operation, but the reference implementation chose to stick with only queues for simplicity.

The authentication layer does not use *Output* or *OutputPriority* Queues. However, the *Input* queue is used to process incoming AuthPDUs (Challenges and Replies). The authentication layer is also

responsible for searching through the receive records pool (discussed later) for messages that require initiation of challenges (that were not sent earlier due to overflow of network queue space). Authentication layer serves both session and transport layers. When a challenge is received, the authentication layer determines the priority of the challenge message and appropriately check the transmit record for a match. If a match is found, it sends the reply. When a reply is received, the authentication layer searches through the receive records pool for a message for which the challenge was issued. If there is no such record, then the reply is discarded. Otherwise, the reply is matched and the authentication flag is set based on whether the match succeeded or failed. As soon as the authentication is completed, the authentication layer will attempt to deliver the packet to the application layer using the same function that is used the transport or session layers. This is done as a courtesy and to avoid the delay in the delivery of the packet until the next cycle. If a packet is flagged as a packet to be authenticated and the authentication process fails (either Reply does not arrive in time or it does not match), the packet is discarded.

2.1.3 Network Layer— The network layer uses *Input*, *Output*, and *OutputPriority* queues. The Output and OutputPriority queues are used to send outgoing APDUs, or TPDUs, or SPDUS, or AuthPDUs. The fixed portion of the queue item indicates whether to use alternate path, the backlog value, destination address etc. The *Input* queue is used to receive incoming NPDUs. The network layer will process the NPDU, strip the header portion, and deliver the enclosed PDU to the appropriate layer by placing it in the *Input* queue of that layer. If the outgoing packet is an APDU, then the network layer will also give success indication to the application layer. The reference implementation requires the output queue size to be at least two or else the program will not run.

2.1.4 Link Layer— The Link Layer uses *Output* and *OutputPriority* queues to process outgoing LPDUs. These LPDUs are generated and placed in the queues by the network layer. As usual, each item has two parts, the first part giving the parameters for handling the LPDU and the second part containing the LPDU itself. The *Input* queue for the link layer is used for receiving incoming LPDUs and is different from all the previous queues in the way it is handled. The *Input* queue is filled by the Interrupt Service Routing(ISR) that handles the physical medium for receiving packets from the network. Since, semaphores are not used for mutual exclusion, there is a potential problem in which the queue is updated (actually updating of the queueSize is the only problem) by both ISR and the link layer. To avoid this problem, we use a queue in which the first part has a flag and size of the LPDU and the second part is the actual LPDU. Link layer maintains a head pointer into this queue and the ISR maintains a tail pointer into this queue. Whenever a new LPDU is received, the ISR checks it the queue item to be used is free by testing the flag. If it is free, it places the LPDU and sets the flag to indicate that the item contains a valid LPDU. Similarly, the link layer checks the flag of the head of the queue to see if it contains valid item. If so, it removes it and resets the flag. Due to corruption of packets when 68360 is asked to compute the CRC, the link layer also computes the CRC for outgoing packets. For incoming packets, the CRC computation is done by the mac layer as bytes are received one by one in the special purpose mode. The reason for is due to the fact that mac layer needs to know whether the packet received has valid CRC or not and the backlog value in the packet to implement the channel access algorithm correctly. Priority slots are present in the channel only after receiving a packet with valid CRC.

2.1.5 Physical Layer— The physical layer is a front end for the Interrupt Service Routine. The physical layer uses only *Output* and *OutputPriority* Queues. There is no need for Input queue as packets are directly retrieved from the network. The *Output* queues are flag based just like the

Input queue for link layer. The Send function of the physical layer checks the data structures for the ISR to see if it is ready for transmitting a new packet. If it is and a packet is available for transmission, it copies the packet from the queue to the data structure for the ISR so that it is sent out. The Receive function of the physical layer currently does nothing as the ISR itself copies the received packet into the link layers's *Input* queue directly.

2.2 Software Timers

Software Timers are implemented with the help of a single hardware timer. For each software timer, we keep track of its current value, the last update tim, and whether the timer expired or not. When a function wants to check if a timer has expired, it simply calls MsTimerExpired. A timer is considered expired only the first time the current value is found to have a value of 0. SetMsTimer function can be used to initialize a timer to some initial value. There is also a function called UpdateMsTimer to update a timer. Any number of software timers are supported. Reference Implementation does not support repeating timers. They can be easily handled by the application program by simply calling SetMsTimer again whenever MsTimerExpired returns TRUE.

2.3 Pragmas

Neuron® C has pragmas to support customization of various parameters affecting the protocol stack. The reference implementation does not have a Neuron® C compiler. Regular C compiler is used to compile the application along with the code for the protocol stack. Hence the file `custom.h` is used to define users configurable constants the node before reset. This file and custom.c contains most of the information that is normally handled through pragma statements in a Neuron® C program.

2.4 Memory Allocation

The allocation of buffers during node reset is done by calling the *Reset* function in each of the layers. A global array that is large enough is set aside to allocate buffers. A variable is initialized to the base address of this array. Each layer takes what it needs from this array and update this variable so that the next layer can allocate buffer using this new address. This mechanism is very simple and helps avoid the use of malloc. The constant MALLOC_SIZE in custom.h determines the size of this array. If it is too small, the layers may not be able to get the storage they need and this situation will result in main returning to start.s (an assembly file giving entry to main) which loops on a single line. DONE BRA DONE; loop if main ever returns.

2.5 Data Structures

The various data structures, including the queues, used for the reference implementation were designed based primarily on the data structures discussed in the LONWORKS® Technology Device Data book. The memory space is divided into two major sections: Stack Data, Neuron® chip Memory Map.

The Neuron® chip Map is used to mirror the memory layout of a Neuron® chip. EEPROM is part of the Neuron® Map and starts at address 0xF000 as in existing Neuron Chips. The data structures other than EEPROM contained in the Neuron Memory Map include the stat structure, SNVT structures, Proxy Data, errorLog, resetCause, and network variable (config, alias, and fixed) tables. The rest of Neuron Memory map is unused. EEPROM contains read only data structure, configuration parameters, domain table, and address table. The reason for not placing network variable tables in EEPROM is to support large number of network variables. The network man-

agement read-memory(or write-memory) command does the necessary mapping necessary to fetch(or write) the data. Reading absolute location 0 is trapped and data value of 11 (base version number) is replaced.

The Stack Data is used to represent the storage for all the queue data structures, hardware timer, transmit records, receive records, tables used by the transaction control sub-layer for assigning transaction ids, api variables such as msgIn, msgOut, respIn, respOut, miscellaneous book keeping variables, and finally, variables representing the status of i/o buttons and LEDs.

To facilitate multiple stacks, these data structures are maintained one for each stack. Three global pointers gp, eep, and nmp are used to point to the appropriate data structures for the individual stack. These variables are set by the scheduler before starting the cycle for an individual stack. The scheduler cycles through the functions for every stack.

The files custom.h and custom.c are used to initialize various data structures of the protocol stack. The scheduler calls the function InitEEPROM to initialize various data structures inside EEPROM based on values specified in these files. This is done only once during the boot process. If the data structures are modified using network management messages, these new values will persist unless the 68360 itself is reset. If the system has access to an external hard disk, the software can be easily modified to save the configuration and binding information in a file on exit and load them after initialization but before the scheduler starts.

The readOnlyDataStruct in EEPROM is 41 bytes long and includes the readOnlyData2 structure described in the Technology Device Data Book. The configData and domainTable are as described in the data book. The number of address table entries is determined by a constant defined in cutom.h. One limitation of the Neuron chip is that the number of address table entries has a maximum of 15. The reference implementation allows any number of address table entries, though it reports a maximum of 15 entries to the management tools through read memory commands relative to readOnlyDataStruct. The use of address tables are discussed in a later section. The network variable config table, alias table, and fixed tables are as described in the data book.

Each stack has one priority and one non-priority transmit record. The number of receive records is determined by a constant defined in custom.h and there is no restriction on the size of receive record table. The transaction control sub-layer uses a table to keep track of transaction ids used for various destinations to ensure that the same id is not used for the next transaction to that destination. The size of this table is determined by the constant TID_TABLE_SIZE defined in the file node.h.

2.6 Features

The reference implementation, not restricted by the limitations of a Neuron chip, has several enhanced features that are not supported in the Neuron chip. This section describes all the features supported by the reference implementation including the enhanced features.

2.6.1 Addressing Modes— Reference Implementation supports Unique Id, Subnet Node, Broadcast, and Multicast addressing modes as is done in the neuron chip.

2.6.2 Broadcast Request— Normally a broadcast request transaction will succeed as soon as the first response is delivered. a new addressing mode called BROADCAST_GROUP is used to support delivery of multiple responses to the application. In this mode, the application can specify how many responses are required. The session layer will keep the transaction until the required

number of responses are delivered or transmission time expires. In any case, the transaction itself will succeed if at least one response is received.

2.6.3 Group Size Compatibility Issue — In Neuron C application programs, the group size for multicast messages where the node is not a member of the group should be set to actual group size + 1 for it to work. In Reference Implementation, this can be set to actual group size unless compatibility is needed in which case it can also be set group size + 1. This is basically a cleaner approach as setting the group size to true group size + 1 is too artificial and is done so that transport or session layer will work properly. The constant GROUP_SIZE_COMPATIBILITY controls this behavior.

2.6.4 Duplicate Detection— Reference Implementation uses a enhanced version of duplicate detection algorithm. For assigning transaction ids, the transaction control sub-layer uses a table in which we remember the last TID for each unique destination address. When a new transaction id is requested for a destination, this table is searched for that destination. If a match is found, we make sure that we don't assign the same id used for that destination. If the destination is not found, we make a new entry in the table. We have an entry in the table for each subnet/node, group, broadcast (subnet or domainwide) and unique id. When a table entry is assigned, we remember the time stamp too. If the table does not have space for a new destination address, we get rid of one which has remained in the table for more than 24 (modifiable in tcs.c) seconds. If no such entry, then we fail to allocate the new transaction id. The table size is configurable. This new algorithm enables client nodes from falsely detecting duplicate transactions from this node. The table is maintained after a software reset to enable the node to remember the transaction ids. This is needed as the time taken to reset could be shorter than maximum receive timer value for all the destinations. For power-up or external reset, we delay transport and session layers by a default value of 2 seconds (configurable in custom.h) to enable the destinations nodes to get rid of all pending receive records from this node.

Another enhancement to the duplicate detection that helps this node is in the way the receive record is handled. Since memory is not a limitation for 68360, we store the entire APDU in the receive records for better duplicate detection. When a new message is received and a matching receive record is searched, we also use the APDU in the matching process to perform better duplicate detection. When authentication is involved, this becomes more important. Neuron chips use a check sum computed from the APDU to make sure that responses for authenticated duplicate requests are sent only when the check sum matches. Matching the whole APDU is certainly better than just using the checksum.

2.6.5 Services— LonTalk Protocol Provides four basic types of message services: *Unacknowledge, Unacknowledged Repeated, Acknowledged, and Request/Response*. The reference implementation provides all these message services with a a few minor enhancements.

2.6.6 Null Response— The reference implementation supports a new response mechanism known as *null-response*. The purpose of the null-response is for the application to indicate to the session layer that it does not want to respond to a request it received earlier. One of the reason an application may not want to respond is that the request needs authentication and the authentication process failed. Null-responses are not sent over the network. A null-response is a cleaner and better solution than not responding at all. The application looks cleaner and friendly! There is however no harm if the application does not respond as the session layer will get rid of the receive record when the timer expires anyway.

2.6.7 Context Dependent Response— In several applications it may be important to make sure that the responses are matched with the correct request to ensure that the response goes to the right destination. The Reference Implementation supports this by assigning a unique request id to each request so that it can be used by the app pgm when sending responses for proper match. Thus there is no need for locking up receive records corresponding to requests without a response yet when the timer expires. As soon as receive timer expires, the corresponding record is released as later responses can be identified as stale with the request id.

2.6.8 Less Traffic to Application Program— In the Neuron Chip implementation, duplicate requests for idempotent responses (> 1 byte) are sent to the application program to respond again. In the Reference Implementation, all responses are saved by the session layer so that duplicate requests are directly responded to by the session layer. If this is undesirable for an application, it is easy to add a field in respIn to indicate this so that session layer can pass duplicate requests back to the application program for a fresh response.

2.6.9 Routing— The reference implementation does not support the functionality of a router node.

2.6.10 Explicit Messages— The reference implementation supports explicit messages with both explicit as well as implicit addressing. For implicit addressing, the application declares bindable tags and uses them as tags for implicit addressing when forming an explicit message. These tags should be bound to msg_in tag of another node for it to have a valid address table entry. For explicit addressing, the application can use non-bindable tags. When using bindable tags, the application can use explicit addressing to override implicit addressing. In other words, if addr field in gp->msgOut (or msg_out) is neither unbound nor turnaround, then it will override any implicit addressing for this tag. gp->msgOut (or msg_out) is re-initialized after each message and hence the address field should be unbound by default. Explicit messages cannot use turnaround addressing.

2.6.11 Address Table Entries— The reference implementation supports more than 15 address table entries. The constant NUM_ADDR_TBL_ENTRIES in custom.h can be used to define the size of the table. The field addressCnt in readOnlyDataStruct has only 4 bits and hence a maximum of only 15 can be reported in this field. Network management tools may only support 15 address table entries. Another problem with using more than 15 address table entries is the limitation of 4 bits for address table index in network variable configuration structure. Thus, network variables cannot use more than 15 address table entries unless these structures are modified somehow to be backward compatible and at the same time allowing new management tools to handle them properly. The only use of these additional entries at present is with implicit addressing for explicit messages. Again, there is a minor problem with this. Normally, bindable tags are used for implicit addressing. Tags in reference implementation are nothing but a 2 byte integer. Non-bindable tags start with the number NUM_ADDR_TBL_ENTRIES. Bindable tags are in the range 0..NUM_ADDR_TBL_ENTRIES -1. The one byte field mtagCount in SNVTStruct keeps track number of bindable tags in a node and this field is used by the management tools. This introduces an upper bound of 255 for the number of bindable tags. Thus, the true limit on number of bindable tags is min(NUM_ADDR_TBL_ENTRIES -1, 255). Since bindable tags are normally associated with address table entries by the management tools, they cannot handle more than 15 entries. Thus, the only way to use these extra entries is to somehow manage them internally. As long as the application program uses bindable tags, the reference implementation will use implicit addressing. The options are either to fool the management tool into thinking that we have less

bindable tags than we actually have (by changing mtagCount) or to introduce a new type of tag for internal use explicitly for this purpose. The function NewMsgTag can be modified to handle this. Yet another way is to use a tag (in the range 16..NUM_ADDR_TBL_ENTRIES-1) in the application program without actually declaring it using NewMsgTag. It is not the purpose of the Reference Implementation to support more than 15 address table entries.

2.6.12 Network Variables or Implicit Messages — The reference implementation supports network variables and all the related features as found in Neuron C programming manuals. Since the reference implementation does not use a Neuron C like compiler, the application program depends on function calls for registering and updating network variables. Synchronous variables, polling of variables, arrays are also supported. The details are explained in the API section.

2.6.13 Neuron C Compatibility.— Since the reference implementation uses modern naming conventions, the data structure names and field names are different from the ones found in Neuron C. However compatible structures and functions have been defined to minimize the porting of Neuron C code to run on the reference implementation.To enable the use of such variables, the constant NEURON_STRUCTURES_NEEDED is defined in lontalk.h. If you do not need this feature, you can comment this constant. If this constant is defined, variables such as msg_in, msg_out etc. are defined and used by the reference implementation. When functions such as msg_send() are called, these structures are copied into corresponding structures in reference implementation (gp->msgOut, gp->respOut etc.) before actual processing. Similarly, when functions such as resp_receive() are called, the reference implementation copies the native structures (such as gp->respIn) to Neuron C like structures (such as resp_in). Thus, there is some overhead involved in using Neuron C like structures.

2.6.14 When Statements — Neuron C program is based on events. The when clauses specify events that are monitored by the scheduler and executed when the event happens. In reference implementation, the scheduler is much simpler. The scheduler simply calls the application program function DoApp(). In order to capture events such as completion of a message, the application program should provide some call back functions. Thus, the application defines several functions corresponding to the various events such as NVUpdateOccurs and the application layer will call these functions to communicate with the application. The application program itself can be written using a sequence of if statements to mirror the sequence of when statements. There is no concept of priority when clauses. It is up to the application program to manage the order of execution of these if statements. For instance, the application program can use a state variable to determine what to do when the application program is called by the scheduler next time. Timer events are handled with a call to MsTimerExpired() function to check for timer expiry.

2.6.15 Extended Statistics— The reference implementation collects some additional statistics that are not available in a Neuron Chip based node. To allow room for expansion in the list of original list of statistics collected, the reference implementation defines space for 11 new statistics. This expansion region is followed by some extended statistics. Read/Write memory command with stat relative can be used to retrieve/update these values.

2.6.16 Explicit Network Management Messages— If the application program generates an explicit request message that is a network management or diagnostic command, the corresponding response is forwarded to application program to handle the response. This is unlike Neuron C where the responses to network management/diagnostic commands are handled by the application

layer itself. The application program can call already existing functions to handle such messages, if needed.

2.6.17 Handling of Address Table Entries— When explicit messages are sent, any implicit address table entries (through the use of bindable tags) are copied at the time msg_send() function is called. Thus, while the message is waiting in the queue, if the node is goes unconfigured, address table entry is changed, and becomes configured again, the new entry will not affect the message already in the queue. However, for network variables, the reference implementation only schedules these variables for generating NVUpdate messages. This message generation is suppressed when the node is unconfigured. If the address table entries are changed in the mean time, these new address table entries will be used for the network variables in the nv queue when the node goes configured. Again, the new entries do not affect those messages already generated using old entries (which will be in transport or session or network layer queue). To get a predictable behavior, it is recommended that the application is off-line for a while to enable all pending messages to be flushed out before bringing it back on-line.

2.6.18 Flex Domain— The reference implementation supports any number of flex domain messages outstanding at any point of time. For example, it can receive several request messages in flex domain and the responses for these messages will use the flex domain in the corresponding request.

3 MC68360 IMPLEMENTATION

This section describes the interface to the physical layer. These functions are primarily the interrupt driven portions of the MAC layer and its interface to the Link layer. The following section describes the overall software design.

The Neuron Chip communication port supports 3 different modes of operation: single ended mode, differential ended mode, and special purpose mode. The single and differential ended modes both involve direct access to the channel through analog circuitry in a transceiver. The special purpose mode requires a smart transceiver that executes a digital serial handshake protocol with the Neuron and the nature of the signal on the channel is hidden from the Neuron Chip's communications port. The difference between the single ended and differential ended modes is that the Neuron Chip's communications port has an additional analog stage built in for the differential mode that includes driver circuitry. The 68360 does not have any similar analog circuitry. To implement the differential mode will require external circuitry to replicate the Neuron Chip's additional analog stage. Echelon makes an interface pod for the PCC-10 card that can provide this capability. Otherwise the single ended and differential ended implementations are identical and will henceforth be referred to together as direct mode.

3.1 Special Purpose Mode

In special purpose mode the 68360 and the smart transceiver simultaneously and continuously exchange a sequence of 16 bit words over a synchronous serial interface [Echelon 91]. In each word the first 8 bits include status and control information and the second 8 bits contain data if any. The interface on the 68360 (transceiver) consists of a transmit (receive) pin, receive (transmit) pin, clockout (clockin) pin, and frame clockout (frame clockin) pin. Each 16 bit word is delimited by the framing clock signal. The SPI port on the 68360 provides most of the needed functionality for this interface. Specifically the SPI port provides for simultaneous clocked double buffered send and receive with the 68360 acting as master. Two significant differences in the Neuron Chip operation from standard SPI interfaces create complications in the implementation.

3.1.1 Continuous Transfer— One is the requirement for *continuous transfer.* Normally SPI processing would have the Master initiate a transfer by enabling its output clock. Once the correct number of bits have been transferred, the Master stops the output clock thereby stopping transfer. The Master can then process the transferred data and prepare for another transfer. With continuous transfer the Master never stops sending and receiving. This means storage must always exist for new receptions and valid data is always ready and waiting for transmit. Processing of data must occur during subsequent transfers.

The 68360 provides a facility for direct memory access from the SPI port through the SDMA controller. The SDMA controller provides circular queues into memory. Local transmit and receive registers in the SPI allow continuous transfers to/from memory to the SPI port. Continuous transfer can be achieved by appropriate use of the circular queues. Each queue must have at least 3 buffers. Each buffer in the queue will be 16 bits wide.

In the original approach a design was developed that attempted to support continuous mode. The configuration is as follows: The transmit buffer queue on the 360 is initialized with a default transmit status byte. The receive queue should be initialized to be ready for reception. The circular buffer queues should be initialized so that the SPI port will instantly move on to the next buffer as soon as it has transmitted (received) a buffer (continuous mode). Upon completion of a buffer the

SPI will generate an interrupt and simultaneously start onto the next pair of buffers (one receive and one transmit). The interrupt service routine must read the status and data of the last received buffer and then write an appropriate status and data for the transmit buffer following the one currently being transmitted. Where appropriate more than one transmit buffer can be written such as in the case of the transmit of a packet. But each word transfer must be checked to see if the transmission must be aborted due to a collision. For data reception the interrupt service routine has to copy out and splice together the data bytes of each transfer to make a packet. For data transmission the interrupt service routing has to segment the packet and fill in status bytes.the interrupt service routine has to execute in much less than the time to transfer one word (16 bit times).

3.1.2 Continuous Transfer Problems— Continuous transfers do not work however due to a combination of features in the 68360 and thee state machine logic in the PL-20 powerline transceiver used to test special purpose mode. The xcvr requires that when it sets the CTS bit in one frame, a valid data byte must be sent in the very next frame. In the continuous transfer approach, a prediction was made as to which frame a valid data byte would be required. This frame was written to a buffer prior to receiving the frame with the CTS. The assumption being that if the xcvr did not send a CTS in the preceding frame that the xcvr would ignore the Data_Valid bit. Instead the xcvr locks up if it gets a Data_Valid in any frame not following a CTS. This makes continous transfer of frames impossible.

A second approach was tried in which only 8 bytes were transferred at a time. Altenating between status and data. In this approach, the status byte of one frame could be processed and the status byte of the next frame written while the data byte of the first frame was being transferred. However the SPI FIFO is 2 bytes long and is double buffered. The SPI pre–fetches the next word before it finishes sending the current word. This makes it impossible to write the next status byte while reading in the previous data byte.

A third approach was tried by using one of the SCC ports. The FIFO arrangement of the SCC ports, however, results in a similar problem.

3.1.3 Non-Continuous Transfer Mode— Finally it was determined that the powerline transceiver could still communicate with the 68360 even if the transfers are non-continuous. In non-continous mode the bit clock and frame clock pause at the end of each frame. This stops transfers. The last frame transferred is then processed by the 68360. The next frame is then loaded and the clocks restarted. Only one buffer is required thus obviating the circular buffer queues. This is a much simpler approach and is easier to implement. In fact any microprocessor with an SPI port of sufficient speed could implement non-continous special purpose mode. However it is a variance from the official Special Purpose Mode specification. It is recommended that the Special Purpose Mode specification be changed to allow non-continuous transfers. The major difficulty with non-continuous mode is it introduces complications in the channel access algorithm timing.

The non-continuous mode configuration is diagrammed in Fig. 3.1 and the timing is shown in Fig. 3.2. The pin mapping for SPM is given in Tbl. 3.1:

The LonTalk protocol specifies certain channel characteristics and time periods such as Beta1 and Beta2 that are important to the channel access algorithm. The non-continuous nature of the reference implementation's special purpose mode requires some modification to make it compatible with Neuron Chips operating on the same channel. In special purpose mode any actions taken by the host processor with respect to the transceiver happen on a frame by frame basis. Consequently any timing of say starting packet transmission or reception is rounded to the nearest frame. There-

TABLE 3.1 PIN MAPPING

68360	Neuron Equivalent
SPIClk	CP2 (Bit Clock Output)
SPIMOSI	CP1 (TX Input)
SPIMISO	CP0 (RX Input)
TCLK (CLK2) (connected to CP2)	
TXD2	CP4 (Frame Clock Output)

fore the effective frame rate is the critical parameter and not the bit rate at which data is transferred. In non-continuous mode the effective frame period is equal to the actual time to shift out 16 bits plus the time it takes for the interrupt service routine to process the frame. For a given desired frame period, the bit rate must be sped rate up so that the effective frame period is the same or less than the desired frame period. The basic Idea is to make the period between frame syncs in non continous mode equivalent to the period between frame syncs in continous mode. For example the slowest allowed bit rate according to the protocol specification for SPM is 156.25 kbps. This has a frame rate of 156.25/16 = 9.7656 kfps. To match this with non-continuous transfer means that the SPI bit clock must be sped up when transferring frames so that the total time period (= 16 bit clocks + ISR processing time) is the same or less than 16 bit clocks in continuous mode. For a bit rate of 156.25 Khz the frame period is 16 * 6.4 usec = 102.4 usec. Suppose the worst case time for ISR is 58 micro seconds then the time left over for the actual frame is 102.4 - 58 = 44.4 usec. This corresponds to a bit rate of 16 * 1/(44.4 usec) = 360.36 kbps. The closest bit rate greater than this that the 360 supports is 390.625 kbps. The effective frame period becomes 58 usec + (16 * (1/390.625 kbps)) = 58 usec + 40.96 usec = 98kbps and the effective frame rate is 6.12 kfps. Unless the ISR time can be shortened to less than half its current duration there is no way to run the bit rate fast enough to make it to the next LONWORKS® compatible step which is the 312.5 Kbps rate.

3.1.4 Frame Clock— The other major implementation detail is to provide the framing clock. The framing clock comes on at the start of a data transfer and turns off one bit time later. It is not simply a divide by 16 of the SPI clock. In other processors that support pulse width modulation it is possible to have one timer change the state of another timer. The 360 does not provide any direct implementation of such an approach. The most straightforward way to generate the framing clock is to use one of the SCC ports on the 360. In this case the SCC is configured in transparent NRZ mode without preamble or CRC. It continuously transmits 1000000000000000 (1 and fifteen 0's) synchronized to the output clock of the SPI. The SPM XCVR interface provides that the Rx input to the host from the XCVR is valid on falling edge of the bitclock and requires that the Tx output from host to the XCVR is valid on next positive edge of the bitclock. The 68360 SPI, however, both provides and requires that inputs and outputs are valid on the same edge of the bitclock. The particular edge (pos or neg) is configurable. This mismatch requires that the SPI output (Tx) be delayed one half clock period. This requires some external hardware consisting of a D flip flop triggered on the rising edge of ~bitclock to delay the Tx output.

3.1.5 SPM Interrupt Service Routine— An interrupt service routine was used to implement the SPM Mac sublayer functions. This was the most effective approach given the timing critical nature of the SPI to power line transceiver interface. The ISR served 3 main funtions:

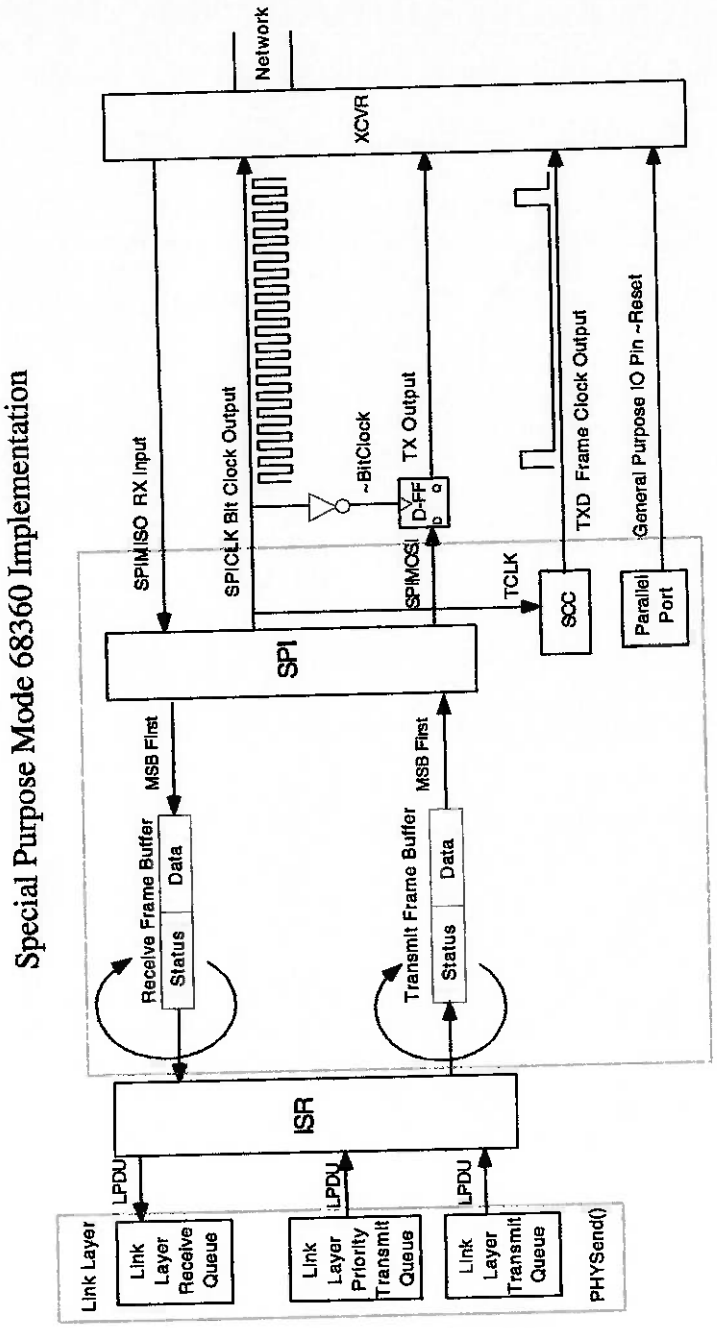


Figure 3.1 Non-continuous Special Purpose Mode Hardware Configuration

a) SPI transfers

b) SPM transceiver handshake state machine

c) Channel access algorithm state machine

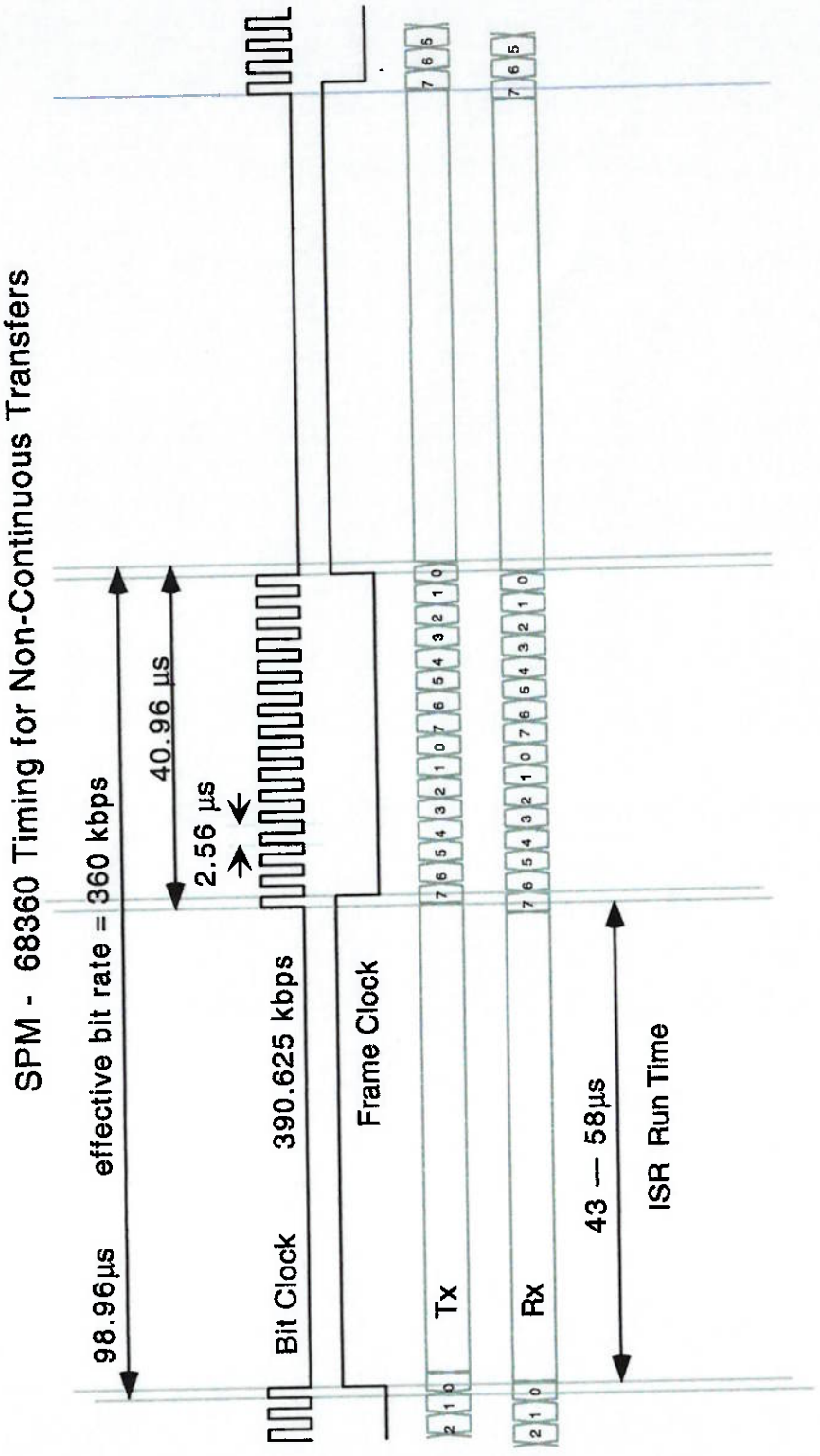


Figure 3.2 Special Purpose Mode Non-continuous Transfer Timing

The ISR would run at the end of each completed SPM frame transfer. The order of execution of each of the ISR functions is as follows: First the received SPI/SPM frame is read in, second the channel access algorithm state machine is executed including the cycle timer code, third the SPM handshake state machine is executed, and finally the transmit SPI/SPM frame is written and and SPI transfer is initiated. The order of execution is important as the interaction between the channel access algorithm state machine and SPM state machine assumes that the channel access algorithm will go first. A diagram of the channel access algorithm state machine is shown in Fig. 3.3. The SPM state machine is shown in Fig. 3.4.

3.2 Direct Mode (Single Ended)

The Neuron chip uses Bi-Phase space encoding with variable length bit sync preambles (all 1's) and a single bit (0) for a byte sync preamble. Bi-Phase space encoding uses one transition at the beginning of each bit time for a 1 and a second transition at the midpoint of a bit time for a 0. Each packet ends with a 16 bit CRC and > 2 bit times of line code violation (no transitions).

The 68360 has four SCC ports. Each port can be configured to automatically support a variety of protocols although LONTalk is not one of them. However, most of the functionality needed for the LONTalk protocol is provided by the SCC ports. An SCC port includes a digital phase locked loop (DPLL) that can be used for synchronizing clocks. The SCC port can also provide carrier sense, encoding and decoding functions and preamble pattern matching. The SCC also provides SDMA access to memory through circular queues.

The attempted 68360 implementation is as follows: The SCC is configured to operate in transparent mode with the DPLL enabled to provide encoding and decoding of FM0 format (same as Bi-phase space encoding). An internal baud rate generator will be used for the nominal bit time clock. The DPLL will use that for transmission and as the basis for generating the receive synchronization clock. The DPLL can be configured to have a base clock rate of 16 times the sync clock at 1.25 Mbps and maximum 32 times for slower channels.

For reception, the preamble sync pattern is set to 1110. The SCC will start dumping bits to a receive buffer as soon as it detects that pattern on the channel. Since LONTalk preambles consist of a variable (depending on the channel type) number of 1's followed by a zero, the SCC will wait until the 0. Each receive buffer should be as large as the largest anticipated packet.

The DPLL can be configured so that a idle is indicated after 1.5 bit times of line code violation. An idle channel is indicated by the CS bit in the SCCS register. A change in CS sets the DCC bit in the SCCE and can cause an interrupt. The SCC does not have a direct way of terminating a packet by noticing line code violations. Consequently it could continue dumping bits into the receive buffer during an idle. The interrupt service routine must manually terminate packet reception and then remove any spurious bits on the end of the packet due to the latency between when the packet really ended and the service routine executed. The latency for entering and interrupt service routine on the 68360 is on the order of 30 clock cycles. This is greater than 1 bit time for the 1.25 Mbps channel (25MHz clock). This makes 1.25 Mbps second operation problematic without external hardware. The interrupt service routine may not be able to resolve the true end of the packet.

The Idle interrupt service routine must also start checking and timing the length of the idle in the case that an outgoing packet is waiting. If the required idle time is reached then it can manually initiate transmission of a packet. It must also copy to a transmit buffer any waiting PPDU's. A multipurpose I/O pin on one of the 68360 ports can be configured for the collision detect input.

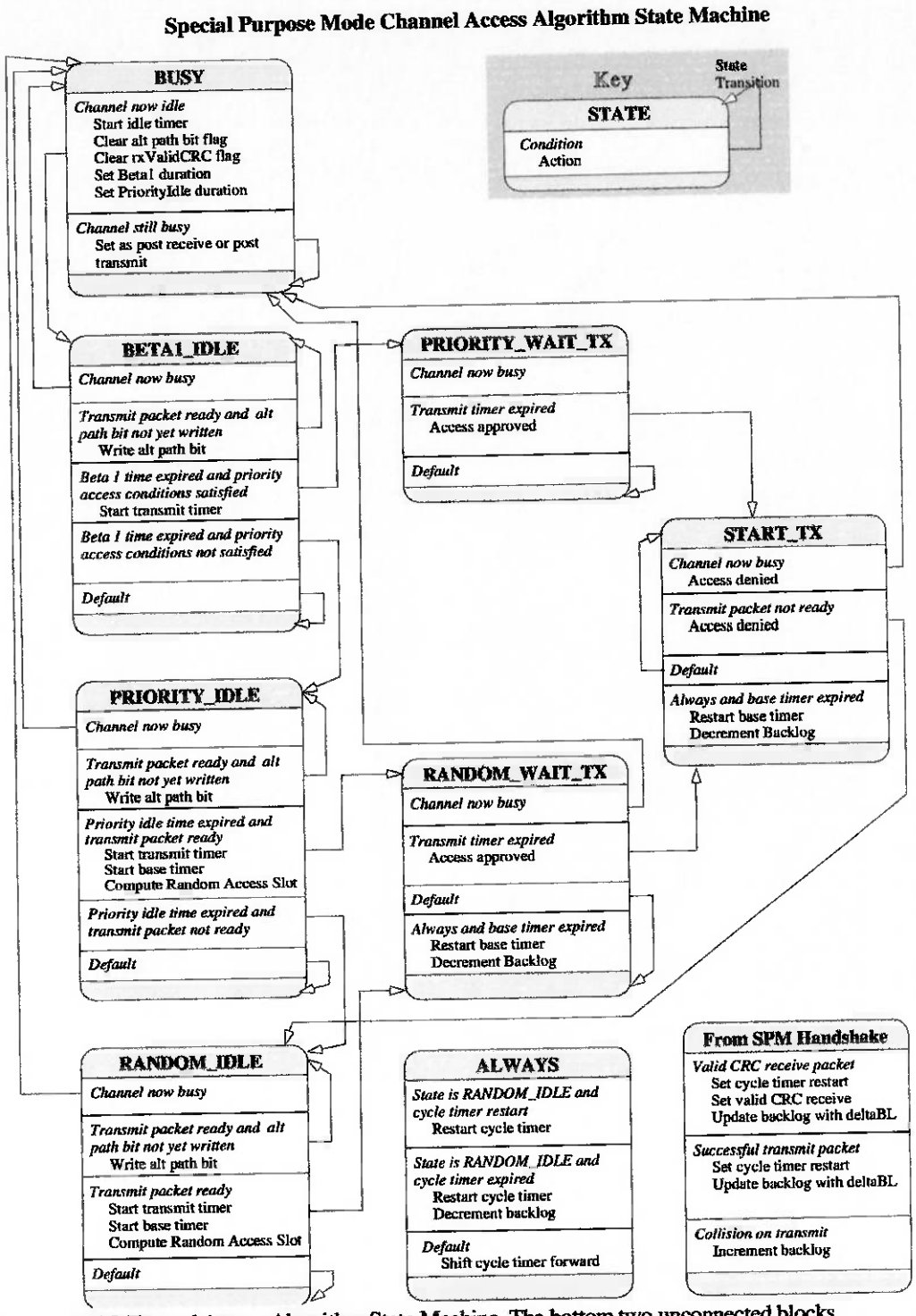


Figure 3.3 SPM Channel Access Algorithm State Machine. The bottom two unconnected blocks labled ALWAYS and From SPM Handshade reprensent code that is executed later in the ISR that is essential to the channel access algorithm but is not formally part of the state machine code section.

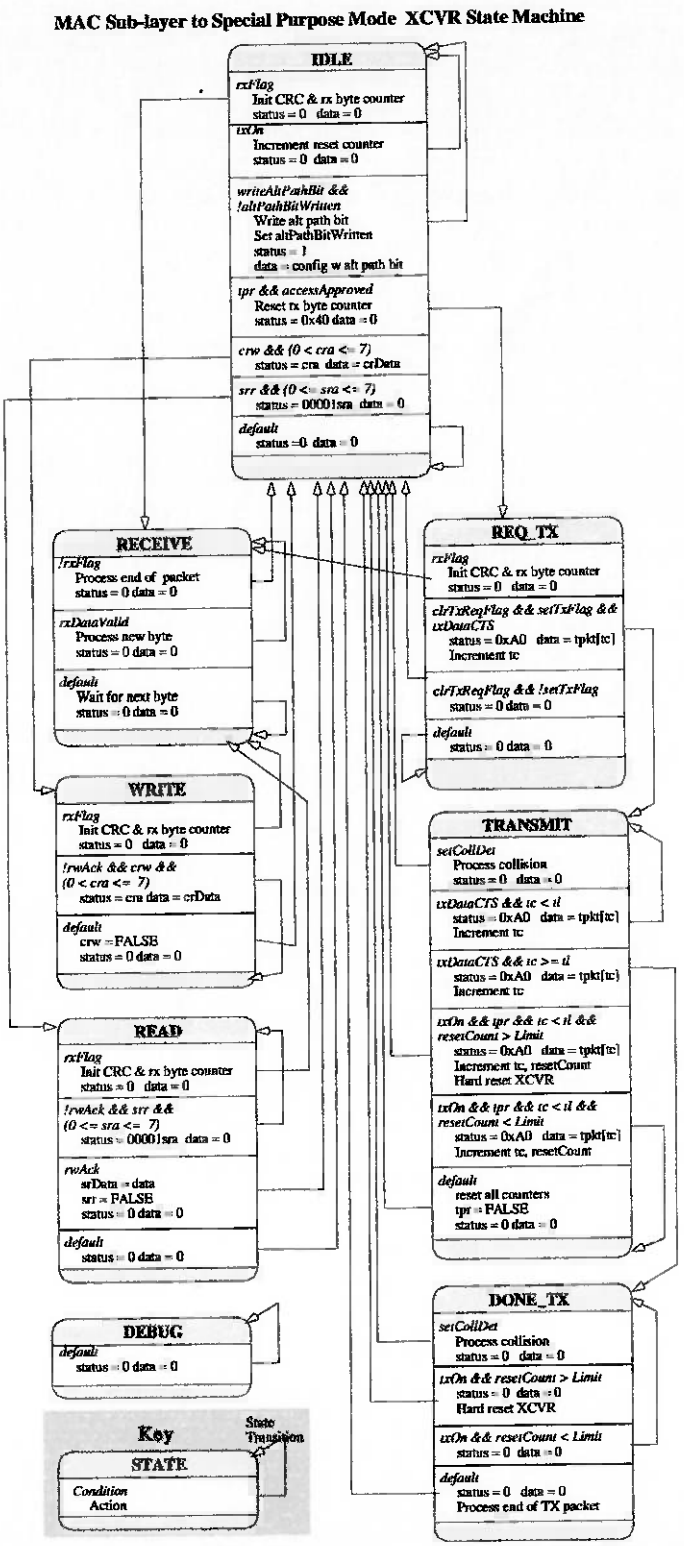


Figure 3.4 Special Purpose Mode 68360 to Transceiver Frame Transfer Handshake State Machine

This pin would be set up to generate an interrupt. The interrupt service routine would manually terminate packet reception..

TABLE 3.2 PIN MAPPING

68360	Neuron Equivalent
RXD2	CP0 (Data Input)
TXD2	CP1 (Data Output)
RTS2	CP2 (Transmit Enable Output)
CTS2	CP3 (Sleep Output)
CD2	CP4 (Collision Detect Input)

Due to hardware bugs in the current revision of the 68360 processor several of the required features of transparent mode do not work as documented. Consequently there is no way to implement a reasonable level of performance of direct mode without external hardware.

4 APPLICATION PROGRAMMER'S INTERFACE (API) DESCRIPTION

4.1 Overview

The reference implementation supports most of the important features of a Neuron C program such as explicit messages, implicit addressing, network variables, and alias variables. Every effort has been made to make the porting of Neuron C programs to run under reference implementation easier. The reference implementation uses modern naming conventions and has hooks for supporting multiple stacks running on a single processor. Thus, there are differences in the naming of structures and fields. For compatibility, structures similar to those in Neuron C are also defined and functions to copy data between the Neuron C format and Reference Implementation format are provided. Thus an application program can use Neuron C like structures or Reference Implementation defined structures, but not both at the same time. This section describes all the details needed to understand how to write application programs for the reference implementation.

4.2 Compilation

The reference implementation uses several .c and .h files. The application programmer normally needs to change only custom.h custom.c and apppgm.c files. The reference implementation includes a make facility to re-compile everything to create the loadable image. See readme file for details regarding compilation, loading, and execution of application program.

4.3 Pragmas

There are no Neuron C equivalent pragma declarations for the reference implementation program. All pragma related initialization can be done using the files custom.h and custom.c. Edit these files and change appropriate values before compiling the code. This section describes each of these constants and their proper use. Most of the constants defined are either related to some pragma or related to fields in readOnlyDataStruct. See Technical Device Data book for more details.

a) MODEL_NUM - Every node has a model number and reference implementation has been assigned 128. Normally, there is no need to change this constant unless you need a different model number.

b) MINOR_MODEL_NUM - This is the minor model number. Default is 0. Normally, there should be no need to change this value.

c) READ_WRITE_PROTECT - Used to protect areas of memory from read or write using network management commands. See Data Book for more details.

d) RUN_WHEN_UNCONF - This is used to indicate whether the application program should run even if the node is in un-configured state. The default is 0 (Do not run). Set it to 1 to run when un-configured.

e) NUM_ADDR_TBL_ENTRIES - This specifies number of address table entries. The maximum value is bounded by 0xFFFF. See Sec. 2.6.11 for more details.

f) RECEIVE_TRANS_COUNT - This specifies number of receive transaction records allocated for this node. For larger incoming traffic, increase the value. If this value is too small, incoming packets could get discarded due to unavailable entry in the receive record pool. If it is too large, then the search time for transport, session, and auth layers will increase every time TPReceive or SNReceive or AuthReceive are called and then the performance degrades.

g) NV_TABLE_SIZE - This specifies the number of network variable table entries. Set this based expected number of network variables used in the application program. For arrays, each entry will consume one network variable table entry. Reference implementation is like host based node and hence you have up to 4096 entries.

h) NV_ALIAS_TABLE_SIZE - This specifies the number of alias table entries used by the node. Alias table entries are used for creating alias entries for primary network variables. These alias entries can have their own selector numbers and helps to resolve some binding problems that are not allowed otherwise. Unless the binding tool can make use of alias tables, there is no use for these entries unless the binding is done manually.

i) SNVT_SIZE - This specifies the number of bytes allocated for snvt structures. If the node has lot of network variables, you need to increase this space to allow information about these variables to be recorded in these structures. See section A.5 of Data Book for more details.

j) APP_OUT_BUF_SIZE - This specifies the number of bytes allocated for application output buffers. This field is encoded using the table on page 9-9 of Data Book Rev. 4.

k) APP_IN_BUF_SIZE - This specifies the number of bytes allocated for application input buffers.

l) NW_OUT_BUF_SIZE - This specifies the number of bytes allocated for network output buffers.

m) NW_IN_BUF_SIZE - This specifies the number of bytes allocated for network input buffers.

n) APP_OUT_Q_CNT - This specifies the number of entries allocated for application output buffers. This field is encoded using table on page 9-10 of Data Book Rev. 4.

o) APP_OUT_PRI_Q_CNT - This specifies the number of entries allocated for application priority output buffers.

p) APP_IN_Q_CNT - This specifies the number of entries allocated for application input buffers.

q) NW_OUT_Q_CNT - This specifies the number of entries allocated for network output buffers.

r) NW_OUT_PRI_Q_CNT - This specifies the number of entries allocated for network priority output buffers.

s) NW_IN_Q_CNT - This specifies the number of entries allocated for network input buffers.

t) NGTIMER_SPCL_VAL - This specifies the receive timer value in seconds used for messages having unique id (i.e. Neuron Id) addressing.

u) NON_GROUP_TIMER - This specifies the receive timer value to be used for messages that do not use group or unique id addressing. This value is encoded using the table on page 9-17 of Data Book Rev. 4. This field is stored in config data structure and hence can be changed by management tools.

v) NM_AUTH - indicates whether network management commands are to be authenticated or not. 0 => no, 1 => yes.

w) NODE_DOC - This is used to store a nodes self documentation string. It should be a C string. The maximum size is 1023 bytes.

x) GROUP_SIZE_COMPATIBILITY - This indicates whether the value of group size indicated in destination address for explicit messages is (actual group size + 1) or actual group size, when the node is not a member of the group. Since the firmware protocol in Neuron Chips always reduce the value by 1 to determine the number of responses or acks expected, the value should be set to (actual group size + 1). In reference implementation, this can be set to either (actual group size + 1) or actual group size depending on whether this constant is defined or not.

y) MAX_NV_ARRAYS - This specifies the maximum number of array network variables that will be used in this node. This is needed to help the protocol allocate some space for storing a table for its internal use.

z) MAX_NV_OUT - This specifies the maximum number of outstanding network output variables that are scheduled due to Propagate function calls at any point of time. A queue with this size is used to schedule network variable updates. If this queue becomes full, further network variable update messages (using Propagate) cannot be processed until space becomes available in this queue.

aa) MAX_NV_LENGTH - This specifies the maximum number of bytes allocated to capture the value of network value variables that are declared as synchronous. When such a variable is updated and propagated, the current value of the variable is copied into the queue and hence this value is useful to indicate maximum size. Note that reference implementation does not use malloc for dynamic allocation after node reset.

ab) MAX_NV_IN - This specifies the maximum number of network input variables that can be scheduled for polling. Every time the application program wants to poll a network input variable, it is placed in a queue whose size is determined by this constant.

ac) MAX_DATA_SIZE - This specifies the maximum size of data array in gp->msgOut or msg_out structure which is used to from explicit messages. This size is independent of APP_OUT_BUF_SIZE but does not make sense for this to be any larger.

ad) TS_RESET_DELAY_TIME - This specifies the timer value in milliseconds used in delaying the transport and session layers for sending any new messages after a power-up or external reset. This is done to avoid sending messages that could be considered as duplicates and tossed by receiving nodes.

ae) MALLOC_SIZE - This specifies the size of array used for allocating storage for all the queues, receive record pools etc. by the protocol. Clearly, the proper value depends on many of the values defined in custom.h and hence only the application programmer can determine this value. If this value is too small, some layer will be unable to do a proper Reset forcing the node to return from main. One can set this to a large value, run the application, and then check the value of gp->mallocUsedSize to compute a more accurate value. Alternatively, once can go through Reset functions of all layers and compute the appropriate value manually. Trial and error approach is probably easier.

In addition to the constants defined in custom.h, the application programmer can also indicate other information about a node in custom.c. This includes information such as the Neuron Id of the node, the number of domains for the node, the subnet/node number, authentication key, etc. These are self-explanatory and the details of the structure definition can be found in custom.h.

4.4 Initialization

Every application program should provide a function called AppInit that will be called once by the main.

```
void AppInit(void);
```

The purpose of this function is to initialize variables, register network variables, tags etc. once during power-up. These initialization are such that they are not performed after every reset.

4.5 Reset

Every application program should provide a function called AppReset that will be called every time the node is reset.

```
void AppReset(void);
```

The purpose of this function is to initialize variables after every reset or do any other work the application program finds appropriate.

4.6 Application Program

Every application program should include <node.h> header file and provide a function called DoApp.

```
void DoApp(void);
```

This function is the entry point for the actual application program. The scheduler calls `DoApp()` once per pass through the round robin schedule loop. The `DoApp()` function has four responsibilities:

a) Process incoming messages, if any. Upon completion of processing a message, call `msgFree()`.

b) Process incoming responses, if any. Upon completion of processing a response, call `respFree()`.

c) Perform application specific processing. (e.g sending messages, responses, updating network variables etc.)

d) Return to the round robin scheduler quickly. This is necessary to allow the protocol stack to process new messages. If the application needs to perform a large amount of processing, it must break the processing up into small pieces, and process one piece per call.

Messages and responses must be processed in a timely function, preferably on each pass through DoApp. If the application messages are not processed quickly enough, messages may be lost because all application input buffers are full.

The user application **must not** define the entry point "`main()`". The reference implementation scheduler defines the entry point.

In addition to the above two functions, the application program must define the following functions:

```
void  MsgCompletes(Status stat, MsgTag tag);
   This function is called whenever a transaction initiated by the application
   is completed. stat will be either SUCCESS or FAILURE. This function has the
   same role as the msg_completes event of the Neuron C program.
void  NVUpdateCompletes(Status stat, int16 nvIndex, int16 nvArrayIndex);
   This function is called whenever a network variable update or poll either
   succeeds or fails. stat will be either SUCCESS or FAILURE. nvIndex is the
   index of the network variable. nvArrayIndex is the array index if nvIndex
```

```
   corresponds to a network array variable.
void  NVUpdateOccurs(int16 nvIndex, int16 nvArrayIndex);
   This function is called whenever a network variable has been updated. The
   nvIndex is the index of the variable that was updated. nvArrayIndex is array
   index if nvIndex corresponds to an array network variable.
void  Wink(void);
   This function is called when the node receives the wink network management
   message. The application can do whatever it wants.
void  OfflineEvent(void);
   This function is called just before the application is placed offline.
void  OnlineEvent(void);
   This function is called just before the application is placed online.
```

4.7 Explicit Messages and Responses

Since the reference implementation has hooks for supporting multiple stacks, a global structure is used for each stack. Before calling any function, the scheduler sets pointers gp, eep, and nmp to point to the global stack data, EEPROM data, and neuron memory mapped data. Within the global stack data, the reference implementation has variables msgIn, msgOut, respIn, respOut, msgReceive, and respReceive. These are similar to the ones defined in Neuron C. msgReceive is a boolean variable that is set to true whenever there is a message available in msgIn. Similarly, respReceive is true if there is a response to be received in respIn.

To make porting of existing Neuron C programs easier, the reference implementation can create variables such as msg_in, msg_out etc if the constant NEURON_STRUCTURES_NEEDED is defined in lontalk.h. If this constant is defined then the application program should only use variables similar to the ones defined for Neuron C. Otherwise, the application programs should variables defined in reference implementation (gp->msgIn, gp->msgOut etc). Some of these structures have additional fields in reference implementation that should be filled by the application program (for example, len field is required for msg_out). See api.h for details of these structures. The following descriptions use Neuron C variables but the same holds for variables such as gp->msgOut that are unique to reference implementation.

```
extern msg_in_type      msg_in;     // For incoming message
extern msg_out_type     msg_out;    // For outgoing explicit message
extern resp_in_type     resp_in;    // For incoming response
extern resp_out_type    resp_out;   // For outgoing response
extern nv_in_addr_type  nv_in_addr; // The source address of incoming message
extern int16            nv_array_index; // array index for nv array var update

Boolean  MsgAlloc(void); // Returns TRUE if explicit message can be formed.
Boolean  msg_alloc(void); // For Neuron C compatibility. Same as MsgAlloc.
Boolean  MsgAllocPriority(void);  // Similar to MsgAlloc for priority messages.
Boolean  msg_alloc_priority(void); // For Neuron C Compatibility.
void     MsgSend(void);           // To send an explicit message.
void     msg_send(void);          // For Neuron C Compatibility.
void     MsgCancel(void);         // Does nothing. For backward compatibility.
void     msg_cancel(void);        // For Neuron C Compatibility.
void     MsgFree(void);           // Releases data in msgIn to receive new one.
void     msg_free(void);          // For Neuron C Compatibility.
Boolean  msgReceive(void);        // TRUE if there is msg in msgIn.
Boolean  msg_receive(void);       // For Neuron C Compatibility.
```

```
Boolean  RespAlloc(void);          // Returns TRUE if response can be formed.
Boolean  resp_alloc(void);         // For Neuron C Compatibility.
void     RespSend(void);           // Send the response in resp_out.
void     resp_send(void);          // For Neuron C Compatibility.
void     RespCancel(void);         // Does nothing. For backward compatibility.
void     resp_cancel(void);        // For Neuron C Compatibility.
void     RespFree(void);          // Releases data in respIn to receive new one.
void     resp_free(void);          // For Neuron C Compatibility.
Boolean  RespReceive(void);       // Check if there is any response to receive.
Boolean  resp_receive(void);       // For Neuron C Compatibility.
```

4.7.1 Receiving Messages—

```
extern   msg_in_type     msg_in;  // For incoming message
Boolean  msg_receive(void);       // For Neuron C Compatibility.
void     msg_free(void);          // For Neuron C Compatibility.
```

When the protocol receives a message addressed to this node, the data from the message is written to the `msg_in` structure, and the next call to function `msg_receive()` will return `TRUE`. The application uses the data in `msg_in`, and then calls `msg_free()`. The `msg_free` function enables a new message to be placed in *msg_in*. When another message is received, or if a message is already buffered, it is placed in `msg_in`, and the next call to the funciton `msg_receive()` will return `TRUE`. Thus the application can get at most one message per cycle. If the application calls *msg_receive*() but does not call *msg_free*(), the reference implementation will automatically call *msg_free*().

4.7.2 Receiving Responses—

```
extern   resp_in_type    resp_in; // For incoming response
Boolean  resp_receive(void);      // For Neuron C Compatibility.
void     resp_free(void);         // For Neuron C Compatibility.
```

When the protocol receives a response to a request made by this node, the data from the response is written to the `resp_in` structure, and the next call to function `resp_receive()` will return `TRUE`. The application uses the data in `resp_in`, and then calls `resp_free()`. The `resp_free` function enables a new response to be placed in *resp_in*. When another response is received, or if a response is already buffered, it is placed in `resp_in`, and the next call to the function `resp_receive()` will return `TRUE`. Thus the application can get at most one response per cycle. If the application calls *resp_receive*() but does not call *resp_free*(), the reference implementation will automatically call *resp_free*().

4.7.3 Sending Messages—

```
extern msg_out_type    msg_out;// For outgoing explicit message
Boolean  msg_alloc(void);      // For Neuron C compatibility. Same as MsgAlloc.
Boolean  msg_alloc_priority(void); // For Neuron C Compatibility.
void     msg_send(void);           // For Neuron C Compatibility.
```

Before sending a message, the application calls `msg_alloc()` or `msg_alloc_priority()` for a priority message. If the allocation function returns `FALSE`, then an application output buffer is not available, and application must try again later. These functions simply check if the corresponding application output buffer has space for storing a message or not.

When the allocation function returns `TRUE`, an application message out buffer is allocated for the current message. The application sets various fields in `msg_out` to construct the message. The

field *len* is new and is **required**. The value of *len* is the number of bytes stored in data array. The remaining fields are as done in Neuron C. When the message in `msg_out` is ready to be sent, the application calls `msg_send()`, and the message is sent. There is no need to call the function *msg_cancel*() as it does nothing. The structure *msg_out* is re-initialized by the reference implementation after each *msg_send*().

In order for the application program to receive message completion, the application program should define the function:

```
void    MsgCompletes(Status stat, MsgTag tag);
```

When a message completes, the API calls the user defined function `MsgCompletes()` to return the completion status of the message. The tag parameter is the value from the tag field of the `msg_out` structure.

4.7.4 Sending Responses—

```
extern resp_out_type   resp_out;   // For outgoing response
Boolean  resp_alloc(void);         // For Neuron C Compatibility.
void     resp_send(void);          // For Neuron C Compatibility.
```

When the application receives a message containing a request, the application returns a response. Before sending a response, the application calls `resp_alloc()`. If `resp_alloc()` returns `FALSE`, then an application response buffer is not available, and application must try again later. The function simply checks if there is space for a new response in the response queue for session layer.

When `resp_alloc()` returns `TRUE`, the application can send a response. The application sets various fields in `resp_out` to construct the response. The field *len* is required and represents the number of bytes in data array of the response. The field *req_id* is used to indicate the request id of the corresponding request. This is used for retrieving the matching request from the receive record pool. The value of 0 is an invalid request id and is used by the reference implementation to detect the fact that the application program did not initialize this field and hence it automatically sets it to the request id of the request last received in *msg_in*. The field *null_response* can be set to true to indicate that it is a null response (one which is not sent out). This field is used by the session layer to mark the request as done. A well behaved application program is expected to respond to every request. A null response is provided for this purpose. The reference implementation will continue to work without any problem even if a response is not sent by the application program.When the response in `resp_out` is ready to be sent, the application calls `resp_send()`, and the response is sent. There is no need to call the function *resp_cancel*() as it does nothing. The structure *resp_out* is re-initialized by the reference implementation after each *resp_send*().

4.8 Network Variables

Network variables are declared in the application program in a different way than done in the Neuron C program. The API defines the following structure to help the declaration of network variables.

```
typedef struct
{
    Bits  priority     : 1; /* TRUE or FALSE */
    Bits  direction    : 1; /* NV_INPUT or NV_OUTPUT */
    Bits  selectorHi   : 6; /* Present only for non-bindable */
```

```
    Bits  selectorLo    : 8; /* Present only for non-bindable */

    Bits  bind          : 1; /* 1 => bindable 0 ==> nobindable */
    Bits  turnaround    : 1; /* TRUE or FALSE */
    Bits  service       : 2; /* ACKD, UNACKD_RPT, UNACKD */
    Bits  auth          : 1; /* TRUE or FALSE */
    Bits                : 3; /* Unused */

    Bits explodeArray : 1;   /* Explode arrays in SNVT structure */
    Bits  nvLength    : 7;   /* Length of network variable in bytes.
                                For arrays, give the size of each item. */

    uint8  snvtDesc;         /* snvtDesc_struct in byte form. Big_Endian */
    uint8  snvtExt;          /* Extension record. Big_Endian. */
    uint8  snvtType;         /* 0 => non-SNVT variable. */
    uint8  rateEst;
    uint8  maxrEst;
    uint16 arrayCnt;         /* 0 for simple variables. dim for arrays. */
    char  *nvName;           /* Name of the network variable */
    char  *nvSdoc;           /* Sel-doc string for the variable */
    void  *varAddr;          /* Address of the variable. */
} NVDefinition;
```

For each network variable, the application must define a structure and initialize it with appropriate values. This structure is similar to the network variable declarations with bind information. There are no defaults and hence you should provide all the values. For each network variable, the application should also declare storage for that variable and an index to communicate with API. For example,

```
nint intIn; /* Polled network input variable. */
NVDefinition intInDef =
{
   FALSE, NV_INPUT, 0,    0,   1, FALSE, ACKD,    FALSE,
   0, sizeof(nint), 0xA0, 0x30, 0,  0,  0,
   0,  "intIn",  "intInSD", &intIn
};
int16 intInIndex;
```

Note that the type *nint* is used. In place of *nint*, one can use almost any type. The following predefined types are convenient to use to declare neuron equivalent types.

```
typedef char                int8;
typedef short int           int16;
typedef long int            int32;
typedef unsigned char       uint8;
typedef unsigned short int  uint16;
typedef unsigned long int   uint32;
/* Neuron C Definitions for int long etc. */
typedef int8                nshort;
typedef int8                nint;
typedef uint8               nuint;
typedef uint8               nushort;
typedef int16               nlong;
```

```
typedef uint16                nulong;
```

The network variable must be registered explicitly using the function *AddNV* in AppInit function. Even though you can initialize the storage for the variable in the declaration statement, you should also do it in AppReset function to ensure that the variable has those values after every node reset (unless the logic calls for otherwise). The following code illustrates this procedure.

```
Status AppInit(void)
{
   /* Register Network variables */
   intOutIndex       = AddNV(&intOutDef);
   longOutIndex      = AddNV(&longOutDef);
   intArrayOutIndex = AddNV(&intArrayOutDef);
   intInIndex        = AddNV(&intInDef);
   longInIndex       = AddNV(&longInDef);
   intArrayInIndex   = AddNV(&intArrayInDef);

   /* Make sure we were successful in registering all variables */
   if (intOutIndex       == -1 ||
       longOutIndex      == -1 ||
       intArrayOutIndex == -1 ||
       intInIndex        == -1 ||
       longInIndex       == -1 ||
       intArrayInIndex   == -1)
   {
      return(FAILURE);
   }


   tag0          = NewMsgTag(BINDABLE);
   tag1          = NewMsgTag(BINDABLE);

   /* Make sure we got the tags successfully */
   if (tag0 == -1 || tag1 == -1)
   {
      return(FAILURE);
   }

   return(SUCCESS);
}
```

AddNV returns -1 if it fails to register. It is a good idea to test for this and return FAILURE for AppInit to indicate to the scheduler that there was a problem.

4.9 Network Variable Related Functions

There are several functions in the API to support registering, updating, polling, propagating of network variables. They are defined below:

```
/*************************************************************************
   Adds a new network variable. This involves adding an entry into nvConfig-
   Table, nvFixedTable, SNVT information, if present etc. The return value is
   the index assigned to the variable. For arrays, each element is like a sep-
   arate network variable. So, multiple entries are added to the
```

```
   tables. However, only the base index is returned.
*************************************************************************/
int16 AddNV(NVDefinition *dp);
/* NVUpdateCompletes is called when an nv update or nv poll completes. The 2nd
   parameter is the array index for array variables, 0 for simple variables. */
void  NVUpdateCompletes(Status stat, int16 nvIndex, int16 nvArrayIndex);

/* NVUpdateOccurs is called when an input nv has been changed. The 2nd
   parameter is the array index for array variables, 0 for simple variables. */
void  NVUpdateOccurs(int16 nvIndex, int16 nvArrayIndex);

/* To send all network output variables in the node.
   Polled or not, Use Propagate function. */
void Propagate(void);

/* To send one simple network variable or a whole array */
void PropagateNV(int16 nvIndex);

/* To send an array element or any other simple variable.*/
void PropagateArrayNV(int16 arrayNVIndex, int16 index);

/* To poll all input network variables */
void  Poll(void);

/* To poll a specific simple input network variable or an array */
void  PollNV(int16 nvIndex);

/* Poll a specific array element or any other simple variable. */
void PollArrayNV(int16 arrayNVIndex, int16 index);
```

The reference implementation supports implicit messages only through the use of explicit calls to one of the three Propagate functions. Since there is no Neuron C like compiler, there is no way to modify the code based on the fact that a variable got updated and hence the need for these explicit calls. The Poll functions are as used in Neuron C.

A network variable update may involve one primary and 0 or more alias entries. For each, the address table entry can have difference address formats. Thus, it is possible that several nv update messages are sent for one network variable update. A network variable update is said to succeed if every transaction scheduled for that variable succeeds. The value of the parameter *stat* in *NVUpdateCompletes* indicates whether the network variable update succeeded or failed. Note that it is not a boolean variable but rather of type Status. So, you should compare against SUCCESS or FAILURE.

A network variable poll may involve one primary and 0 or more alias entries. For each, the address table entry can have different address formats. Thus, it is possible that several nv poll messages are sent for one network variable poll. A network variable poll is said to succeed if it is turnaround only or every transaction (not including turnaround) succeeds and at least one response contains valid data. Thus, a turnaround only poll will always succeed. As another example, if a network variable poll is both turnaround and connected to a network output variable in another node and that node returns null data, then the poll will fail. The value of the parameter *stat* in *NVUpdateCompletes* indicates whether the network variable poll succeeded or failed. Note that

it is not a boolean variable but rather of type Status. So, you should compare against SUCCESS or FAILURE.

4.10 Network Variables and Address Table Entries

Due to limitation of 4 bits for address table index in network variable config structure and the fact that network management tools may not support more than 15 address table entries, the reference implementation does not support implicit addressing for network variables using address table entries beyond the 15th entry. It is conceivable for someone to modify the code so that this is possible and at the same time the code is backward compatible. One easy way this can possibly be done is to use an additional table for the network variables and store only the index of the address table entry that should be used for each network variable. This entry can be used only if the original network variable table indicates that the network variable is not bound to any address table entry. If both indices are unbound, then the network variable is not bound to an address table entry. The functions that handle nv update and poll should be modified to take care of this change by setting a local variable representing the address table index appropriately. Note that the address table entries should be initialized by the application program somehow (either using custom.c or some other way). This is just a suggestion and there is no guarantee that this technique work well as it has not been tried. Even if it works, care should be taken to make the code inter-operable with existing tools and neuron nodes. It is best to use this for internal use when the situation demands the use of network variables with lots of address table entries.

4.11 Message Tags

The application program can define either bindable tags or non-bindable tags in the application program. The following function is used to declare new tags.

```
/* To get a new message tag. NewMsgTag(BIND) or NewMsgTag(NOBIND) */
MsgTag NewMsgTag(BindNoBind bindStatusIn);
```

The number of bindable tags is usually bounded by 15 but can be up to 0xFF. The number of bindable tags declared in a node is stored in a field in SNVT structures and is accessible to management tools. Thus, one should limit the number of bindable tags to no more than 15 for the purpose of interoperability. Each bindable tag consumes an address table entry and hence it is wise not to use up all these entries if you also want to bind network variables. Bindable tags are used for implicit addressing. An application program can use large number of non-bindable tags. The limit on number of non-bindable tags is 0xFFFF - NUM_ADDR_TBL_ENTRIES.

The reference implementation will use implicit addressing if the tag value supplied in *msg_out* structure is less than NUM_ADDR_TBL_ENTRIES and the destination address in *msg_out* is unbound or turnaround. Since tag is just a number, an application can use tag values > 15 but less than NUM_ADDR_TBL_ENTRIES to use implicit addressing. Such tags should be declared explicitly. However, these address table entries should be explicitly initialized by the application program either using custom.c or some other way. Non-bindable tags are primarily used for correlation with message completion.

4.12 Miscellaneous Functions

The following are some miscellaneous functions that an application program can call.

```
/* Application can call this fn to put itself offline */
void GoOffline(void);
/* Appplication can call this fn to put itself unconfigured */
void GoUnconfigured(void);
```

```
/* To send a service pin message */
Boolean ServicePinMessage(void);
```

4.13 Alias Tables

The reference implementation does support the concept of alias tables that are used to perform more complex binding of network variables. Some older tools may not support the use of alias tables. However, an application can initialize the alias tables based on known indices (Network variables are registered in the order in which they are given starting with an index value of 0) using custom.c file. Alternatively, a modern management tool that supports alias tables can be used. In either case, the application program should define the number of alias table entries available in custom.h. This information is stored in SNVT structures and is available to network management tools.

4.14 Software Timers

The reference implementation allows the application program to use any number of software timers. All software timers are maintained using a single hardware timer. The software timers are not automatically updated. The application needs to call either UpdateMsTimer or MsTimerExpired. Only millisecond timers are supported. The following is the definition for the software timer structure.

```
/* Software Timer definition. */
typedef struct
{
   uint32 curTimerValue;     /* Number of clock ticks left in timer.      */
   uint32 lastUpdatedTime;   /* The time when the timer was last updated. */
   Boolean expired;          /* TRUE => it has already expired.           */
} MsTimer;
```

The functions that are related to the use of software timers are:

```
void    SetMsTimer(MsTimer *timerOut, uint16 initValueIn);
void    UpdateMsTimer(MsTimer *timerInOut);
Boolean MsTimerExpired(MsTimer *timerInOut);
```

SetMsTimer is used to initialize the timer to given number of milliseconds. *UpdateMsTimer* is used to update the timer. There is no harm to update a timer that has already expired. The field *curTimerValue* indicates the number of milliseconds left before expiry. Instead of using *UpdateMsTimer*, an application program can use *MsTimerExpired* (similar to timer expiration event in Neuron C) to check if a timer has expired. *MsTimerExpired* will return true only during the first time the timer expired. If the timer has already expired, the function *MsTimerExpired* will return false. *MsTimerExpired* calls *UpdateMsTimer* and checks the curTimerValue to determine whether the timer expired or not. The reference implementation does not support repeating timers. The application program should re-initialize the timer upon expiry.

```
Example:
      MsTimer myTimer;

      SetMsTimer(&myTimer, 1000); /* For 1 second */
      /* Do some processing */
      :
      if (MsTimerExpired(&myTimer))
```

```
{
    /* Do whatever you want */
    :
    SetMsTimer(&myTimer, 1000); /* Reset if needed */
}
```

5 REFERENCES

Arnewsh 95	SBC360/EC User's Manual Revision 1, Arnewsh Inc.1995
Echelon 95	Lontalk Protocol Specification Version 3.0 078-0125-01A ,Echelon Corporation1995
Echelon 96	LonWorks Protocol Layer 1 Timing, Preliminary, Echelon Corporation 1996
Echelon 91	Neuron Chip Special-Purpose Mode Transceiver Interface Specification, LonWorks Engineering Bulletin, Echelon Corporation 1991
Echelon 94	NV Connectivity Extensions: The Aliasing Alternative, Revision 2.2 April 1994, Echelon Corporation
Harbison 95	Harbison S. P. and Steele G. L. Jr., C A Reference Manual, Prentice Hall 199? ISBN 0-13-110933-2
Motorola 97	LonWorks Technology Device Data DL159/D Rev4, Motorola 1997.
Motorola 95	MC68360 Quad Integrated Communications Controller User's Manual MC68360UM/AD REV 1, Motorola 1995
Motorola 90	M68300 Family CPU32 Central Processor Unit Reference Manual CPU32RM/AS REV1, Motorola 1991
SDS 96a	CrossCode User Guide Version 7, Software Development Systems 1996
SDS 96b	SingleStep User Guide Version 7, Software Development Systems 1996