Universidade Estadual de Campinas
FACULDADE DE ENGENHARIA ELÉTRICA
DEPARTAMENTO DE TELEMÁTICA

UNICAMP

Descrição e Implementação Formal de um Protocolo de Acesso para Redes Locais de Fibra Óptica

Tese apresentada à Faculdade de Engenharia Elétrica da Universidade Estadual de Campinas, como parte dos requisitos exigidos para a obtenção do título de Mestre em Engenharia Elétrica.

por

Gorgonio Barreto Araújo
Engenheiro Eletricista – FEE/UNICAMP

em 12 de agosto de 1994 perante a banca examinadora

Akebo Yamakami – **Orientador**

Ivanil Sebastião Bonatti
Vitório Bruno Mazzola

Este exemplar corresponde à redação final da tese defendida por Gorgônio Barreto Araújo e aprovada pela Comissão Julgadora em 12.08.94.

"A web of glass spans the globe. Through it, brief sparks of light incessantly fly, linking machines chip to chip and people face to face", Vinto G. Cerf

Dedico este trabalho ao meu filho Luan, que pertence à geração da era da comunicação.

Agradecimentos

Agradeço a oportunidade de fazer este trabalho a Motoyama. O apoio recebido dos colegas de projeto: Maria Cristina, Paulo Pessoa e Cláudia. A André Goldstein, que muito ajudou numa fase da implementação. A Renato Figueiredo e Ilka Barros, que ajudaram-me com o *V*HSIC *H*ardware *D*escription *L*anguage (VHDL). Ao Centro de Desenvolvimento Científico e Tecnológico (CNPQ) que financiou a viagem na qual conheci o VHDL. A Miguel que nunca faltou com o seu apoio técnico. A Ivanil que decisivamente nos apoiou nos momentos difíceis. Ao *D*epartamento de *T*elemática (DT), pela excelente infra-estrutura proporcionada. Ao *C*entro de *Pesq*uisa e *D*esenvolvimento (CPqD), nas pessoas de Arlindo, Rogério e José Roberto que possibilitou a utilização do Lab. de Micro-eletrônica. Ao grupo de desenvolvimento do *P*rocessador *P*referencial (PP), que, com muita gentileza, nos forneceu apoio e documentação sobre o PP. À comissão técnico científica do *L*aboratório de *C*omputação *A*plicada à *E*ngenharia *E*létrica (LCAEE), pela infra-estrutura criada e mantida neste laboratório e por viabilizarem a apresentação deste trabalho no anfiteatro do laboratório. A José Raimundo, por viabilizar a utilização do *software* da *M*entor *G*raphics *C*orporation (MGC), na *F*aculdade de *E*ngenharia *E*létrica (FEE).

Agradeço a ajuda direta e indireta na editoração da tese de Ricardo Lüders, Lúcia, Airto, Leandro Dibb, Eduardo e Pedro Gapski.

Agradeço especialmente ao amigo, mestre e orientador Akebo, que não só sempre deu todo o seu apoio e orientação, mas descisivamente colaborou para a minha formação.

Agradeço a toda minha grande família, que foi a maior sacrificada nesse processo. Presto especial homenagem a meus pais Ana Julina e Gorgonio, e à minha querida Dalila.

RESUMO

Este trabalho apresenta a especificação formal da camada *M*edium *A*ccess *C*ontrol (MAC) da *R*ede de *Á*rea *L*ocal com *F*ibra *Ó*tica (RALFO) que é uma rede tipo anel de Cambridge modificada, integrando serviços de voz e de dados. O método de acesso ao meio é detalhadamente especificado, desde os serviços atendidos, os quadros, a sua estrutura em blocos e a especificação funcional de cada bloco. Após, sua especificação esquemática e em linguagem de descrição formal é apresentada, utilizando ambiente VHDL. Resultados de testes são apresentados e comentados.

ABSTRACT

This work presents a formal specification of the MAC layer of the RALFO, wich is a modified Cambridge ring type network, integrating voice and data. The medium access method is explaned in detail, showing the block structure and the funcional specification of the each bloc are given. Besides, the schematic specification and in the formal description language in the VHDL environment are presented. Results of the tests are presented and discussed.

Conteúdo

Lista de Figuras

Capítulo 1

Introdução

A disseminação intensa de computadores, aliada a uma forte evolução tecnológica na área de tratamento e transmissão de informações, notadamente nos últimos dez anos, criou para as sociedades humanas o que se convencionou denominar por era da informação ([Der91]).

Podemos notar hoje, presença cada vez mais intensa de computadores, micro-computadores, controladores e processadores microprogramados, robôs, veículos auto-guiados, etc em todos os ambientes: nas empresas, nas fábricas, nas lojas, em setores públicos, nas residências, etc. A necessidade de intercâmbio cada vez mais rápido e eficiente de informações está trazendo avanços na tecnologia para interligar estes equipamentos: fibras ópticas, comunicação sem fio, protocolos sofisticados de comunicação, codificação e decodificação de sinais, criptografia, etc. Além disso, a tendência é de que esta rede suporte vários tipos de serviços, de uma forma integrada (RDSI - Rede Digital de Serviços Integrados).

Por exemplo, a presença de computadores na medicina hoje é bastante intensa, onde as análises e diagnósticos estão cada vez mais informatizados, baseados naquilo que chamamos de processamento de conhecimentos (redes neurais, inteligência artificial, etc). Este tipo de procedimento traz benefícios quando as informações adquiridas tendem a tornar os diagnósticos cada vez mais precisos. Ainda, as operações podem ser auxiliadas e ou efetuadas por sistemas programados, tornando-as também precisas. A comunidade médica pode ter consultas "on-line" entre si e em banco de informações através de redes, transmitindo inclusive imagens.

Um outro setor que recebe influência significativa desta era da informação é a da educação, onde aulas podem ser assistidas à distância e dadas por professores de qualquer parte do planeta, ou podem ser gravadas e reproduzidas independente do local e tempo. As buscas às

informações como em enciplopédias, livros, manuais, etc. tornam-se cada vez mais rápidas e precisas pela rede, trazendo não só textos mas também sons, imagens e movimentos e permitindo enfocar aspectos cada vez mais especializados.

As grandes lojas e instituições, em muitos países, já oferecem serviços e mercadorias anunciadas e realizadas via rede. A integração de serviços oferece uma diversidade cada vez maior de alternativas, inclusive para o lazer, quando torna possível, por exemplo, que se assistam a concertos ou que se visitem museus à distância. Os sistemas vão auxiliar na escolha adequada de produtos ou de serviços ou até de diversões.

A infraestrutura de redes de máquinas interligadas vai modificar substancialmente a forma de trabalho nas organizações e empresas. Vai tornar possível eliminar a necessidade de locomoção dos empregados e de espaço físico das empresas. Trabalhadores poderão ficar em sua casa e efetuar suas tarefas via rede, em horários não fixos. Muitas das tarefas poderão dispensar a presença de homens, em virtude da automatização e de informatização.

Os veículos para locomoção e transporte poderão ser autoguiados, tendo computadores à bordo fornecendo as informações desejadas ou necessárias através de rádio-comunicação.

Para ser possível todo este cenário, é fundamental uma estrutura de redes que suporte vários serviços, com segurança e eficiência. Nosso trabalho se ocupa de uma pequena parte dessa rede, qual seja, a de especificar a camada da rede que faz interface entre a camada física (meios de transmissão) e a camada lógica (tratamento e processamento das informações).

Neste trabalho procuraremos registrar o conhecimento acumulado durante o nosso curso de pós-graduação na FEE da *Uni*versidade Estadual de *Camp*inas (UNICAMP), e a nossa "tese" de mestrado no DT dessa Faculdade. Enfocaremos aqui duas vertentes que julgamos mais importantes: as técnicas de Redes de Comunicação e as técnicas para Projetos Eletrônicos, buscando contribuir apresentando nosso trabalho como uma experiência nessas áreas.

Neste capítulo 1 fazemos uma análise da importância das redes de comunicação nas relações sociais da humanidade no presente e no futuro próximo, enfatizando as promessas que as novas tecnologias fazem para as próximas décadas. Procuramos abordar o tema informalmente, tornando o capítulo acessível ao leitor de qualquer formação. Além disso, mostramos a organização desta tese, visando facilitar a sua leitura e acompanhamento.

No capítulo 2 descrevemos os principais conceitos envolvidos com Redes de Comunicação. Enumeramos as razões que levam à existência de padronização internacional para as Redes de Comunicação; o modelo de re-

ferência *O*pen *S*ystem *I*nterconnection (OSI) da International Organization for Standardization (ISO); os diversos serviços já padronizados; as redes de integração de serviços, os avanços tecnológicos esperados e as tendências futuras. Esse capítulo será de muita valia ao leitor interessado em conceitos básicos de Redes de Comunicação.

No capítulo 3 descrevemos o projeto RALFO dando especial atenção à sub-camada MAC. Faremos uma introdução ao projeto: os serviços propostos, a topologia adotada, as características dos nós e a descrição das suas camadas. Aqui detalhamos os serviços prestados pela MAC da RALFO, o método de acesso ao meio e o quadro da MAC. Fizemos a descrição funcional da sub-camada MAC especificando sua interface com a camada Física e com os seus usuários, as camadas: LLC/dados, LLC/voz e *Apl*icação (APL). Este capítulo é importante ao leitor interessado no entendimento da MAC da RALFO, assim como ao leitor do capítulo 5.

No capítulo 4 expomos as técnicas de automação de projetos eletrônicos (*E*lectronic *D*esign *A*utomation (EDA)) utilizadas. Introduzimos ao leitor nos diversos conceitos envolvidos na Engenharia de Projetos de Circuitos. Nesse capítulo não pretendemos analisar essa área com profundidade, mas apenas registrar a nossa experiência, que poderá ser aproveitada por aqueles interessados em trabalhar com o projeto de circuitos e em especial com as ferramentas da MGC, exemplificando os diversos recursos existentes no sistema. Nesse capítulo supomos que o leitor tem familiaridade como a Eletrônica Digital.

No capítulo 5 detalhamos, com desenhos esquemáticos ou com Linguagens de Descrição Formal, toda a MAC, apresentando o esquema de cada circuito, ou sua descrição funcional; sua descrição simbólica e seu comportamento, i.e., os resultados dos testes dos circuitos. Trata-se de um capítulo extenso e que deve ser lido pelo leitor que precise entender detalhadamente nosso projeto.

No capítulo 6 analisamos criticamente o nosso projeto e propomos algumas continuações ao projeto RALFO.

A redação deste trabalho não pretende ser exaustiva nos assuntos abordados, exceto no que diz respeito à especificação da MAC da RALFO, portanto no transcorrer dos capítulos sugerimos leituras complementares ao leitor interessado.

Encerramos este capítulo lembrando que o cenário que traçamos, traz várias questões que merecem nossa atenção. Certamente influenciará significativaamente nos costumes e comportamentos das pessoas, principalmente nos aspectos legais e de liberdade. As questões ligadas à segurança das informações e à privacidade das pessoas devem ser tratadas com cuidado, assim como as questões ligadas aos direitos autorais e formalização das regras de trabalho. Também é importante buscar respostas às questões

ligadas aos aspectos humanos, sociais e de criatividade que podem sofrer fortes influências deste processo.

Capítulo 2

Redes

No mundo atual a disseminação da informação é imprescindível para o seu funcionamento, como o fluxo de sangue é imprescindível para o funcionamento do nosso organismo. A palavra comunicação, que significa a troca de informação, possui um sentido muito mais amplo do que o utilizado neste trabalho. O cantar de uma andorinha, a conversa das baleias, os *flashs* para sinalização entre embarcações, a troca de correspondências, a transmissão de um evento esportivo por rádio ou TV, uma chamada telefônica e o acesso remoto a um serviço de um computador são exemplos de comunicação, mas neste trabalho nos ateremos às comunicaçõs efetuadas via meios eletro-magnéticos e fotônicos, mais especificamente às telecomunicações (o prefixo grego *tele* significa distante ([Cla91])), em especial à telefonia (transmissão de voz à distância) e às comunicações entre computadores, chamadas de redes de computadores.

Quando o objetivo deixa de ser apenas a comunicação entre dois pontos e passa a ser importante a possibilidade de troca de informações entre um conjuto de pontos, necessitamos de uma rede. Este será o objetivo da próxima seção, quando citaremos alguns tipos de redes de telecomunicação dando especial ênfase à telefonia e às redes de computadores. Na seção 2.2, falaremos sobre as necessidades de padronização, os orgãos internacionais de padronização, o modelo de referência para sistemas abertos da ISO. Alguns exemplos de serviços e protocolos de rede de computadores abertas. Na seção 2.4 falaremos sobre as motivações que levam às redes de serviços integrados, dando especial ênfase à *R*ede *D*igital de *S*erviços *I*ntegrados (RDSI), em padronização pelo *C*omité *C*onsultatif *I*nternational de *T*élégraphique et *T*éléphonique (CCITT).

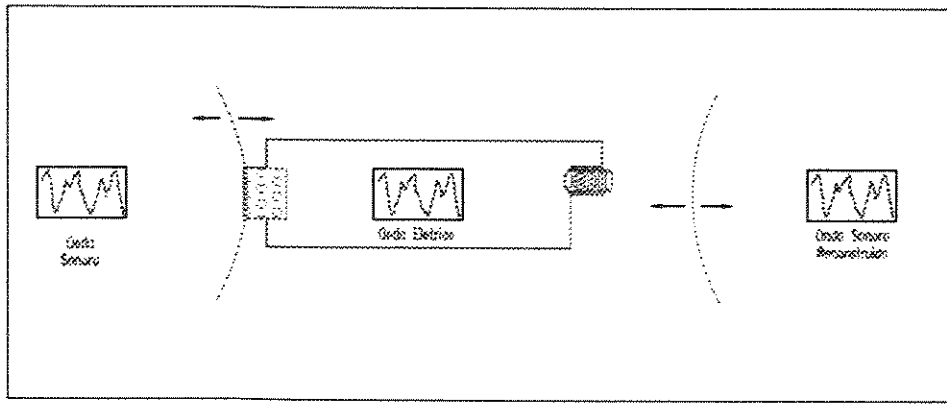


Figura 2.1: Circuito Telefônico Simplificado

2.1 Tipos de Redes

2.1.1 De Telefonia

A telefonia, comunicação de voz à distância ([Cla91]), inventada por Alexander Graham Bell em 1876 ([Bel82]) tornou-se a rede de comunicação mais popular do planeta, devido à sua simplicidade e conseqüente baixo custo. Em seu esquema original ainda utilizado até hoje e simplificado na Fig. 2.1 ([Cla91]), utiliza-se um microfone que transforma ondas de pressão provocadas pela voz no ar em ondas elétricas, que, atingindo o autofalante no outro extremo da conversação, permite a reprodução do som original no destino.

Utilizado apenas para comunicação entre dois pontos, o engenho de Graham Bell não teria grande utilidade prática. Foi então que, em 1878, na cidade norte-americana de Connecticut/New Haven, foi instalada a primeira central de comutação ([Bel82]), que permitiria a Conversação entre usuários utilizando quaisquer dois aparelhos ligados à central. Iniciava-se ali a rede telefônica.

As tecnologias envolvidas na comutação telefônica sofreram grande evolução. Inicialmente utilizou-se centrais de comutação eletro-mecânica de dois tipos: passo-a-passo ou *crossbar*. Com o advento da eletrônica digital integrada, que proporciona menores custos e maior confiabilidade aos circuitos eletrônicos, migrou-se para as centrais digitais, i.e., com controle interno digital e ligações digitais entre centrais digitais. Há uma tendência de digitalização até ao nível de usuário, como veremos na seção 2.4. A multiplicação dos números de terminais telefônicos levou à necessidade de

interligação de centrais via outras centrais, numa arquitetura hierarquizada, com *links*, entre elas, de altas taxas, permitindo o encaminhamento de várias chamadas por um único *link*. Uma descrição mais detalhada destas técnicas pode ser encontrada em [Bel82].

O controle sobre a comutação também sofreu sua evolução. No início totalmente manual, operado por telefonistas, hoje está totalmente automatizado.

A junção de duas tecnologias de comunicação, a telefonia e o radio, permitiu, em meados da década de 60 o aparecimento do *M*obile *T*elephone *S*ystem (MTS) ([Yac93]), que proporciona ao usuário um terminal telefônico, chamado *M*obile *S*tation (MS), ligado à central via radio. Porém, face o limite do espectro eletromagnético, esta tecnologia mostra-se insuficiente para suprir a demanda existente nos centros urbanos. Foi então que surgiu a telefonia celular, esquematizada na Fig. 2.2 ([Yac93]), que tem como princípio a sub-divisão de uma região em células hexagonais, sendo possível em cada célula a utilização de um sub-conjunto dos canais disponíveis, de tal forma que células vizinhas nunca utilizam os mesmos canais, evitando assim interferências. Cada célula possui uma *B*ase *S*tation (BS) que se comunica com as MSs via rádio e com o *M*obile *S*witching *C*enter (MSC) via cabos. O MSC é o responsável pela monitoração da posição dos MS ativos, chamada de *locating*, para determinação de qual BS será utilizada a cada instante. Portanto controlará as mudanças de canais (*hand-off* e *hand-over*) necessárias quando uma MS ativa locomover-se de uma célula para outra. Cabem também à MSC as funções comuns às centrais telefônicas, inclusive a interligação com a rede telefônica tradicional. Uma descrição completa da telefonia celular pode ser encontrada em [Yac93].

Dentre as características da telefonia destacamos a estreita largura de faixa utilizada: de 300 Hz até 3.4 k Hz ([Cla91]), a baixa *Q*uality *o*f *S*ervice (QOS) e aceitação de perda de até 1% da informação, segundo [Gon83]. A codificação da voz em *P*ulse *C*ode *M*odulation (PCM) ([Cla91]) feita com 8bits por amostra, a uma taxa de 8 k Hz, imposta pelo teorema de Nyquist ([Lat79]), leva-nos a uma largura de faixa 64 kbits/s; esta taxa não é alta, em comparação com as utilizadas nas *L*ocal *A*rea *N*etworks (LANs) tradicionais, mas é necessário que tal canal esteja disponível durante a conversação. Dois parâmetros cujos valores são baseados em conceitos subjetivos: o QOS e a taxa de perda de amostras de voz. Quando nos referimos a QOS estamos querendo referir especificamente o nível de erro da rede, ou o *B*it *E*rro *R*ate (BER). Lembramos que o termo QOS tem uma definição muito mais abrangente, estabelecida pelo CCITT no [CCI89i] e descrita de uma forma clara em [Cla91]. A taxa de perdas, citada por Gonsalves, permitirá a algumas redes telefônicas digitais o descarte de

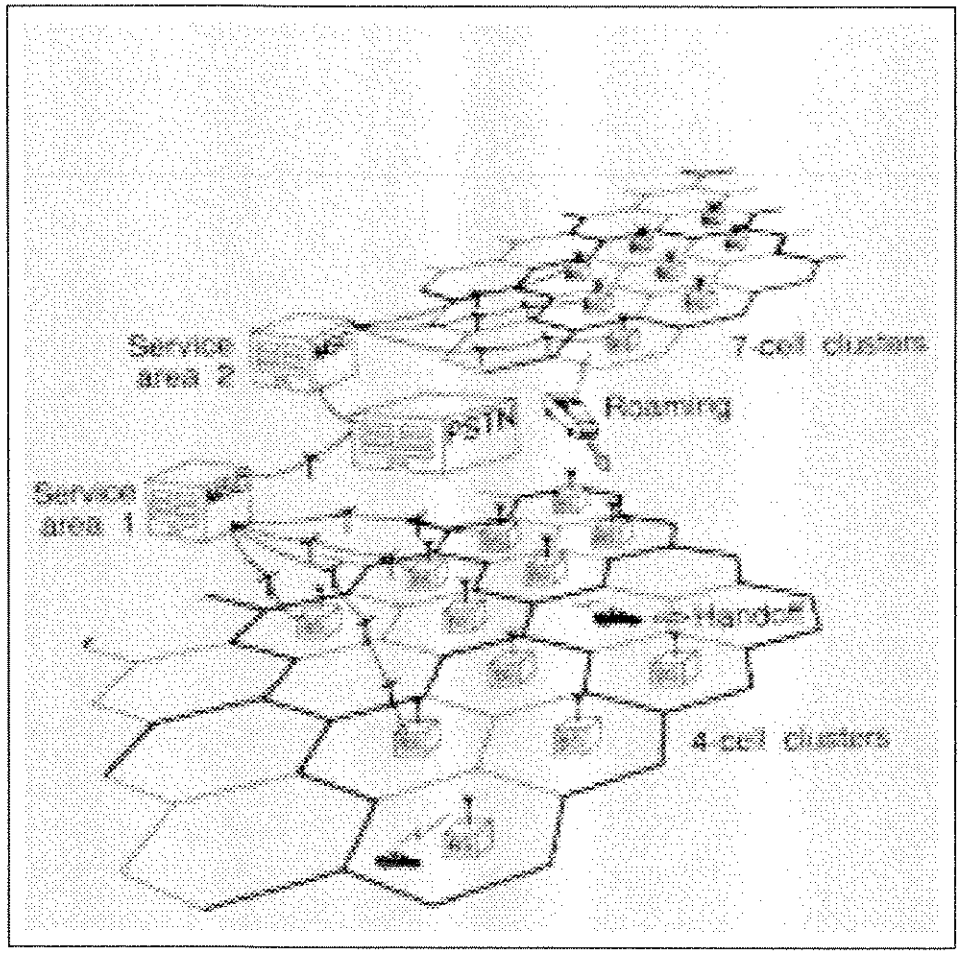


Figura 2.2: Telefonia Celular

algumas amostras de voz quando o sistema estiver congestionado, sem conseqüências desastrosas na qualidade de serviço.

2.1.2 De Computadores

As redes de computadores, evolução natural das redes *front-end* que interligam terminais a computadores de grande porte, propiciam a interligação entre computadores motivada por diversos fatores e proporcionando diversos serviços aos usuários.

Andrew S. Tanenbaum destaca os seguintes fatores como motivadores para o uso de redes de computadores ([Tan88]): o compartilhamento de recursos, a confiabilidade, a economia, a disponibilidade de um meio de comunicação. Todos os programas, dados e equipamentos estão disponíveis para qualquer usuário independentemente da distância física que os separe, terminando assim com a "tirania geográfica". A redundância de recursos interligados via rede pode formar um sistema mais confiável, permitindo a utilização de recursos extras na eventual falha de algum elemento de um sistema. Outro fator importante é a economia, propiciada pelo uso de sistemas de médio e pequeno porte em substituição aos caros *main-frames*. O termo *downsizing* expressa a tendência atual de substituição de sistemas de grande porte por redes de estações de trabalho e/ou micro-computadores. As redes de computadores representam também um poderoso e versátil meio de comunicação.

Costuma-se classificar as redes de computadores em função do seu tamanho em ([Men]):

LAN: que interliga computadores em geral numa única sala ou num único prédio. Tipicamente de alguns metros até 5 km;

***M*etropolitan *A*rea *N*etwork (MAN):** que interliga sistemas numa região metropolitana. Tipicamente de 5 a 50 km;

***W*ide *A*rea *N*etwork (WAN):** para interligação de sistemas a grandes distâncias, podendo chegar a nível mundial.

Associam-se os parâmetros de desempenhos melhores para as redes menores, como largura de faixa e BER, muito embora haja uma tendência de termos MAN ou mesmo WANs com a qualidade das atuais LANs como veremos ainda neste capítulo.

As redes oferecem aos usuários uma gama nova de aplicações tais como o acesso a programas remotos; acesso a base de dados remotas; publicações eletrônicas como jornais, revistas, foruns de discussões, artigos técnicos; correio eletrônico; fax; aquisição de programas, imagens, músicas; jogos

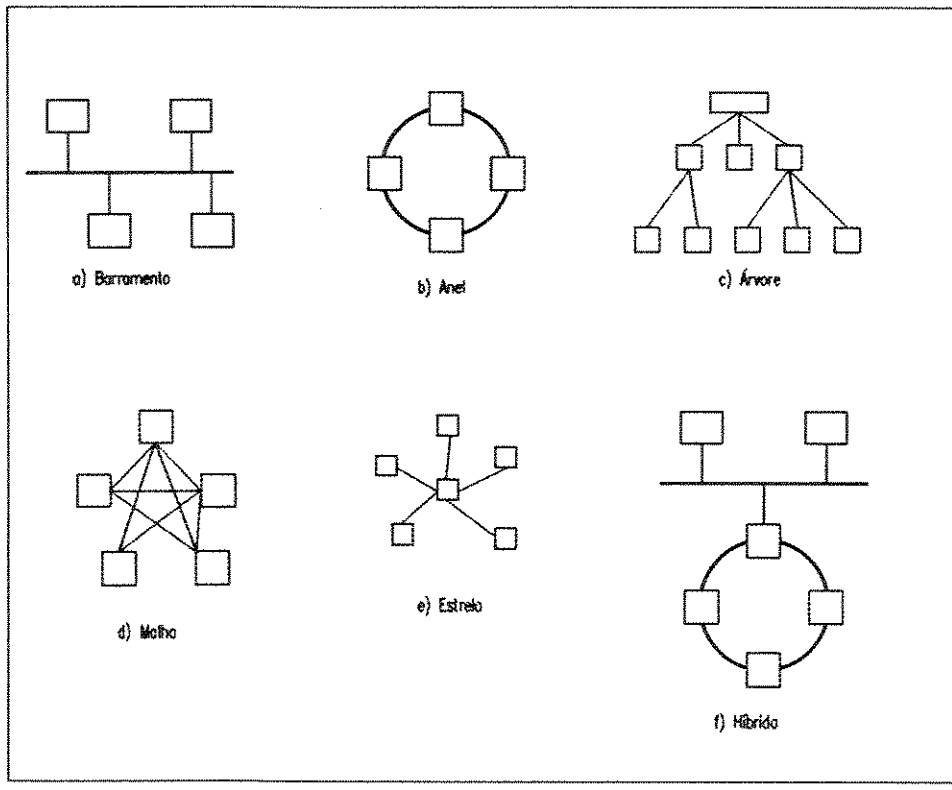


Figura 2.3: Topologias de Rede

compartilhados por usuários espalhados pelo mundo. Alguns serviços ora recentes ou em desenvolvimento necessitam de larga faixa. Destacamos alguns exemplos fornecidos pelo CCITT: tele-educação, tele-compras, tele-*marketing*, segurança de prédios, monitoração de tráfego, transmissão de programas em várias línguas simultaneamente, transmissão de vários programas simultâneos, controle em tempo real, telemetria, alarmes, transmissão de texto, imagens e desenhos em alta velocidade, imagens médicas, imagens profissionais, distribuição de programas de *Televisão* (TV).

O modelo cliente/servidor vem sendo usado em larga escala nos diversos tipos de rede a nível de aplicação. Temos um processo que fica aguardando a conexão de um ou vários processos: o servidor, e do outro lado os clientes, que tomam a iniciativa de requerer um determinado serviço ao servidor. Por estes processos, cliente e servidor trocam informações entre si via um

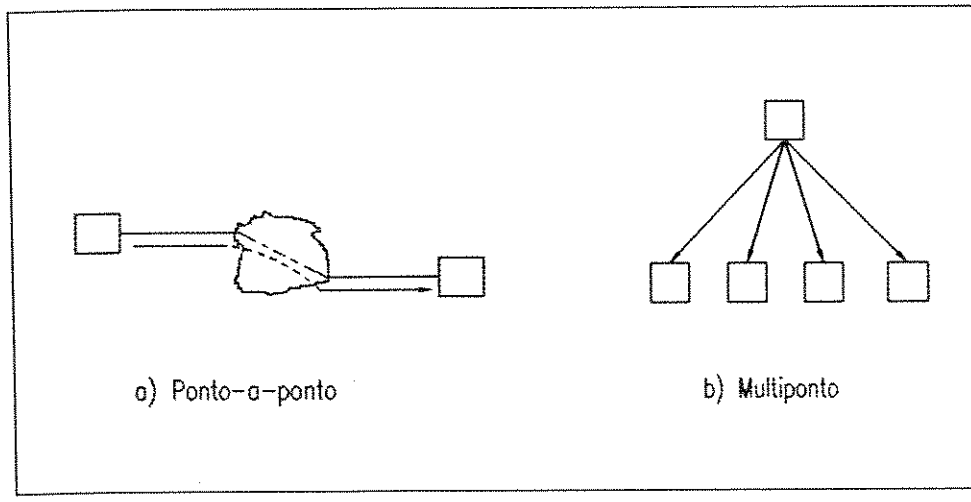


Figura 2.4: Formas de Transmissão

pré-estabelecido protocolo.

Classificam-se também as redes de computadores ([Men]) em função da forma pela qual estão interligadas, i.e., a topologia. Destacamos as seguintes topologias: barramento, anel, árvore, estrela, malha e híbrida (Fig. 2.3), e também em função da forma de transmissão: ponto-a-ponto ou multiponto (*broadcast*) (Fig. 2.4).

2.1.3 Via Radiodifusão

A transmissão de informações via ondas eletromagnéticas pelo espaço, cuja possibilidade foi provada no final do século passado pelo alemão H. G. Hertz ([Yac93]), é de fundamental importância no mundo das comunicações. A modulação é a técnica utilizada para transmissão de sinais via radio por duas razões: (1) para que o alcance do sinal seja maior dada uma determinada potências do transmissor e (2) para permitir a emissão simultânea de diversas infomações no mesmo espaço físico. Há diversas técnicas de modulação, cujas descrições podem ser encontradas em [Lat79].

Muito embora o termo *rádio* nos lembre as rádios *A*mplitude *M*odulada (AM)s e *F*reqüência *M*odulada (FM)s comerciais, onde nós temos a informação sendo transmitida de uma forma ponto-multiponto (*broadcast*), a transmissão de informações pelo ar a partir da antena de um transmissor, a radiodifusão, é utilizada em diversas outras ocasiões e não somente para transmissão de audio. Michel Ycoub cita em [Yac93] algumas outras formas

de redes por radio-difusão.

O serviço de *beep*, como é conhecido no Brasil, propicia ao usuário saber, via um *beep* do seu rádio receptor portátil, que há algum recado para ele numa central. O usuário deverá então ligar para a central para receber o recado. Os serviços de *beep* mais avançados permitem a emissão de uma mensagem alfa-numérica para o rádio receptor do usuário, desobrigando-o assim de ter que telefonar para a central.

Redes *P*acket *R*adio (PR) são a adptação da tecnologia de comutação utilizando-se de rádios móveis. A intenção é propiciar a usuários em grandes regiões a comunicação de dados. Utiliza-se PR quando a conexão física entre os usuários é difícil ou cara. Nesta rede a topologia é modificada dinamicamente, pois, além dos receptores/transmissores poderem ser móveis — podemos ter um nó em um navio, por exemplo — as condiçõe ambientais podem impedir a conexão entre um determinado conjunto de rádios numa dada região. Neste caso torna-se necessária a utilização de técnicas *store-and-forward* e/ou roteamento dinâmico, onde um determinado nó funciona como intermediário para a transmissão de um pacote para outro nó da rede.

Os *F*uture *P*ublic *L*and *M*obile *T*elecommunication *S*ystems (FPLMTS) é um rede por radio-difusão emergente que está sendo padronizada a nível de *C*omité *C*onsultative *I*nternational de *R*adio (CCIR) com a intenção de propiciar serviços fixos para paises em desenvolvimento com dificuldade para implementação de telefonia rural ou para paises com lugares de difícil acesso, ou com grande território, ou com a população geograficamente distribuida. O uso de radio propicia flexibilidade, modularidade, capacidade de cobrir grandes áreas e baixos custos. Além da transmissão de voz são previstos também serviços ponto-multiponto, envio de mensagens curtas, *beep*, facsimile, texto e dados.

Telefone sem fio é outra rede de comunicação por radio-difusão. Estamos hoje com a terceira geração de telefones sem fio: *C*ordless *T*elephone (CT)3 nos *E*stados *U*nidos da *A*mérica (EUA) e *D*igital *E*uropean *C*ordless *T*elephone (DECT). É possível utilizá-los não apenas para uso doméstico, como também para *P*rivate *A*utomatic *B*ranch E*x*change (PABX) e telepoint, i.e., onde o usuário pode fazer chamadas telefônicas, mas não pode recebê-las. Maiores detalhes sobre as diferenças entre as três gerações de telefones sem fio podem ser encontrados em [Yac93].

A emergente *P*ersonal *C*ommunication *N*etwork (PCN) tem como objetivo prover o uso de telefones a todo tempo e em todo lugar para o mercado de massa. A idéia é manter a arquitetura da telefonia celular diminuindo o tamanho do raio das células para entre 1 km (centros urbanos) a 6 km (meio rural), e propiciando ao usuário um serviço de baixo custo e alta qualidade. Na telefonia celular o valor típico para o raio da célula

varia de 2 a 16 km hoje.

Comunicação móvel por satélite é utilizada para comunicação entre estações terestres via um ou mais satélites ou entre estações terrestres móveis. As *estações terrestres móveis* podem estar localizadas em embarcações marítimas, aeronaves ou veículos terrestres, como trens. Algumas organizações internacionais provêem sistemas de telecomunicação via satélite. Entre outras destacamos a *Inte*rnational *Tele*communication *Sat*ellite Organization (INTELSAT), a *Eu*ropean *Tele*communication *Sat*ellite Organization (EUTELSAT) e a *I*nternational *Mar*itime *Sat*tellite Organization (INMARSAT).

Um dos mais populares meios de comunicação, a televisão pode funcionar via radio-difusão, emitindo para o receptor não apenas um canal de voz, assim como imagem. Na transmissão da imagem os sistemas de televisão costumam enviar a imagem monocromática, acessível por monitores *preto-e-branco* mais a informação de cor. Uma descrição detalhada sobre o sistema PAL-M de televisão, adotado no Brasil, pode ser encontrada em [SS87]. Além do sistema de difusão *broadcast* onde cada usuário pode captar a programação de um ou diversos canais de TV, bastando possuir um aparelho receptor, a televisão por assinatura tem sido bastante difundida. Nela o assinante paga uma taxa mensal para ter acesso a um determinado número de canais. Pode ser transmitida tanto via cabo quanto via rádio, sendo neste último caso necessária a presença de um decifrador fornecido pelo prestador do serviço.

Como já dissemos, os sinais de voz, vídeo, audio de alta fidelidade, dados, etc. são modulados para serem transmitidos em alguma freqüência do espectro eletromagnético. Como o espectro é, na prática, limitado, e como as ondas eletromagnéticas não respeitam fronteiras políticas, é necessário uma certa cooperação a nível mundial evitando o caos no setor. A nível mundial a *I*nternational *T*elecomunication *U*nion (ITU) dividiu o globo terrestre em três grandes regiões, que têm autonomia, a nível das conferências internas regionais da própria ITU, para definir a utilização do espectro eletromagnético. No Brasil a administração do uso do espectro é feita pelo *De*partamento *N*acional de *Tele*comunições (DENTEL), nos EUA pelo *F*ederal *C*ommunications *C*ommission (FCC) para orgãos não governamentais e pelo *I*nterdepartment *R*adio *A*divisory *C*ommittee (IRAC) para orgãos governamentais.

2.1.4 De Automação Industrial

A automação industrial é uma área tecnológica emergente de grande importância para os paises industrializados. Alta qualidade, baixos custos e versatilidade são características impressidíveis nos mercados abertos.

A integração da manufatura via computador (*C*omputer *I*ntegrated *M*anufacturing (CIM)) passou a ser a chave da questão, onde é necessária a existência de diversos tipos de redes com características distintas.

A hierarquia de uma fábrica pode ser composta por vários níveis ([GN86, Men90, Men89]):

Nível de atuadores/sensores: Com inteligência local para controlar um determinado componente, e.g., uma válvula, um forno, veículos, robôs. Implementado por *C*ontroladores *L*ógicos *P*rogramáveis (CLPs), *C*omandos *N*uméricos *C*omputadorizados (CNCs);

Nível de processo: Controla um conjunto de unidades, um sub-sistema;

Nível de sistemas: Geração de programas de produção reais para a operação local. Trabalha a nível de célula de manufatura;

Nível de áreas: Busca utilizar eficientemente os recurso, minimizar os custos, maximizar a produção e qualidade e minimizar os atrasos;

Nível empresarial: É o nível de tomada de decisões.

Cada nível possui requisistos próprios no que concerne à ligação entre os equipamentos.

Entre o nível de atuadores/sensores, o chamado chão de fábrica, e o nível de processo temos a comunicação em taxas da ordem de 10 kbits/s, a distâncias de até 100 m. A este nível temos uma grande diversidade de equipamentos simples e especializados, que são interligados aos computadores supervisores, a nível de processo, via ligações ponto-a-ponto. Alguns padrões *E*lectrical *I*ndustries *A*ssociation (EIA)/CCITT são largamente utilizados. Para distâncias da ordem de 1 km utiliza-se o RS-232C com *loop de corrente*. O RS-422 tabalha com dois pares trançados, para ligações *full duplex* da ordem de 100 m e com taxas maiores que 1 M *B*its *p*or *S*egundo (bps) ([Hal88]). Em 1986 o *I*nternational *E*lectrotechnical *C*ommission (IEC) lançou os requisitos funcionais do projeto *Field-bus* ([Men90]) que tem como meta a padronização universal das interfaces dos equipamentos de campo, propiciando a ligação a estes equipamentos via um barramento, trazendo assim alguns benefícios, tais como redução dos custos com cabeamento e interfaces, facilidade de instalação e manutenção, separação da instrumentação do controle, fácil e rápida expansão, deteção automática de falhas no cabo, desempenho maior do que usando-se ligações analógicas, disponibilidade imediata de informações sobre qualquer local do sistema, possibilidade de deslocar o processamento dos sinais para o campo.

No nível de processo e no nível de sistema, temos a necessidade de garantir a comunicação em tempo real, assegurando uma determinada freqüencia de varredura dos equipamentos de controle pelos de supervisão. Visando aplicações como as *C*élulas *F*lexíveis da *M*anufatura (CFMs) e *S*istemas *D*igitais de *C*ontrole *D*istribuído (SDCDs) temos uma proposta de padronização da IEC: o *Pro*cess Data High*way* (PROWAY) ([Men90]), que tem como requisitos a resposta rápida da ordem de 30 ms, para mensagens curtas de alta prioridade, alta confiabilidade do meio e dos métodos de codificação—mesmo em ambientes hostis— facilidade de atibuição de prioridades dinamicamente, conexão com as redes principais da fábrica via pontes/roteadores e suporte à redundância de componentes de rede.

Os três níveis mais baixos da hierarquia de uma fábrica estão esquematizados nas Figs. 2.5 e 2.6, através de um exemplo de refinaria de petróleo. Na Fig. 2.5 temos neste exemplo ligações ponto-a-ponto, e.g., RS-422 entre os controladores e o computador supervisor e temos esquematicamente os sensores de temperatura, pressão, análise química e fluxo ligados, suponhamos, via um barramento *Field-bus* ao supervisor. Neste esquema a torre de *crakeamento* comporta-se como um célula de *manufatura*, tendo seus diversos processos controlados por um supervisor. Na Fig. 2.6 temos um computador *scheduler* responsável pela geração de programas de produção reais para a operação local. Digamos que o barramento que interliga o *scheduler* com o os supervisores seja um PROWAY.

Os níveis mas altos de uma fábrica podem contar com as tecnologias já disponíveis de LANs e WANs, que serão vistas mais adiante na seção 2.2. A toda esta hieraquia integrada em uma fábrica chamamos de CIM, que aponta como a tendência para as indústrias futuras.

2.2 Padronização

A padronização das redes de comunicação propicia a ligação entre diversos tipos de equipamento, de diversos fabricantes e, no caso de computadores, diversas arquiteturas e sistemas operacionais. A padronização propicia o fornecimento de produtos por diferentes fabricantes, permitindo uma maior competição que tende a elevar a qualidade e diminuir os preços. A fabricação em massa permite a economia em escala, a produção por exemplo, no caso de componentes eletrônicos, de circuitos *V*ery *L*arge *S*cale *I*ntegration (VLSI) que é outro fator para diminuição do preço e aumento da aceitação do produto. Quando nos referimos ao preço de um produto, no caso um sistema de comunicação, ou, mais particularmente uma rede de computadores, nos referimos também aos custos de manutenção e operação

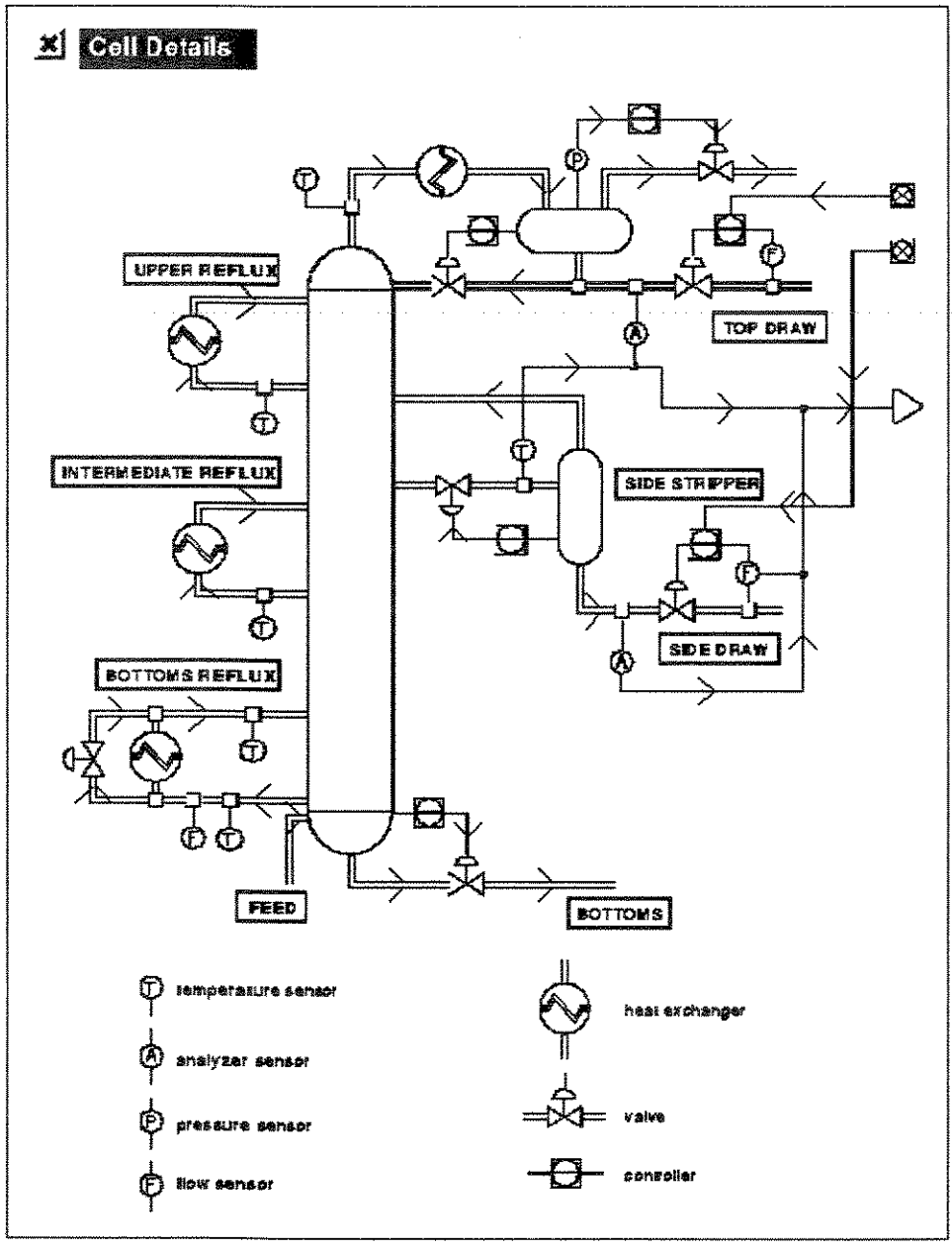


Figura 2.5: Célula de Manufatura

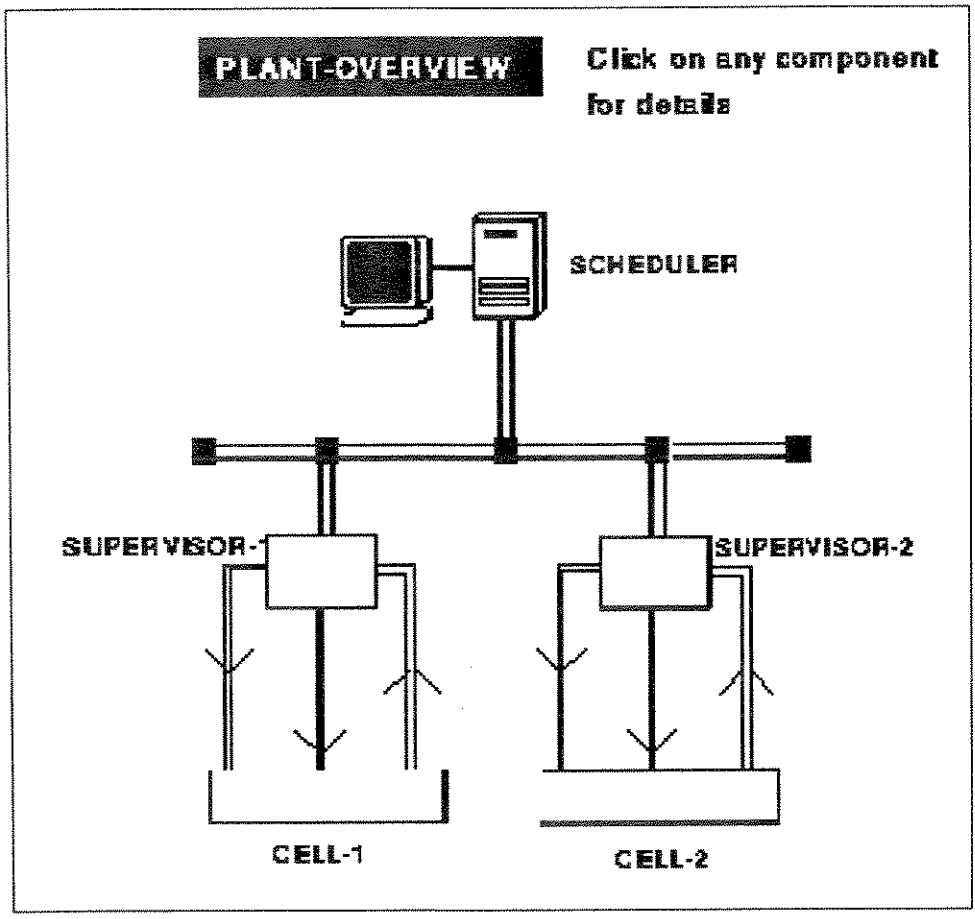


Figura 2.6: Planta de uma Fábrica

do sistema, que, sem dúvida, tendem a ser menores para os sistemas padronizados.

Além dos motivos já citados, equipamentos de comunicação são freqüentemente utilizados para intercâmbio de infomações a nível mundial, o que só é possível se houver alguma padronização.

A padronização pode ser *de facto* ou *de jure*. Um padrão de *facto* é estabelecido pelo mercado sem necessariamente a intervenção de orgãos de padronização. Os computadores pessoais *P*ersonal *C*omputers (PCs), produzidos inicialmente pela *I*nternational *B*usiness *M*achine (IBM) e hoje feitos por dezenas de fabricantes diferentes, é um padrão *de facto* e o mesmo podemos dizer com relação ao uso do sistema operacional Unix[1] nos centros de pesquisas e universidades do mundo. Os padrões *de jure* são estabelecidos e especificados formalmente por organizações internacionais ou nacionais com autoridade para tal. De fato o que importa para o mercado são os padrões *de facto*, sendo este o objetivo de cada padrão *de jure* ([Tan88]).

2.2.1 Quem é Quem no Mundo da Padronização

Em 1932 com a junção da *International Telegraph Union* (1865) e da *Radio Telegraph Union* (1903) foi criada a ITU, a agência especializada da *O*rganização das *N*ações *U*nidas (ONU) em telecomunicações, com o objetivo de harmonizar e cuidar das telecomunicações no mundo. A estrutura geral da ITU, esquematizada na Fig. 2.7, possui 6 orgãos:

Conferência Plenipotenciária — Re-examina quinqüenalmente as convenções, determina a política geral, o orçamento da ITU, os salários e elege os membros dos outros orgãos;

Conselho Administrativo — Composto por 36 membros que são responsáveis por dirigir os trabalhos da ITU nos períodos entre as conferências plenipotenciárias, que em geral reunem-se anualmente;

CCITT — Elabora recomendações técnicas sobre telefonia, telegrafia e interfaces de comunicação de dados;

CCIR — Elabora recomendações técnicas sobre comunicação via rádio;

IFRB — Analisa e registra a alocação das faixas de freqüência nos paises;

[1] Unix é marca registrada do Unix Laboratory.

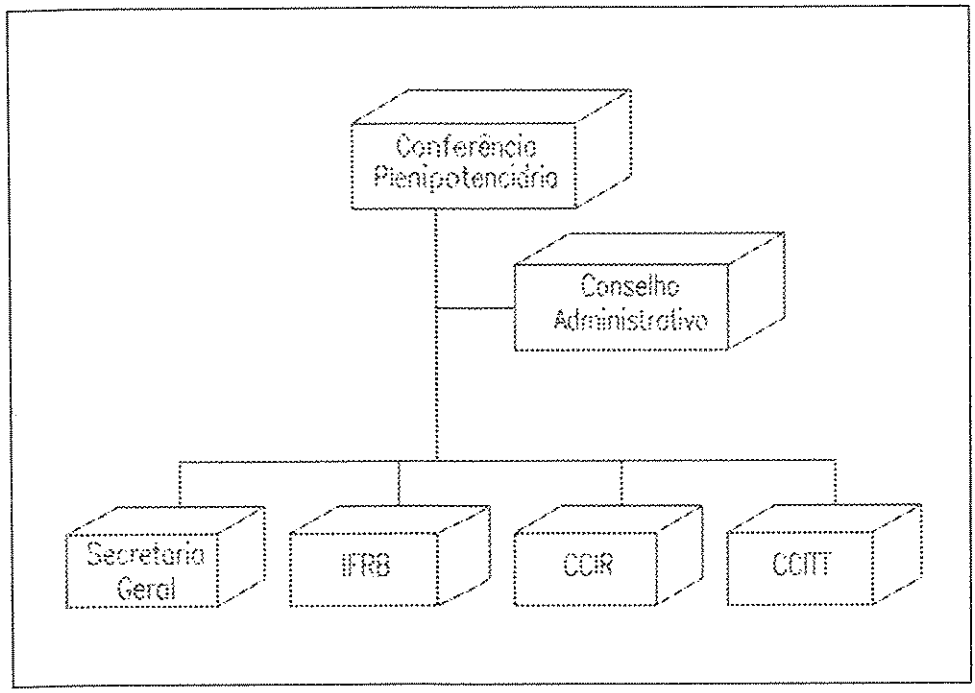


Figura 2.7: Estrutura da ITU

Secretaria Geral Responsável pela gerência executiva e cooperação técnica. Cuida da tradução, interpretação e publicação dos documentos da ITU.

Os membros do CCITT são divididos em cinco classes:

A: são as *P*osts, *T*elegraphs & *T*elephones (PTTs), empresas estatais que monopolizam o setor em determinado pais, por exemplo a Telebrás no Brasil;

B: organizações privadas que reconhecidamente prestam serviços no setor, por exemplo a *A*merican *T*elegraph *and T*elephone (AT&T) nos EUA;

C: organizações científicas e industriais com interesses diretos no CCITT;

D: outras organizações internacionais de padronização, por exemplo a ISO;

E: outras organizações com interesses secundários no trabalho do CCITT;

onde apenas os membros da classe *A* têm direito a voto ([Tan88, Yac93]).

A ISO, é uma organização internacional fundada em 1946 que visa estabelecer padrões internacionais para os mais variados fins. Os seus membros são as organizações nacionais dos 89 paises membros, entre outros temos a *A*merican *N*ational *S*tandard *I*nstitute (ANSI), a britânica BSI, a francesa AFNOR, a alemã DIN e a brasileira BRISA. A ISO é formada por cerca de 200 comitês técnicos, numerados por ordem de criação. O *T*echnical *C*ommittee (TC)97 trabalha com computadores e processamento de dados. Cada TC é subdividido em *S*ub*C*ommittees (SCs), que por sua vez são subdivididos em *W*ork*G*roups (WGs), onde os trabalhos são realizados por cerca de 100.000 volutários espalhados pelo mundo.

O procedimento para estabelecimento de um padrão pela ISO, assim como pelo CCITT, busca chegar o mais próximo possível do consenso. O processo começa quando alguma organização nacional sente a necessidade de padronização em alguma área. Um WG é formado e elabora um *D*raft *P*roposal (DP), que circula por todos os outro membros, sendo criticado por seis meses. Se a maioria substancial aprovar, um documento revisado chamado *D*raft *I*nternational *S*tandard (DIS) é lançado em circulação para comentários e votos. Conforme o resultado dessa etapa é preparado um *I*nternational *S*tandard (IS), aprovado e publicado ([Tan88]).

Outro orgão de padronização de relevância internacional é o *I*nstitute of *E*lectrical and *E*lectronics *E*ngineers (IEEE), maior organização mundial de profissionais, que destaca-se pelo seu trabalho na padronização de LANs. O padrão IEEE802 foi adotado pela ISO como ISO 8802 ([Tan88]).

A EIA é uma associação cujo primeiro objetivo foi o estabelecimento de padrões para a camada física do OSI-*R*eference *M*odel (OSI-RM), e.g., a interface RS-232 ([Ste90]).

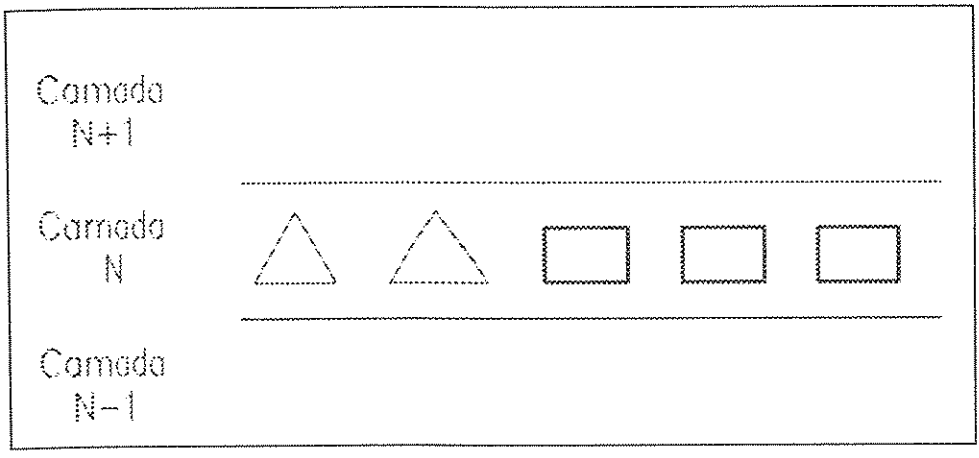


Figura 2.8: Entidades e Instâncias

A *I*nternet *A*ctivities *B*oard (IAB) é um pequeno grupo de pesquisadores que dirige a maior parte dos trabalhos envolvendo os protocolos *T*ransmission *C*ontrol *P*rotocol (TCP)/*I*nternet *P*rotocol (IP) ([Ste90]).

2.2.2 Modelo de Referência OSI

Em busca da padronização para redes de *sistemas abertos* foi proposto na ISO em 1979 o DP 7498, chamado OSI-RM. Tal modelo estabelece que uma rede aberta é formada por camadas sobrepostas, onde cada camada N executa determinada função, fornecendo serviços à camada superior, $N+1$, utilizando-se para isto dos serviços da camada inferior, $N-1$. O OSI-RM possui 7 camadas, especificando a função de cada uma. A especificação formal do OSI-RM pode ser encontrada no documento [ISO82] e uma descrição didática pode ser vista no livro [Tan88].

A estratificação em camadas do OSI-RM é feita de tal forma que uma determinada camada *enxerga* a camada inferior, sua provedora de serviços, como sendo a *rede*, sem tomar conhecimento do que ocorre nas camadas inferiores. No linguajar da ISO uma camada que utiliza dos serviços de uma camada imediatamente inferior, é chamada usuária desta camada. Para cada camada de uma rede aderente ao OSI-RM podemos ter uma ou mais implementações, chamadas entidades, daquela camada. Uma dada entidade de uma camada N pode ter várias instâncias, já que podemos ter a prestação simultânea de um mesmo serviço diversas vezes num determinado instante. A Fig. 2.8 esquematiza, num determinado instante, a existência

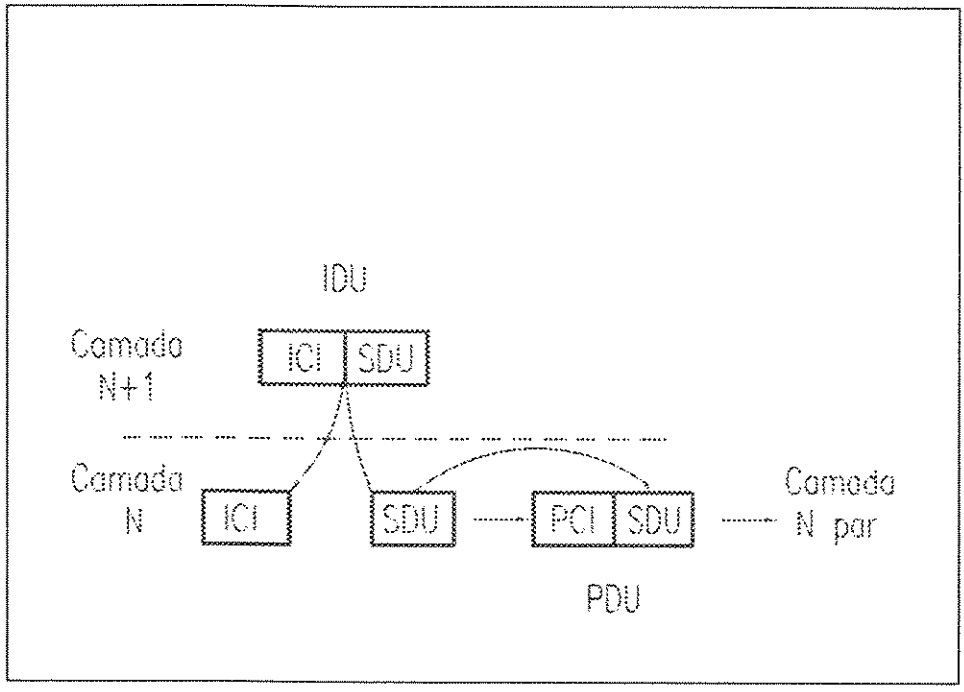


Figura 2.9: PDUs

de duas instâncias da entidade △ e de três instâncias da entidade □. Uma instância de uma entidade usuária N tem como objetivo *conversar* com uma instância de sua camada N-par, utilizando-se para isto de N-*P*rotocol *D*ata *U*nits (PDUs) (Fig. 2.9). A PDU encaminhada para a camada par, contém dados da camada usuária, chamados de *S*ervice *D*ata *U*nit (SDU) anexados à *P*rotocol *C*ontrol *I*nformation (PCI). A entidade da camada usuária encaminha seus dados, a SDU, para a entidade da provedora, anexando uma *I*nterchange *C*ontrol *I*nformation (ICI), formando a *I*nterchange *D*ata *U*nit (IDU). A entidade provedora retira a ICI, anexando ao SDU a PCI, formando o N-PDU. A comunicação entre entidades de camada adjacente é feita via um ou mais *S*ervice *A*ccess *P*oint (SAP), estabelecendo-se *C*onnection *E*nd *P*oints (CEPs) para identificar cada instância de cada entidade usuária. Uma discussão mais detalhada sobre SAPs e CEPs pode ser encontrada em [ISO82, TPP87].

As entidades de uma rede podem prover serviços com diversas características. Algumas características mais comuns são serviços com e sem conexão, com e sem validação, com e sem confirmação, com diversos tamanhos de mensagens, serviços *half-* ou *full-duplex*, e serviços confiáveis ou não. Podemos fazer uma analogia entre uma chamada telefônica convencional e serviços com conexão, assim como entre o serviço postal e serviços por datagrama. No primeiro caso há o estabelecimento de um canal, mesmo que virtual, para a troca de mensagens entre instâncias pares. No segundo temos que cada mensagem é transmitida em um envelope, i.e. datagrama, devidamente endereçado. Os serviços com validação são aqueles que, utilizando-se geralmente de técnicas para inserção de redundância na mensagem transmitida (informações sobre tais técnicas podem ser encontradas em [Hal88]), checam o conteúdo da mensagem recebida, verificando se confere com a emitida (com grande probabilidade de acerto). Os serviços com confirmação são aqueles onde o receptor da mensagem confirma o seu recebimento ao emissor. Os serviços que oferecem conexão contínua (*stream*), permitem à usuária enviar sua mensagem por um canal, eventualmente virtual, até a entidade par; outra opção é a utilização de datagramas, onde, ao contrário dos *streams* a mensagem tem tamanho limitado. O tamanho máximo de uma mensagem, uma SDU, quando existe, é chamado de *M*ax *T*ransmission *U*nit (MTU). Os serviços que proporcionam comunicação bi-direcional simultânea são chamados de *full-duplex*, os que proporcionam a comunicação bi-direcional não simultânea são os *half-duplex*, e os que proporcionam a comunicação unidirecional são os *simplex*; podemos fazer uma analogia dos serviços *full-duplex*'s com uma ligação telefônica, dos *half-duplex*'s com uma conversa entre dois radio-amadores e dos *simplex*'s com os programas das rádios comerciais. Serviços confiáveis são aqueles que possuem validação da

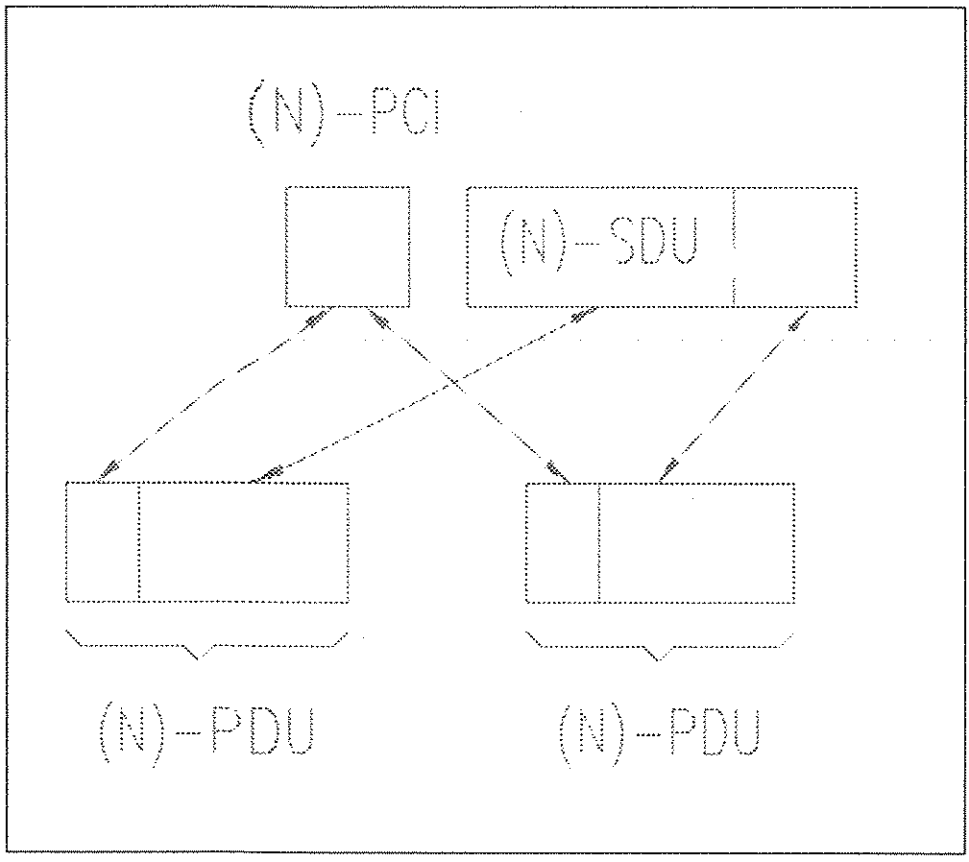


Figura 2.10: Segmentação e Remontagem

mensagem e fornecem a entidade usuária um canal contínuo (*stream*).

Uma entidade, ao fornecer um serviço com determinada característica, o faz podendo ou não utilizar a mesma característica da camada inferior. Por exemplo, uma entidade pode fornecer um serviço com conexão utilizando-se de um serviço sem conexão oferecido pela camada inferior.

O [ISO82] define também algumas técnicas para transmissão de dados. Quando a SDU, recebida da entidade usuária, em conjunto com a PCI cabe numa PDU é necessário apenas esta PDU para transmitir os dados para a entidade par (Fig. 2.9). Quando uma SDU em conjunto com a PCI são maiores do que a MTU é necessário que a entidade emitente faça a seguimentação (*segmenting*) e que a entidade par faça a remontagem (*reassembling*) da SDU (Fig. 2.10). Para tal é necessário que a PCI

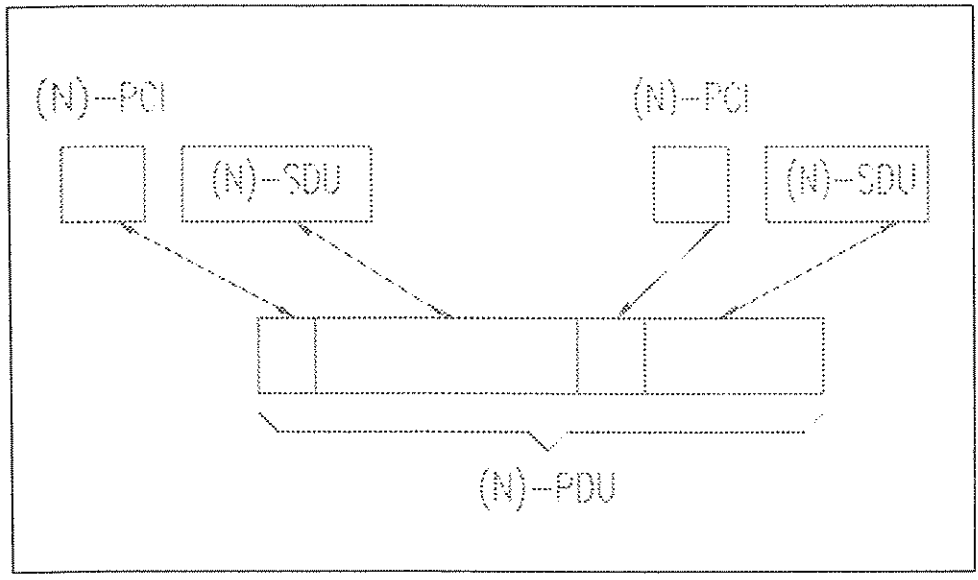


Figura 2.11: Empacotamento e Desempacotamento

contenha informações sobre à seqüencia das PDUs, já que pode haver o estravio de alguma PDU na trasmissão ou mesmo a ordem de chegada das PDUs pode não obedecer a seqüência correta das PDUs. Algumas técnicas para controle de seqüenciamento (*A*utomatic *R*epeat Re*q*uest (ARQ)) são descritas em [Hal88]. Outra técnica descrita em [ISO82] utilizada para melhor aproveitamento dos recursos da rede é o empacotamento (Fig. 2.11), onde uma PDU é utilizada para transmissão simultânea de diversas SDUs. Uma entidade encaminha suas PDUs via SDU da camada inferior. É possível também o envio de diversar PDUs numa única SDU, técnica esta chamada de concatenação (*concatenation*), sendo necessário que a entidade par faça a separação (*separation*). Assim como o empacotamento (*blocking*)/desempacotamento (*deblocking*), a concatenação/separação são utilizadas para otimizar os recursos da rede. A diferença entre os dois métodos é que o primeiro é feito pela camada provedora dos serviços e o segundo pela camada usuária.

O controle de fluxo é também utilizado entre entidades pares para aumentar ou diminuir o fluxo de informações quando se faz necessário. Há também o envio de mensagens urgentes, que devem ser enviadas com maior prioridade do que o normal.

É possível a utilização de mais de uma conexão da camada inferior

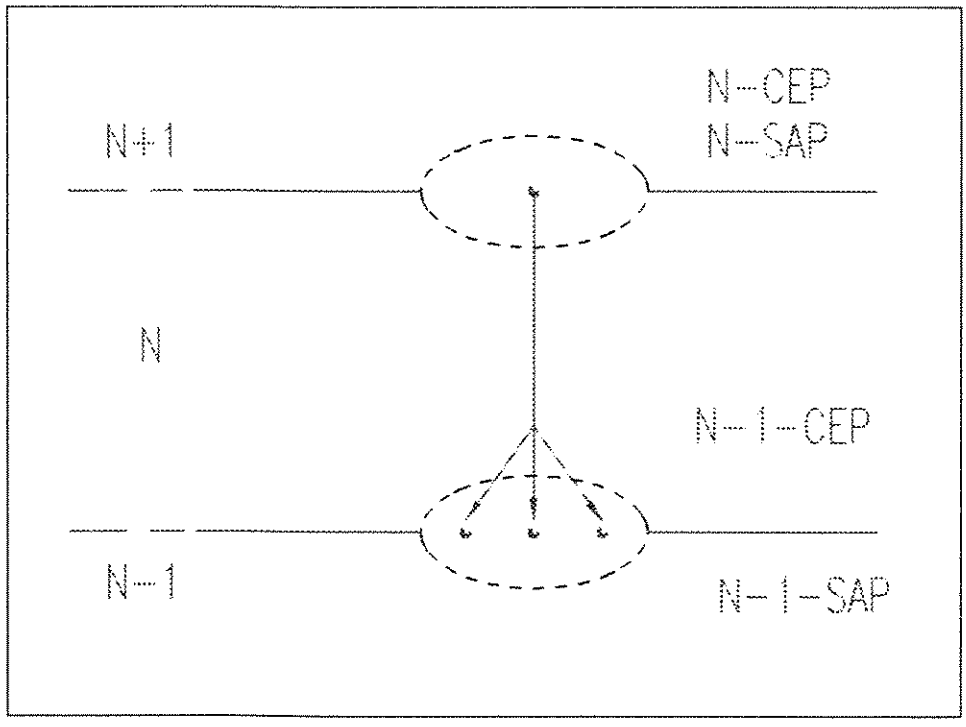


Figura 2.12: Splitting

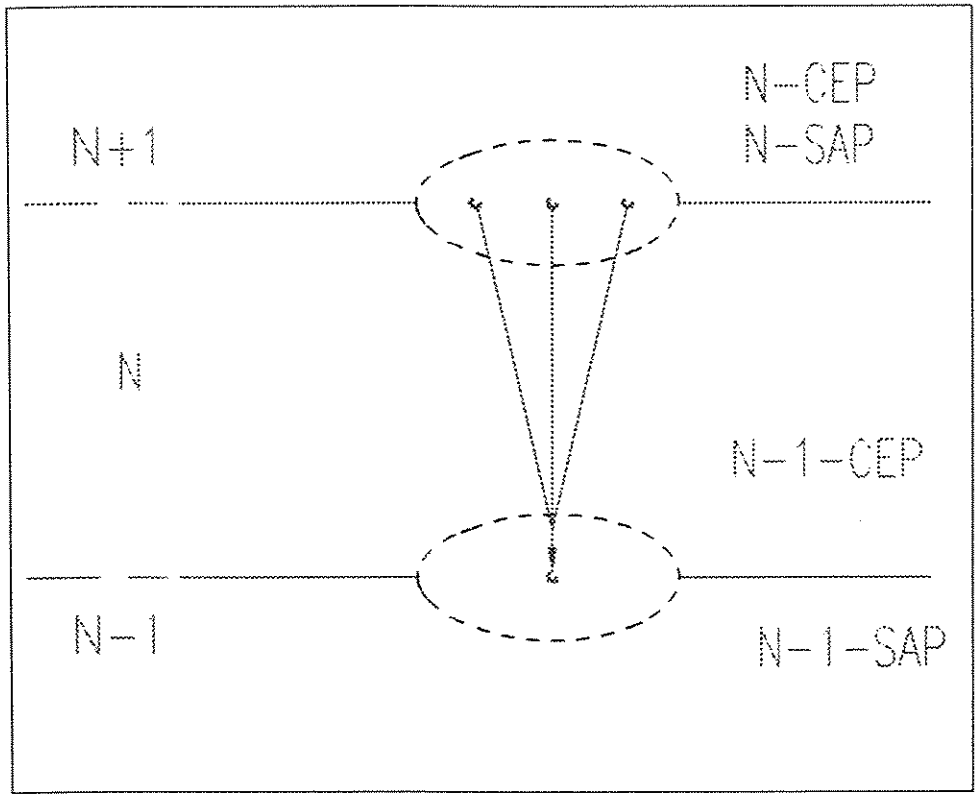


Figura 2.13: Multiplexação

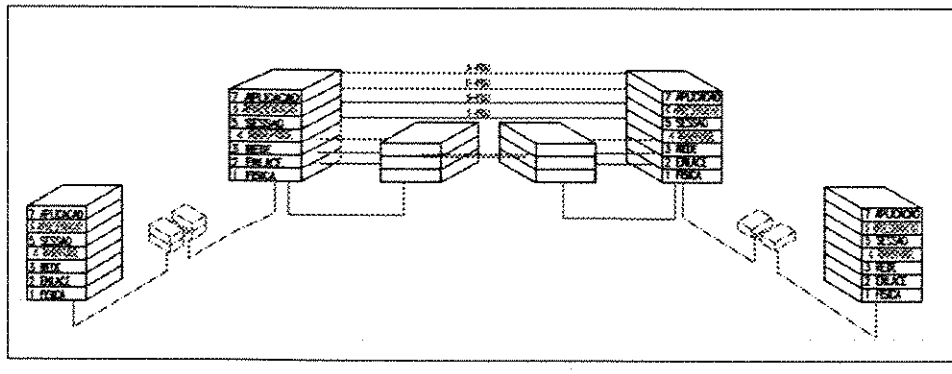


Figura 2.14: OSI-RM

(Fig. 2.12) para suportar apenas uma conexão de uma camada corrente. Esta técnica é chamada de *splitting* sendo necessário a recombinação na camada par. Quando uma entidade utiliza-se de uma conexão da camada inferior para diversas conexões da camada corrente chamamos de multiplexação (Fig. 2.13), sendo a demultiplexação feita na camada par.

Algumas vezes nesta seção estaremos nos referindo à qualidade de serviço, querendo referenciar não apenas à taxa de erro do meio, mas também a outros parâmetros como tempo de atraso, manutenção do custo estabelecido, disponibilidade do serviço, confiabilidade, taxa de transmissão e tempo de estabelecimento de uma conexão.

O OSI-RM estabelece 7 camadas para uma rede aberta descrevendo o objetivo de cada camada (Fig. 2.14):

Camada 7: Aplicação;

Camada 6: Apresentação;

Camada 5: Sessão;

Camada 4: Transporte;

Camada 3: Rede;

Camada 2: Enlace; e

Camada 1: Física.

A seguir detalharemos os serviços e as funções de cada camada do OSI-RM. Os serviços de uma camada são aquelas atribuições *vistas* pela camada usuária, como por exemplo o encaminhamento de SDUs para uma

a camada par do usuário. As funções são as atribuições da camada. Toda camada, exceto a de aplicação, oferece à camada superior os serviços de encaminhamento de SDUs e os *pontos de identificação da conexão* (CEPs), quando existe conexão. Um serviço comum a toda camada de rede é a troca de PDUs com a camada par. Algumas funções, como a detecção de erros, podem ser efetivadas em diferentes camadas, buscando otimizar o desempenho de uma rede. É desejável que tais funções sejam executadas por uma camada apenas quando necessárias. Cabe também a cada camada o seu gerenciamento.

Camada Física

Na camada física é onde os bits são transmitidos, de forma seqüencial. Os padrões da camada física descrevem-na mecanicamente e eletrica/fotonicamente definindo também o meio físico. A camada física enxerga os dados a serem transmitidos apenas como bits, não fazendo nenhum interpretação além do nível lógico de cada bit. Pode ser provido pela camada física o seqüenciamento, a notificação de falhas e parâmetros de qualidade de serviço.

Camada de Enlace

Na camada de enlace há o controle de acesso ao meio físico e o estabelecimento de enlaces lógicos via camada física. Um meio físico pode ser partilhado por diversos sistemas numa rede. Torna-se necessário um método de utilização deste meio, controlado pela camada de enlace. É necessário também o controle de fluxo, escrita e leitura de quadros, correção/detecção de erros da camada física, etc. As *Link-PDUs* (L-PDUs) são também chamadas de quadros. A camada de enlace pode prover para a rede os serviços de seqüenciamento, notificação de erro, controle de fluxo e parâmetros de qualidade de serviço. O mapeamento de serviços da camada física, *splitting*, sincronização a nível de quadro, seqüenciamento, detecção de erros, recuperação de erros, controle de fluxo, identificação e troca de parâmetros são funções da camada de enlace.

Camada de Rede

Roteamento é um conceito descrito em [ISO82] e importante para a camada de rede. E. Krol faz uma boa analogia sobre roteamento em [Kro89]. Imaginemos uma pequena criança tentando chegar a uma mesa em um restaurante. Para um adulto, com estatura bem mais elevada, fica fácil vizualizar os caminhos possíveis até a mesa desejada, mas para

a criança, que não tem a mesma visão, é difícil achar o melhor caminho, sem que lhe seja indicado. O mesmo problema temos para enviar os dados por uma rede com topologia diversa. O roteamento é a principal tarefa na camada de rede.

As entidades da camada de enlace *enxergam* apenas as outras entidades pares dos sistemas ligados à mesma sub-rede que elas. Assim quando um sistema precisa mandar mensagens para outro sistema não diretamente ligado a si, é necessário a utilização da camada de rede. São serviços prestados pela camada de rede para a de transporte o endereçamento de rede, parâmetros de qualidade de serviço, notificação de erros, seqüenciamento, controle de fluxo, transferência de *N*etwork *SDU*s (N-SDUs) urgente, reiniciação da comunicação, e desistência da comunicação.

Multiplexagem de conexões de rede, segmentação e empacotamento, detecção de erro, correção de erro, seqüenciamento, controle de fluxo, transferêcia de dados urgente, reinicação, e seleção de serviço são funções da camada de rede.

A ligação entre dois meios físicos de comunicação (*hops*) pode ser feita a nível de camada física, de camada de enlace ou de camada de rede (Fig. 2.14). O elemento que faz a ligação entre dois ou mais meios físicos, chamado na *ARPA Net*work (ARPANET) de *I*nterface *M*essage *P*rocessor (IMP) ([Tan88]), quando trabalha a nível de camada física é chamado de repetidor, e tem a função de amplificar um sinal ótico/eletrônico atenuado e repassá-lo para o outro meio físico. Quando o IMP trabalha a nível de camada de rede, chamamos de ponte, e tem a função de, por exemplo, interligar duas camadas de enlaces com entidades diferentes, por exemplo *token-ring* e Ethernet[2], ou de possibilitar a ligação de mais um *hop* quando o número máximo em uma rede já foi atingido. Quando o IMP trabalha a nível de camada de rede, chamamos de roteador e tem a função de interligar duas ou mais sub-redes em uma rede.

A Camada de Transporte

À camada de transporte cabe prover a transmissão das mensagens da camada de sessão, com a qualidade desejada, provendo um canal de comunicação independente dos aspectos físicos existentes. Quando necessário é feita a segmentação ou fragmentação, empacotamento ou concatenação da mensagem, já que as entidades de transporte podem oferecer serviços com MTU ilimitado ou maior do que o MTU da camada de rede. Na camada de transporte pode ser feita correção/detecção de erro fim-a-fim. Aqui temos a primeira camada fim-a-fim, já que nas camadas de rede,

[2]Ethernet é marca registrada da Xerox Corporation.

enlace e física as conexões são *hop*-a-*hop*. A camada de transporte é a última do chamado sistema de transporte ([Men90]). São funções da camada de transporte o mapeamento do endereço de transporte em endereço de rede, a multiplexação fim-a-fim de conexões de transporte em conexões de rede, controle do seqüenciamento fim-a-fim em cada conexão, a detecção de erros fim-a-fim e monitoração da qualidade de serviço, a correção de erros, a segmentação, empacotamento e concatenação fim-a-fim, o controle de fluxo fim-a-fim em cada conexão, e a transferência de *T*ransport *SDU*s (T-SDUs) urgentes.

Camada de Sessão

O serviço de quarentena é uma facilidade da camada de sessão que permite que um determinado número de *S*ession *SDU*s (S-SDUs) enviados via uma conexão de sessão estejam disponíves para a camada de apresentação par apenas quando indicado pela entidade usuária remetente ([ISO82]).

Na camada de sessão são estabelecidas sessões para as entidades da camada de apresentação. O estabelecimento de uma sessão permite à entidade usuária utilizar de diversos serviços simultaneamente, assim pode ser aberta uma sessão para transferência de arquivos e simultaneamente para *log* remoto num mesmo computador com multiprocessamento. Se for necessária a gerência do diálogo entre dois sistemas, e.g., controle de tráfego numa conexão *half-duplex*, ela será feita na camada de sessão. Alguns serviços de rede necessitam da existência de *tokens*, que autorizam o sistema possuidor do *token* a executar alguma tarefa de acesso restrito, como, por exemplo, o acesso a uma operação crítica que deva ser executada por no máximo um sistema a cada instante. Esta é outra tarefa da camada de sessão. A sincronização, é outro serviço da sessão. Vamos imaginar que esteja havendo a transferência de um arquivo grande, que demore, digamos, 2 horas para ser transferido, e que ao final da 1a hora algum sistema IMP situado no único caminho entre os dois sistemas finais tenha uma *pane* de 5 minutos. Se não houvesse uma função de sincronização seria necessária toda a transmissão do arquivo novamente. Havendo tal função só uma pequena fração já transmitida será retransmitida, além, é claro, do restante do arquivo ([Tan88]). A camada de sessão é a primeira das camadas orientadas à aplicação ([Hal88]). Cabe à camada de sessão prestar para a camada de aplicação os serviços de quarentena, a transferência de dados urgente, a gerência de atividades, a sincronização de conexões de sessão e a sinalização de exceção.

Cabe à camada de sessão as funções de mapeamento das conexões de sessão nas conexões de transporte, de controle de fluxo nas conexões de

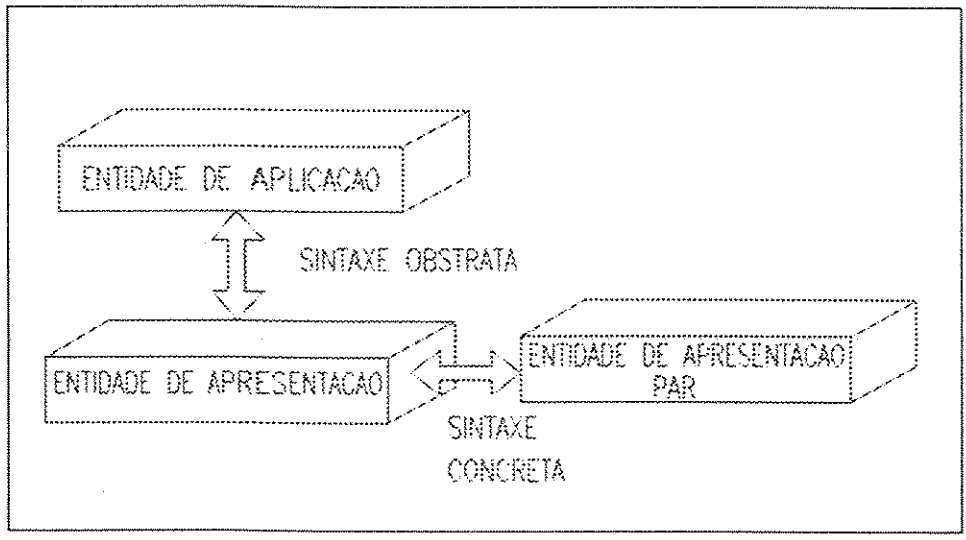


Figura 2.15: Sintaxes Convertidas pela Camada de Apresentação

sessão, de transferência de dados urgente e de recuperação de conexão de sessão.

Camada de Apresentação

À camada de apresentação cabe a representação da informação trocada entre entidades pares da camada de aplicação. Assim, aqui devem ser estabelecidas as regras sintáticas de representação da informação, e.g., a representação de inteiro, reais, caracteres, tipos complexos, etc. Sem a existência das funções da camada de apresentação seria inviável a interligação de dois computadores com representação de caracteres diferentes, como, por exemplo, *A*merican *S*tandard *C*ode for *I*nterchange *I*nformation (ASCII) e *E*xtended *B*inary *C*oded *D*ecimal *I*nterchange *C*ode (EBCDIC). É prevista uma sintaxe comum abstrata para representação dos dados, cabendo à camada de apresentação a transformação desta sintaxe comum em uma sintaxe concreta reconhecida por qualquer camada de apresentação aderente ao OSI-RM. É possível também a adoção de uma sintaxe qualquer entre a camada de aplicação e a de apresentação no lugar da sintaxe abstrata (Fig. 2.15). Cabe à camada de apresentação prestar os seguintes serviços à camada de aplicação:

- negociação da sintaxe da transferência apropriada para a troca de

mensagens;

- transformação das mensagens da entidade usuária, na sintaxe abstrata (comum ou não) entendida pela camada de apresentação par;
- transformação das mensagens recebidas da camada de apresentação par na sintaxe abstrata para a entidade usuária; e
- mapeamento dos serivços da camada de sessão.

Cabe à camada de apresentação as seguintes funções:

- pedido de estabelecimento de sessão;
- transferência de dados;
- negociação e re-negociação das sintaxes de traferência adequadas ao tipo de dado transferido;
- conversão de sintaxes; e
- finalização de uma sessão.

Algumas funções como criptografia e compactação de dados, por estarem relacionadas à forma de representação da informação, são funções típicas da camada de sessão ([ISO82, Hal88, Men90, Tan88]).

Camada de Aplicação

A camada de aplicação é a interface da aplicação, usualmente um programa, com a rede. Muitos aspectos necessários numa rede não são previstos nas camadas descritas até agora. Estas funções devem ser providas na camada de aplicação. Em [ISO82] são citadas:

- identificação dos pares envolvidos;
- determinação da disponibilidade de comunicação no instante desejado;
- estabelecimento da autoridade para comunicação;
- estabeleciemnto de um mecanismo de privacidade;
- autenticação dos pares envolvidos;
- determinação do método de cálculo do custo;
- determinação dos recursos envolvidos;

- determinação da qualidade de serviço aceitável;
- sincronização de aplicações cooperativas;
- selecionamento da disciplina envolvida no diálogo;
- acerto na responsabilidade de recuperação de falhas;
- acerto no processo de recuperação de falhas;
- identificação na sintaxe adotada.

Como veremos na seção 2.3.7, é comum o agrupamento de funções da camada de aplicação nas chamadas *A*plication *E*ntities (AEs).

Como vimos nas descrições das funções e serviços de cada camada do OSI-RM, é comum a duplicação de serviços em camadas diversas. Cabe-nos enfatizar que o [ISO82] estabelece que alguns destes serviços e destas funções têm sua localização em determinada camada facultativa. Outras vezes o oferecimento de um serviço por determinada camada à camada superior se faz apenas através do mapeamento do mesmo serviço oferecido pela camada inferior.

Muito embora o documento [ISO82] não fale em primitivas de comunicação, seu uso é constante na especificação dos serviços de uma entidade aderentes ao OSI-RM. Duas camadas adjacentes no OSI-RM utilizam-se de primitivas para *conversarem* entre si. Quatro tipos de primitivas são previstas ([Tan88]):

request: indica que uma entidade deseja um determinado serviço;

indication: uma entidade está sendo informada de um evento;

response: uma entidade está respondendo a um evento; e

confirm: uma entidade está sendo informada sobre um request seu.

Uma entidade que utiliza-se das quatro primitivas para prover um determinado serviço iniciaria o serviço com o recebimento de uma primitiva de request, por exemplo, que faria a sua entidade par enviar à sua camada superior uma primitiva tipo indication. A camada par receberia em resposta à indication uma primitiva de response, que causaria o envio de uma primitiva de confirm para a entidade usuária requisitadora do serviço.

Algumas críticas são feitas ao OSI-RM e aos serviços/protocolos definidos pela ISO ([Tan88]) tais como o excesso de camadas com diferentes graus de importância, complexidade na definição dos serviços e protocolos, ineficiência, duplicação de funções como endereçamento, controle de fluxo e controle de erro. De fato a definição de um modelo de redes de comunicação

que pretende, se não a eternidade, pelo menos ter vida longa não é uma tarefa simples. Se levarmos em conta o crescimento vertiginoso da industria de computadores, acompanhada pela industria da comunicação, possivelmente esse fato ajudou a precipitar o atual modelo. Parece-nos claro que a duplicação de funções deve ser evitada, mas parece-nos importante também a existência de diversas funções de rede, como, por exemplo, acesso ao meio físico, correção de erros, roteamento, separação da aplicação dos detalhes físicos, oferecimento simultâneo de serviços, oferecimento concorrente de um mesmo serviço, recuperação de falhas, padronização da sintaxe de comunicação e padronização da interface com a aplicação. A implementação de diversos deste serviço, na forma de *hardware* ou de funções de biblitecas, evita a duplicação desnecessária de esforços no desenvolvimento de aplicações que necessitam de comunicação, além de favorecer a padronização. Olander, McGrath e Israel ([OMI86]) destacam a implementação da camada física e de enlace em *hardware*, e da incorporação a nível de sistema operacional das camadas de rede e transporte, apontando uma tendência de absorção das camadas de transporte e de rede pelo *hardware* e da incorporação das camadas de sessão, apresentação e aplicação pelo sistema operacional, tornando as redes aderentes aos OSI-RM mais eficientes.

2.3 Padrões para as Camadas de uma Rede Aberta

Nesta seção ilustraremos cada camada do OSI-RM com pelo menos um padrão. Alguns destes padrões não estão ainda na forma de *draft purpose*, como é o caso da sub-camada MAC *D*istributed *Q*ueue *D*ual *B*us (DQDB). Outros têm sua origem no IEEE ou no CCITT, sendo que quase todos foram adotados ou são inerentes à ISO. Uma lista dos padrões da ISO pode ser encontrada em [Cha89].

2.3.1 Entidades Físicas

V.24

Um dos padrões mais populares de camada física é o EIA RS-232, especificado pelo CCITT através do padrão V.24 ([CCI89f]), utilizado não apenas para interconexão de computadores, mas também para sua ligação a periféricos e mesmo de controladores de chão de fábrica, como, por exemplo, CLPs, a computadores supervisores, como já foi visto na seção 2.1.4. Neste padrão especifica-se desde as características mecânicas, como as

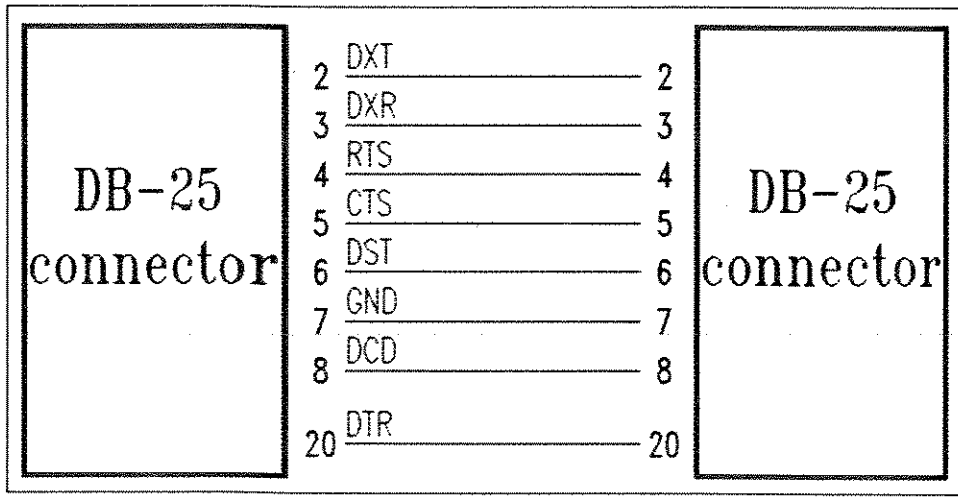


Figura 2.16: Esquema de Ligação da RS-232

características dos sinais elétricos e a lógica envolvida. O conector da RS-232C, 3a versão RS-232, conhecido como *db25*, tem suas dimensões determinadas. Este padrão prevê a ligação de terminais de dados ou computadores, aqui chamados de *D*ata *T*erminal *E*quipment (DTE), em dispositivos que viabilizam a transmissão de sinais digitais pela rede telefônica analógica, convencional, os modens, chamados de *D*ata *C*ircuit-terminating *E*quipment (DCE). A ligação entre um DTE e um DCE está esquematizada na Fig. 2.16. Aqui quando uma entidade tem dados para serem enviados ela indica o fato à RS-232 par através do sinal *R*equest *t*o *S*end (RTS), aguardando a autorização vinda no *C*lear *t*o *S*end (CTS). Os dados são enviados pelo sinal Transmit Data (TXD) e recebidos pelo Receive Data (RXD) no DTE. O sinal *D*ata *C*arrier *D*etect (DCD) serve para o DCE indicar que recebeu a portadora ao DTE. Através do *D*ata *T*erminal *R*eady (DTR) o equipamento indica ao equipamento par que está ligado e é indicado que o par está também ligado através do *D*ata *S*et *R*eady (DSR). O pino 7 é o neutro e o 1 o terra. O V.10 estabelece que o comprimento máximo do cabo que interliga as duas interfaces é dependente da taxa utilzada, a uma taxa típica de 9600 baud chega-se a uma distância pouco inferior a 100 m. O V.6 estabelece a taxa máxima de 14.400 baud. Os dados são mandados serialmente através do TXD precedidos de um *start bit* (%0), 7 ou 8 *bits* de dados, 1 *bit* facultativo de paridade, e 1 ou 2 *stop bits* (%1). Andrew S. Tanenbaum descreve em [Tan88] a RS-449 como a evolução da RS-232, possuindo a mesma especificação funcional, mecânica e

procedural e com duas variações de interfaces elétricas RS-423 e RS-422. A primeira, RS-423A, similar à RS-232C, com os circuitos possuindo um único terra comum e a segunda especificada na RS-422, utiliza-se de transmissão balanceada, tendo dois fios para cada circuito. Tando a RS-232C quanto a RS-449 possuem mais alguns sinais não descritos nesta seção que permitem a existência de duas interfaces num único conector ou mesmo a transmissão síncrona.

X.21

Já em 1986 o CCITT padronizou uma interface física digital, o padrão X.21 ([CCI89e]), descrita em [Tan88]. No X.21 temos uma interface com 8 fios. Os sinais *T*ransport (T) e *C*ontrol (C) são utilizados para o DTE enviar os dados e informação de controle. O mesmo ocorre com os sinais *R*eceive (R) e *I*ndication (I) com relação ao DCE. O sinal *S*ignal (S) provê ao DTE sincronismo a nível de bit para o recebimento dos dados do DCE. O sinal opcional *B*yte Timing (B) provê sincronismo a nível de octeto. Caso ele não seja usado é necessário prover o sincronismo por via dos sinais de dados. A linha Neutro do DTE (Ga) é o neutro e a *G*round (G) é o terra. O diagrama da Fig. 2.17 exemplifica uma comunicação bem sucedida utilizando X.21. Quando não está havendo comunicação os sinais C, T, I e R ficam desligados, na notação do CCITT em %1. O DTE inicia a comunicação ligando o C e ativando o T, o DCE solicita o endereço do destinatário enviando via R o ASCII "+"seguidamente, recebendo o endereço em seguida via T. O DCE indica o prosseguimento da chamada ao DTE via R. A efetivação da chamada é confirmada com a ativação do I e com o R enviando %1. Finalmente os dados são trocados. A comunicação é encerrada por iniciativa, digamos, do DTE desligando-se o C e fazendo-se T <= 0 e o desligamento do I, em seguida o R é colocado em %1 e o mesmo ocorre com o T, finalizando a comunicação.

Camada Física do Acesso Básico da RDSI

Falaremos sobre a camada física do acesso básico da RDSI. Uma descrição mais detalhada desta camada pode ser encontrada em [Bap90, AAG+90] e sua descrição formal em [CCI89h].

Concebido para ficar nas dependências do usuário, o acesso primário prevê uma rede baseada em pares trançados fornecendo-lhe uma taxa de 144`kbits/s`, dois canais Bs, e um canal de 16`kbits/s`, o canal D. Na configuração ponto-a-ponto a distância entre os *T*erminal *R*esistors (TRs) — há um na conexão do *T*erminal *E*quipament (TE) com os pares trançados e outro na conexão da *N*etwork *T*erminator (NT) — pode chegar a 1000`m`,

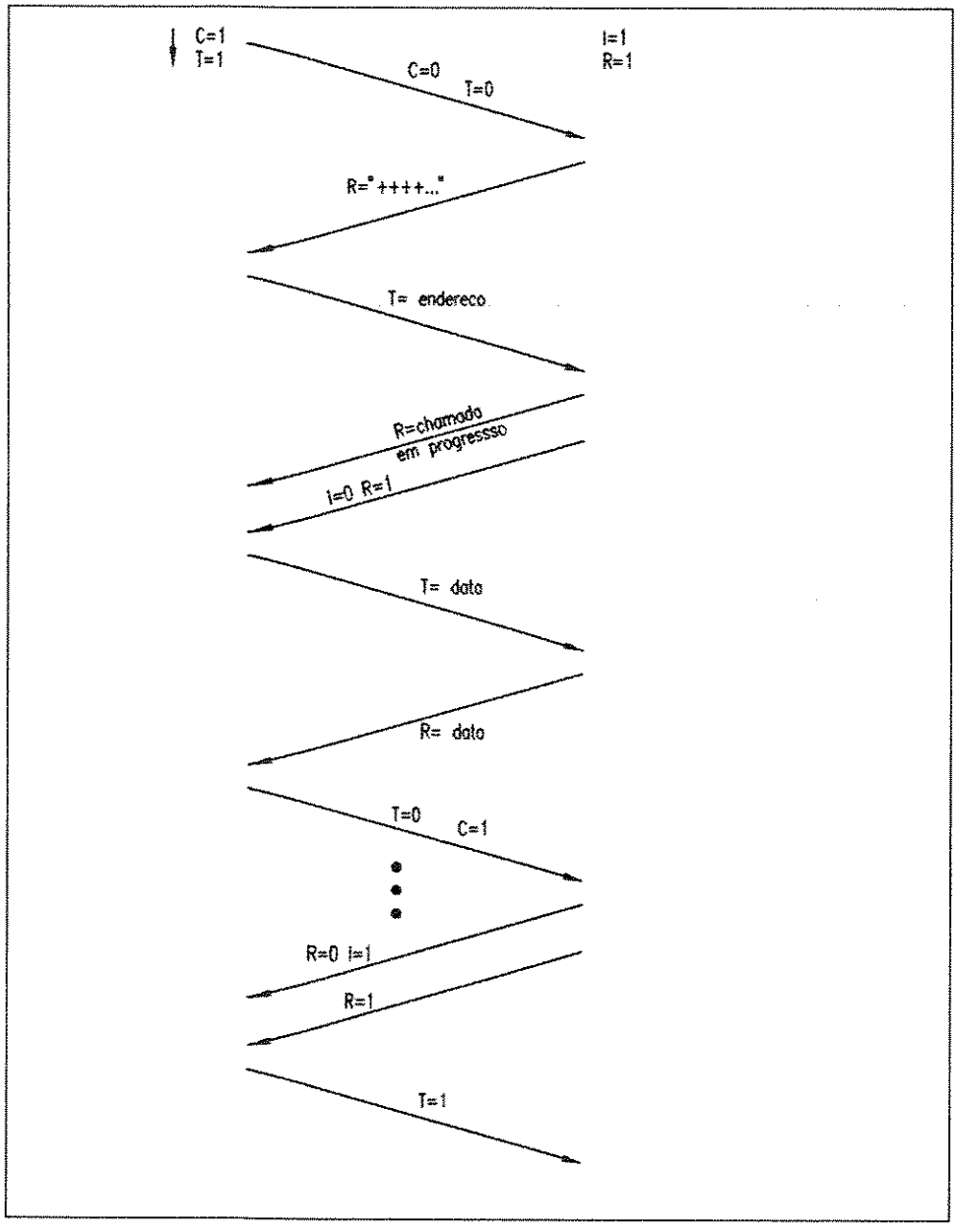


Figura 2.17: Exemplo de Utilização do X.21

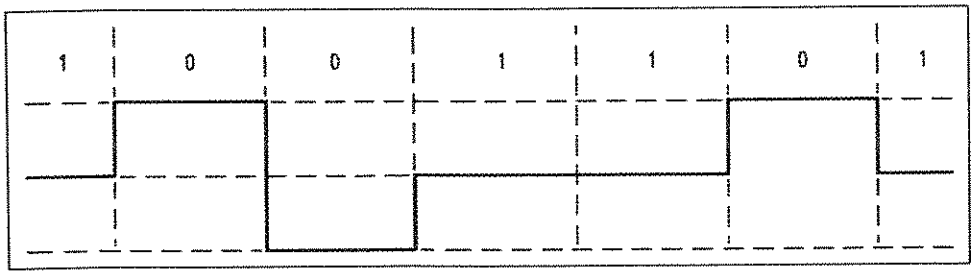


Figura 2.18: AMI Invertido

já na configuração barramento podemos ter até oito TEs ligados em paralelo num barramento de 100m/75Ω ou 200m/150Ω. Originalmente previsto para funcionar numa configuração mínima com dois pares trançados, que transmitem de forma unidirecional e balanceada, já é possível utilizar-se de apenas uma par trançado ([BSF+93]), aproveitando assim a base instalada para telefonia tradicional. A codificação utilizada é a chamada *A*lternate-*m*ark-*i*nverted (AMI) invertido (Fig. 2.18) com violação de código, onde os dígitos %0 são representados por um pulso de polariadade invertida em ralação ao %0 anterior — exceto em alguns instantes pré-determinados onde a polariadade é mantida (violacão do código) — e os dígitos %1 são representados pela ausência de sinal.

Camada Física do Acesso Primário da RDSI

Falaremos sobre a camada física do acesso primário da RDSI. Uma descrição mais detalhada desta camada pode ser encontrada também em [Bap90, AAG+90] e sua descrição formal em [CCI89h].

Estão padronizadas duas interfaces usuário/rede para o acesso primário da RDSI, uma inspirada nas redes síncronas europeias, e adotada no Brasil, que oferece uma taxa de 2048`kbits/s` e outra inspirada nas redes síncronas americanas que oferece uma taxa de 1544`kbits/s`. Para o acesso primário só é prevista a ligação ponto-a-ponto.

Outras entidades físicas serão mencionadas mais adiante, por terem sido padronizadas em conjunto com entidades da camada de enlace como é o caso dos padrões IEEE 802.3, 802.4, 802.5 e 802.6 (seção 2.3.2).

2.3.2 Entidades de Enlace

O IEEE em muito contribuiu e contribui para a especificação de padrões abertos para a camada e rede. O projeto IEEE 802 estabelece a divisão

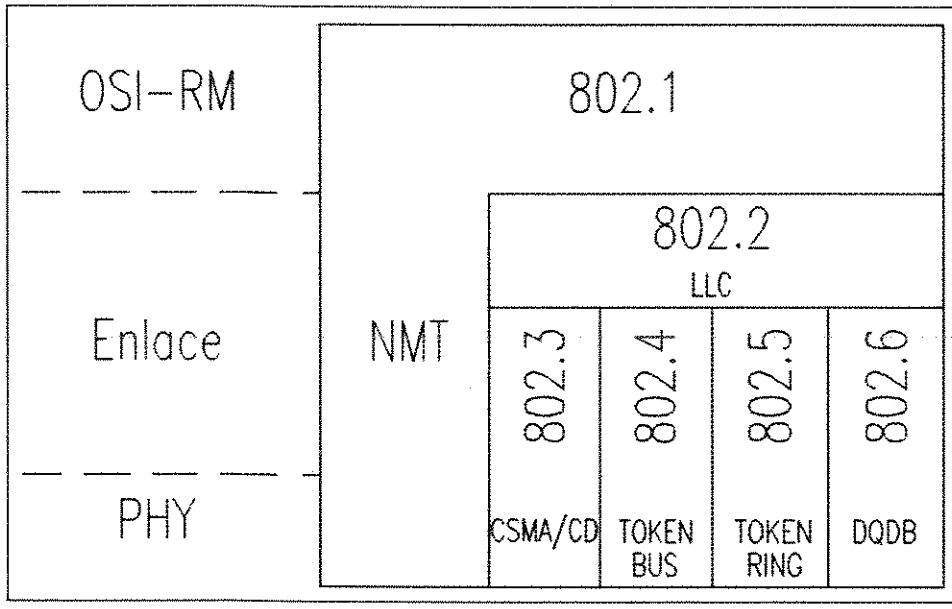


Figura 2.19: Projeto 802

da camada de enlace em duas sub-camadas: a LLC e a MAC, sendo padronizada também a camada física para cada MAC. (Fig. 2.19). A primeira é responsável pelo controle do enlace lógico e a segunda pelo método de acesso ao meio físico. A LLC é padronizada pelo documento IEEE 802.2 ([IEE85b]). Já a MAC possui diversos padrões, destacando como um padrão de fato o Ethernet, padrão IEEE 802.3 ([IEE85a]), utilizado em LANs de automação de escritório ou em redes acadêmicas, onde o requisito de tempo para acesso à rede não é determinante. O Token Ring, padrão IEEE 802.5 ([IEE85d]), e o Token Bus, padrão IEEE 802.4 ([IEE85c]) são importantes padrões utilizados também em LANs onde a restrição de atraso de comunicação é maior. Mais recentemente um padrão para LAN/MAN foi proposto também pelo IEEE, o padrão IEEE 802.6 ([IEE90]), o DQDB. Outro padrão para MAC de MAN proposto pela ANSI é o *F*iber *D*istributed *D*ata *I*nterface (FDDI).

LLC

Definem-se na LLC os serviços que a camada de enlace presta à de enlace e os protocolos de enlace. Segundo [Men90] o IEEE 802.2 estabelece dois tipos de sub-camadas LLC:

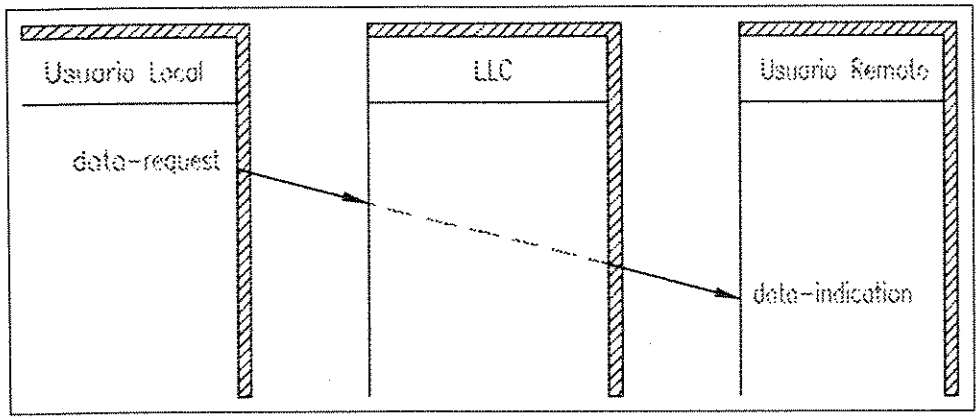


Figura 2.20: Primitivas da LLC Tipo 1

tipo 1: sem conexão; e

tipo 2: com conexão.

A LLC tipo 1 é utilizada quando deseja-se a transferência de dados com um mínimo de *overhead*. Aqui não é proporcionado nem o seqüenciamento nem a recuperação de erros, supondo que tais serviços são providos pelas camadas superiores. Assim utiliza-se apenas a primitiva *Link Data Request* (L-data.request) para solicitar o envio de dados da entidade de rede pela LLC e o *Link Data Indication* (L-data.indication) para a LLC indica a chegada de dados à entidade de rede par, conforme esquematizado na Fig. 2.20 ([Hal88]).

A LLC tipo 2 provê à camada de rede a recuperação de erros e o seqüenciamento. Para tal, é necessário o estabelecimento prévio de conexão, antes do envio dos dados. As primitivas utilizadas na LLC tipo 2 estão esquematizadas na Fig. 2.21. Para o estabelecimento de conexão é utilizada a primitiva *Link Connection Request* (L-connection.request) pelo usuário iniciador da conexão, o que acarreta num *Link Connection Indication* (L-connection.indication) ao usuário par, e um *Link Connection Confirm* (L-connection.confirm) confirmando o estabelecimento da conexão ao usuário que a iniciou. Para a transferência de dados é utilizada a primitiva *Link Data Connection Request* (L-data_connection.request) pelo usuário local, que causará uma *Link Data Connection Indication* (L-data_connection.indication) para o usuário par remoto, e uma confirmação com a *Link Data Connection Confirm* (L-data_connection.confirm) para o usuário local. O mesmo processo pode ser concomitantemente utilizado por

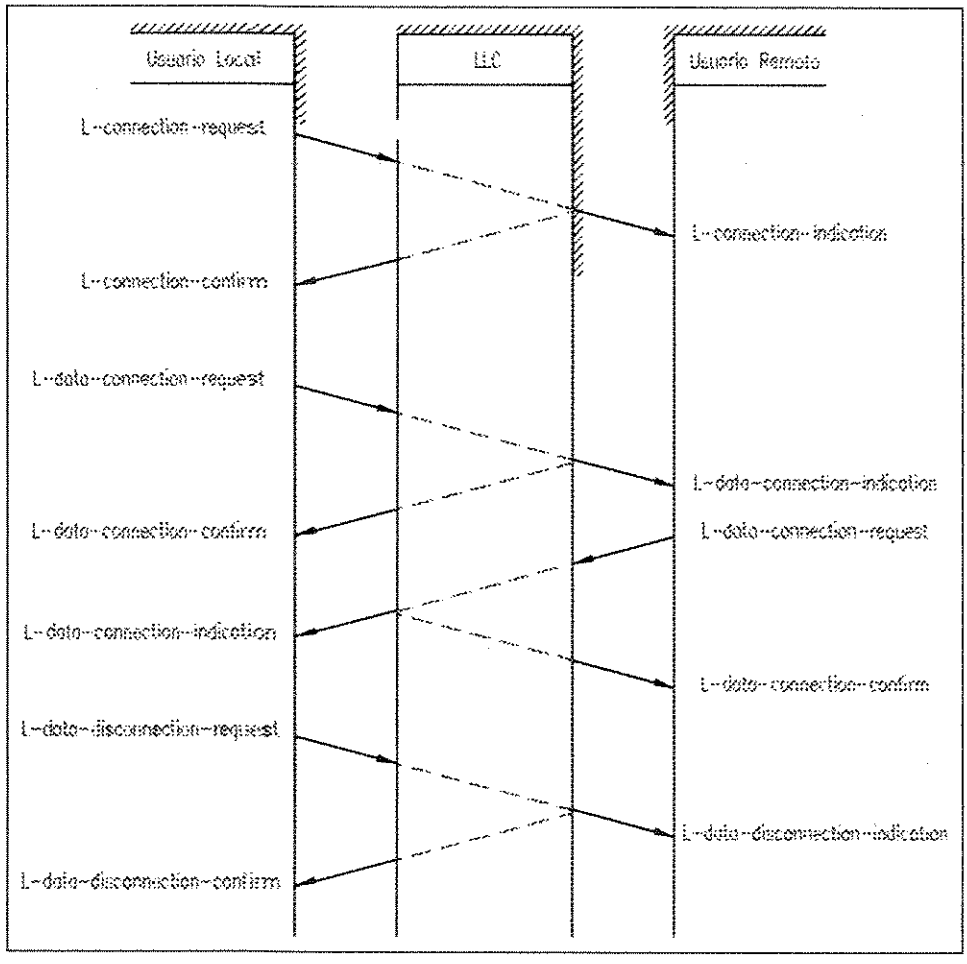


Figura 2.21: Primitivas da LLC Tipo 2

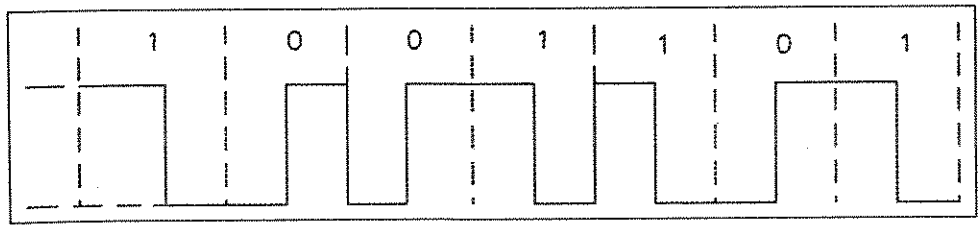


Figura 2.22: Codificação Manchester

iniciativa do usuário remoto. Na desconexão utiliza-se a *Link Disconnect Request* (L-disconnect.request), que causará o envio de uma *Link Disconnect Indication* (L-disconnect.indication) para a camada par e o envio de uma *Link Disconnect Confirm* (L-disconnect.confirm) para o usuário local. Em todos os casos citados a confirmação é enviada após o envio da indicação à camada par. [Hal88] e [Tan88] apontam a LLC tipo 2 como muito similar ao *High-level Data Link Control* (HDLC) da ISO, diferenciando-se no que diz respeito à construção dos quadros e à detecção de erros, ambos providos pela MAC. O leitor poderá encontrar uma descrição do formato dos quadros da HDLC em [Tan88].

LAPD

Comparando o *Link Access Procedure in D* chanel (LAPD) utilizado na RDSI com os protocolos HDLC/*Link Access Procedure B* (LAPB)/LLC 2 ([SR89]) chegamos às seguintes conclusões: o LAPB provê operações ponto-a-ponto confirmadas, enquanto que o LAPD provê operações ponto-a-ponto confirmadas ou não e operações *broadcast*. O quadro do LAPB diferencia-se do quadro do LAPD apenas pelo campo de endereços, que no segundo possui 2 bytes e no primeiro 1. O LAPD oferece também o serviço sem conexão, oferece primitivas especiais para o gerenciamento e serviços administrativos.

Ethernet

Baseia-se no método de acesso ao meio da rede via rádio ALOHA, desenvolvida na Universidade do Hawaii ([Arc86]): o *Carrier Sense Multiple Access* (CSMA)/*Colision Detected* (CD), pesquisadores do *Xerox Palo Alto Research Center*, em Palo Alto, *California* (CA), EUA ([MB76]). O projeto, iniciado em 1973, teve uma versão experimental que trabalhava com uma taxa de 2,94 `Mbits/s`, suportando até 256 estações numa extensão máxima de 1 `km`, utilizando como meio físico um cabo coaxial e como esquema de codificação o Manchester (Fig 2.22). Algumas dezenas de

redes desta versão experimental foram instaladas pela Xerox, mas nunca chegaram a ser comercializadas. Em 1979 a Xerox, a *D*igital *E*quipament *C*oorporation (DEC) e posteriormente a Intel engajaram-se na tentativa de tornar a Ethernet um padrão internacional. Em 1980 foi publicada pelas três a sua especificação, aqui funcionando a uma taxa de 10`Mbits/s`, podendo ser composta por segmentos de cabos coaxiais de até 500m, tendo entre dois nós da rede a distância máxima de 2.500`m`, i.e., utilizando-se até 4 repetidores entre eles. Na V2.0, desenvolvida buscando a padronização pela IEEE, esta distância máxima foi aumentada para 2.800`km` e foram inseridas funções de gerência de rede. Com uma pequena modificação no campo `Type`, a criação dos campos `Pad` e do `Preamble` e uma definição de endereçamento a nível global, a Ethernet tornava-se um padrão de direito por intermédio da especificação IEEE 802.3 ([IEE85a]).

Uma entidade de enlace Ethernet, como toda entidade de enlace, transmite pela camada física quadros. O quadro da Ethernet está esquematizado pela figura 2.23. Possui os seguintes campos:

`Preamble`: 7 octetos com %10101010 que possibilitam o sincronismo a nível de quadro;

*S*tart *F*rame *D*elimiter (SFD): 1 octeto que indicam o início do quadro: %10101011;

`Destination Address`: 2 ou 8 octetos com o endereço do destinatário;

`Source Address`: 2 ou 8 octetos com o endereço do remetente;

`Length`: 2 octetos que indica o tamanho do campo `Data`;

`Data`: de 0 a 1.500 octetos, a L-PDU;

`Pad`: de 0 a 46 octetos que garantem o tamanho mínimo do quadro de 72 bits;

*F*rame *C*heck *S*equence (`FCS`): 4 octetos que comtém o *C*yclic *R*edundancy *C*heck (CRC) do quadro (exceto `Preamble` e SFD).

Como já foi dito, o método de acesso ao meio, principal função de uma MAC, da Ethernet é o CSMA/CD, onde o nó interessado em transmitir algum quadro aguarda o canal estar vazio (*Carrier Sense*), envia o quadro e continua observando o canal ainda por um determinado tempo, buscando detectar alguma colisão (*Colision Detect*), i.e., o envio de um quadro simultaneamento por outro nó. Caso a transmissão não tenha sido bem suscedida, o nó aguarda um determinado tempo aleatório, para fazer a retransmissão.

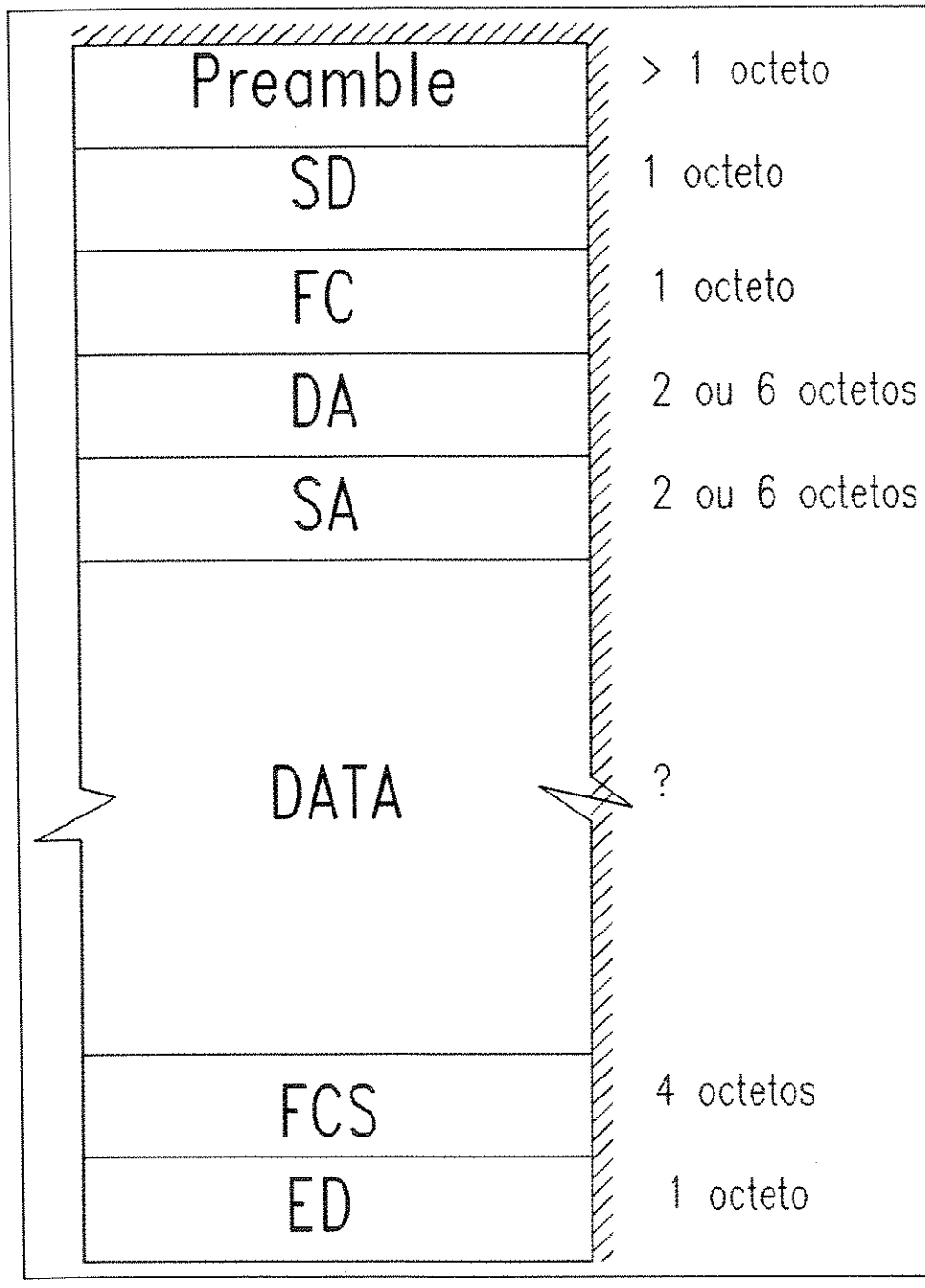


Figura 2.23: Quadro Ethernet

Algumas camadas físicas já padronizadas para utilização com a Ethernet são:

10base5: formada por segmentos de cabos de até 500 m e com uma taxa de 10 Mbits/s;

10base2: formada por segmentos de cabos de até 200 m e com uma taxa de 10 Mbits/s;

10base-F: formada por enlaces óticos à taxa de 10 Mbits/s ([Jai93]); e

10base-T: formada por enlaces *U*nshielded *T*wisted *P*airs (UTPs).

A camada física é composta por:

***Physical Signaling* (PLS):** responsável pela comunicação da MAC com a *Phy*sical (PHY);

***A*ttachment *U*nit *I*nterface (AUI):** cabo/conectores que ligam o DTE ao *M*edium *A*ccess *U*nit (MAU); e

MAU: possui a interface com o meio e os circuitos para transmissão:

***P*hysical *M*edia *A*ccess (PMA):** possui os circuitos para transmissão; e

***M*edium *D*ependent *I*nterface (MDI):** interface com o meio físico.

Token Ring

Considerando que a topologia em anel possibilita uma política de acesso ao meio mais justa, onde justiça aqui significa igualdade de tempo de uso do meio físico, e que a tecnologia utilizada em redes em anel pode ser completamente digital, lembrando que a detecção de colisão na Ethernet, por exemplo, é analógica, e que num anel temos alta ocupação da largura de faixa disponível, foi adotada tal topologia pela IBM, que desenvolveu um esquema de acesso ao meio justo e que garanta um tempo de espera para transmissão pré determinado, para quadro de alta prioridade, além de implementar um esquema de prioridade: trata-se do *Token Ring*. Em [Tan88] temos uma didática introdução ao *Token Ring*, em [Hal88] temos uma explicação mais detalhada do seu quadro e do algoritmo de controle das prioridades e em [IEE85d] temos a especificação formal.

O quadro do *Token Ring*, visto na Fig. 2.24, é composto pelos seguintes campos:

***S*tarting *D*elimiter (SD):** seqüência especial de símbolos que indica o início do quadro: JK0JK000;

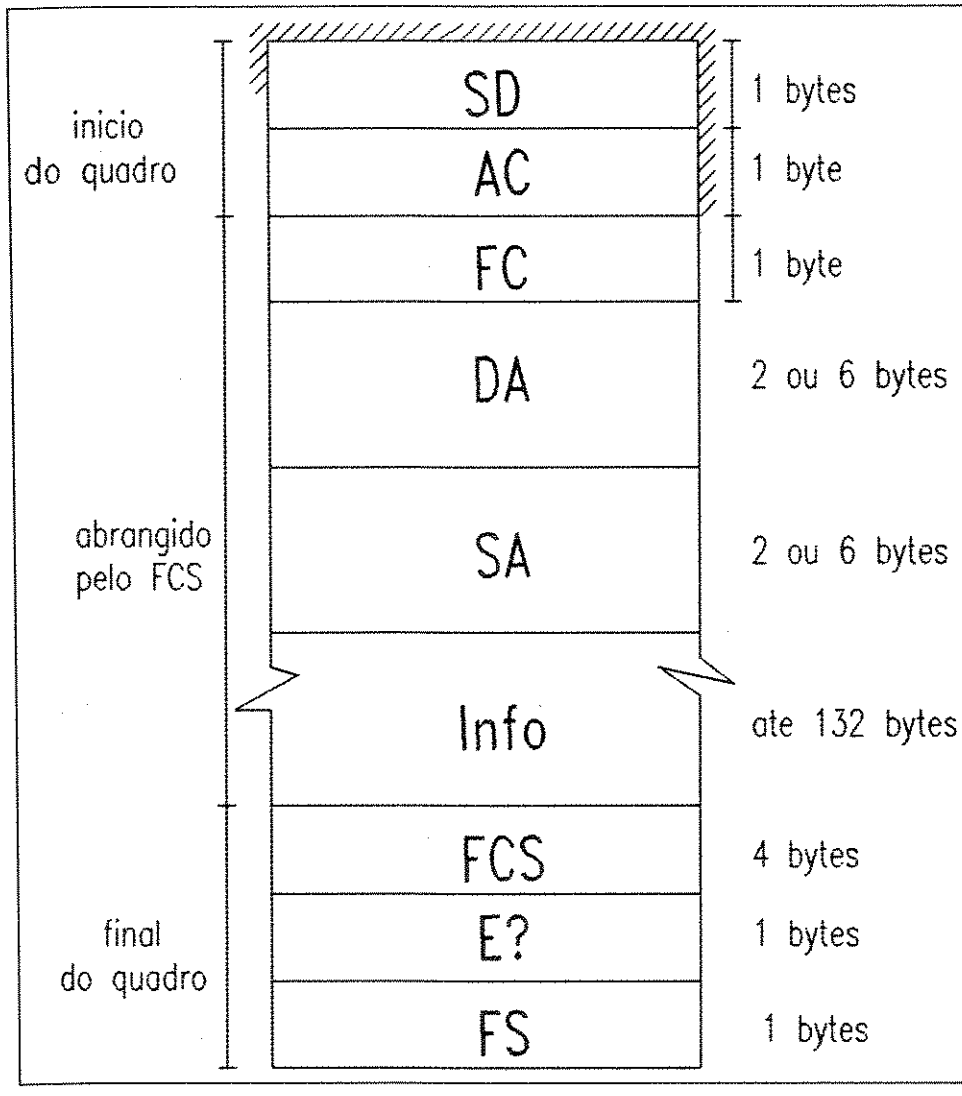


Figura 2.24: Quadro do Token Ring

Access Control (AC): utilizado para garantir o envio de quadro de maior prioridade primeiro, para indicar o formato do quadro, e também quando um quadro já passou pela estação monitora;

Frame Control (FC): indica o tipo de quadro, com LLC PDUs ou com informações da MAC;

Destination Address (DA): endereço da MAC destino;

Source Address (SA): endereço da MAC remetente;

Information (Info): informação a ser transmitida;

Frame Check Sequence (FCS): redundância de inforção para validação do quadro;

Ending Delimiter (ED): seqüência especial de síbolos que indica o início do quadro: JK1JK1IE

Binary One (I): quando 1 indica que não é único quadro de uma seqüência, quando 0 indica que é o único ou o último de uma seqüência;

Error-detected Bit (E): quando 1 indica que houve a detecção de erro por alguma MAC; e

Frame Status (FS): ACrrACrr, quando os A = 1 indica que o destino existe e quando os C = 1 indica que o pacote foi copiado no destino.

Aqui temos a cada instante no máximo um quadro circulando no anel. Inicialmente circula-se um *token*, que é um quadro especial com apenas o SD, o AC e o ED. Este *token* é transformado, pela estação que possui a mensagem com maior prioridade a ser transmitida, num quadro comum, portador da mensagem, ou de fragmento dela. A mensagem é recebida ou não pela estação destinatária, e retornando a sua origem é retirada do anel. Havendo ainda mensagens com igual ou maior prioridade a serem transmitidas, e desde que não expire o tempo de detenção do *token*, tipicamente 10ms, a estação continua a enviar quadro para a rede. Após ter enviado todos os quadros e sempre antes de expirar o tempo de detenção do *token*, este é enviado ao anel, possibilitando a sua utilização por outra estação. Em linhas gerais o mecanismo garante que sempre os quadros com maior prioridade serão transmitidos primeiro e que estações com quadro com a mesma prioridade terão igual direito ao acesso ao anel.

Muito embora [IEE85d] estabeleça o uso de par trançado como meio físico à taxa de transmissão de 1 ou 4 Mbits/s, utiliza-se hoje também a taxa de 16 Mbits/s. A codificação utilizada é a Manchester diferencial,

onde a inversão de polaridade, na primeira metade do ciclo, indica um bit 0.

A camada PHY do *Token Ring* é ligada ao *T*runk *C*oupling *U*nit (TCU) via o *M*edium *I*nterface *C*onnector (MIC), e pelo TCU passa o anel físico.

Token Bus

O *Token Bus*, assim como o *Token Ring*, baseia o acesso ao meio físico alternadamente entre os diversos nós, garantindo para isso a circulação de uma autorização, o *token*, entre todos os nós ativos na rede, formando um anel lógico, apesar de sua topologia ser em barramento. Entre as diversas diferenças entre os dois projetos destacamos a topologia adotada, barramento e anel; o tratamento das prioridades: enquanto, como vimos, no *Token Ring* garante-se a transmissão das mensagens mais prioritárias em detrimento das menos prioritárias em toda a rede, no *Token Bus* cada estação terá acesso ao anel por um determinado tempo, transmitindo então todos os envelopes que puder durante este período. Assim, ao terminar de trasmitir os envelopes mais prioritários, e havendo mais tempo, os menos prioritários serão também transmitidos. Voltado para a automação industrial, o *Token Bus* nasceu do projeto *M*anufacturing *A*utomation *P*rotocols (MAP), da *G*eneral *M*otors (GM) ([Men90]). Uma introdução ao *Token Bus* pode ser encontrada em [Tan88], uma exposição didática e mais detalhada sobre o algoritmo de prioridades pode ser encontrada em [Hal88] e a descrição forma da MAC e da PHY pode ser encontrada em [IEE85c].

O quadro do *Token Bus* é semelhante ao do *Token Ring*, não existindo aqui o campo FS e havendo aqui um `Preamble` como esquematizado na Fig. 2.25. Neste caso não temos a necessidade de retirar o quadro do meio, também não é provida pela MAC a confirmação após o retorno do quadro indicando o recebimento ou não pela MAC par, já que não há retorno do quadro. Aqui é prevista a confirmação ou quando o quadro é enviado à PHY ou, a pedido da LLC, quando a MAC recebe um quadro da MAC par indicando o *status* do último quadro enviado.

Aqui utiliza-se também a modulação do sinal na camada PHY. São previstas três entidades da PHY: (1) codificado com Manchester (Fig. 2.22) com modulação *Phase Continuous Frequency Shift Key* (FSK) a taxa de 1`Mbits/s`, (2) modulação *Phase Coherent* FSK a taxa de 5`Mbits/s` e 10`Mbits/s`, e (3) codificação/modulação *multilevel duobinary* AM/*Phase Shift Key* (PSK).

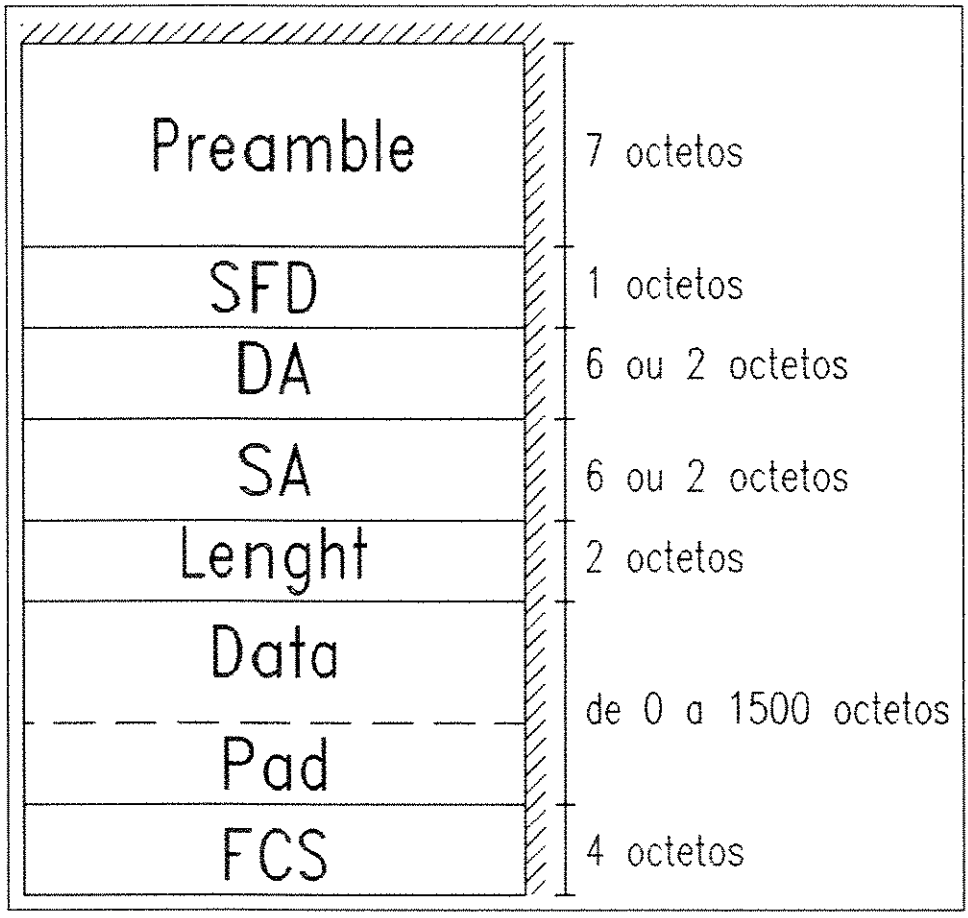


Figura 2.25: Quadro do Token Bus

FDDI e FDDI II

O FDDI é um padrão ANSI para as camadas de enlace e física de uma MAN de alto desempenho. Trata-se de uma rede inicialmente prevista para usar, como meio físico, fibra ótica, trabalhando a uma taxa de 100 Mbits/s, com topologia em anel duplo. O método de acesso ao meio é baseado no *Token Ring* com as seguintes principais diferenças: a possibilidade de existência de mais de um quadro num determinado instante na rede, dada a grande latência de cada anel, que pode ter um perímetro de até 200 km ([Tan88]); e a implementação das prioridades conforme o *Token Bus*.

Buscando prover à FDDI suporte para serviços isócronos, i.e., serviços que precisam de um determinado canal à uma taxa fixa, e.g., transmissão de voz PCM à taxa de 64 kbits/s é necessário o oferecimento de um canal de 8 bits a cada 125 μs, uma extensão à FDDI foi acrescida, formando a FDDI II. A FDDI II pode funcionar no modo básico, i.e., o moda da FDDI, mas quando estiver provendo serviços isócronos, funcionando no modo híbrido, numa rede que só tenha nós FDDI II ativos, com um período de 125 μs é gerado um quadro chamado *ciclo*, que contém 5 *slots* chamados *W*ide*b*and *C*hannels (WBCs) que garantem o oferecimento de serviços isócronos.

A FDDI II é composta pelas seguintes camadas:

MAC/MAC-II: controla o método de acesso ao meio ([ISO89a, ANS92a]);

*H*ibrid *R*ing *C*ontrol (HRC): implementa o serviço isócrono ([ANS92b]), não existe na FDDI;

PHY/PHY-II: faz a codificação/decodificação e é responsável pela sinalização ([ISO89b, ANS92c]);

*S*ingle-*m*ode *F*iber (SMF-PMD): camada física que trabalha com *S*ingle-*m*ode *F*iber (SMF), permitindo enlaces mais longos sem a necessidade de repetidores;

*L*ow-*c*ost *F*iber *PMD* (LCF-PMD): camada física que trabalha com *M*ulti*m*ode *F*iber (MMF), permitindo o uso de fibra a custos mais baixos;

*T*wisted-*p*air *PMD* (TP-PMD): camada física que trabalha com par trançados a taxa de 100 Mbits/s; e

*S*tation *M*anagement (SMT): gerência da rede.

Essas camadas estão esquematizadas na Fig. 2.26. O uso do par trançado possibilita a FDDI II utilizar a vasta rede de pares-trançados existente.

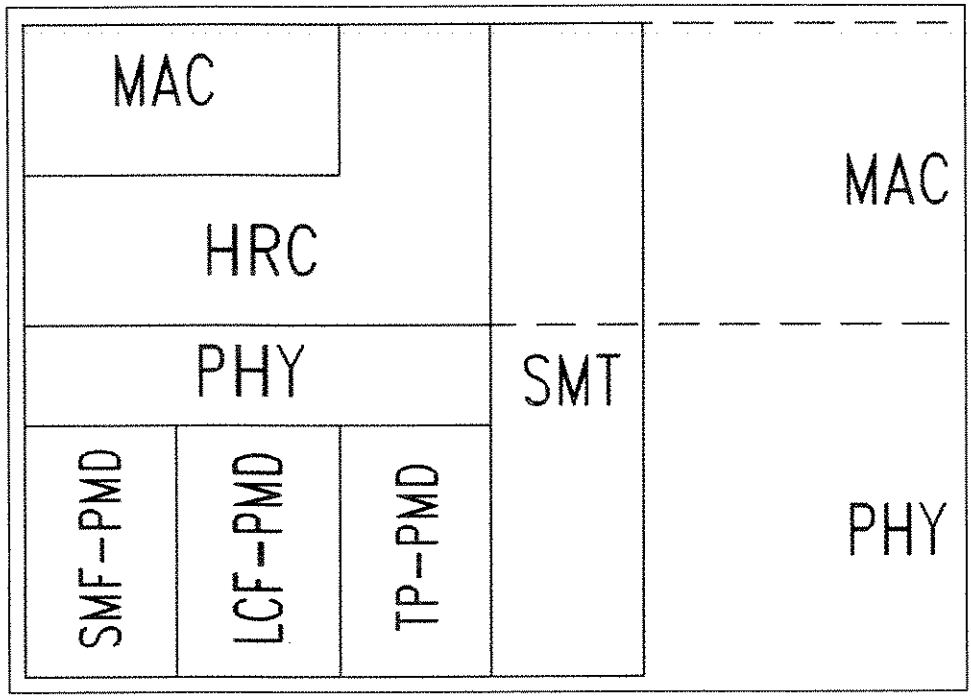


Figura 2.26: Camadas da FDDI-II

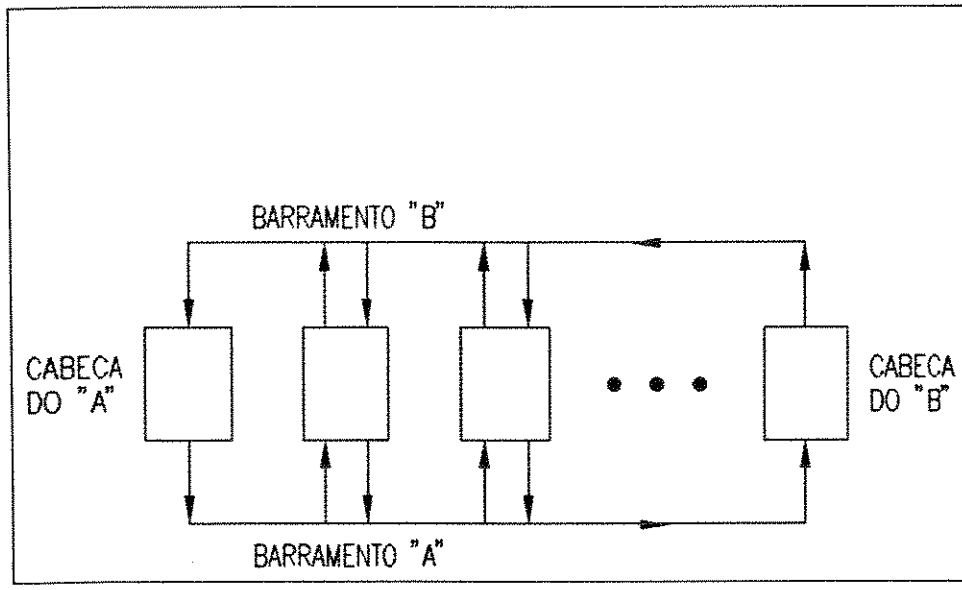


Figura 2.27: Topologia do DQDB

Surgindo como uma rede que possibilita a ligação de redes locais e redes FDDI II está sendo proposta a *F*DDI *F*ollow *O*n (FFOL) trabalhando à elevada taxa de até 2,5 `Gbits/s`, sendo capaz de interligar redes *A*ssyncronous *T*ransfer *M*odes (ATMs) ([Jai93]).

Recentemente Raj Jain fez uma análise dos aspectos atuais e planos futuros para as redes FDDI ([Jai93]), citando algumas referências básicas para FDDI: [BR84, RM84, Ros86, Ros89, Ros91a, HGH91], para FDDI II: [CF86, Ros91b], e para FFOL: [OH90, RF92].

DQDB

Ainda em discussão, está sendo proposto pelo IEEE como padrão para MAN o DQDB, o padrão IEEE 802.6. no capítulo 2 de [IEE90] encontramos uma detalhada introdução da MAC DQDB, tendo a maioria dos seus pontos formalmente detalhada no restante deste documento, havendo ainda alguns pontos, como o oferecimento de serviços com conexão, ainda não especificados.

No DQDB temos dois barramentos unidirecionais com sentidos opostos, como esquematizado na Fig. 2.27. Há duas formas de acesso ao meio: (1) *Q*ueue *A*rbitrated (QA); e (2) *P*re-*a*rbitrated (PA). No QA um usuário

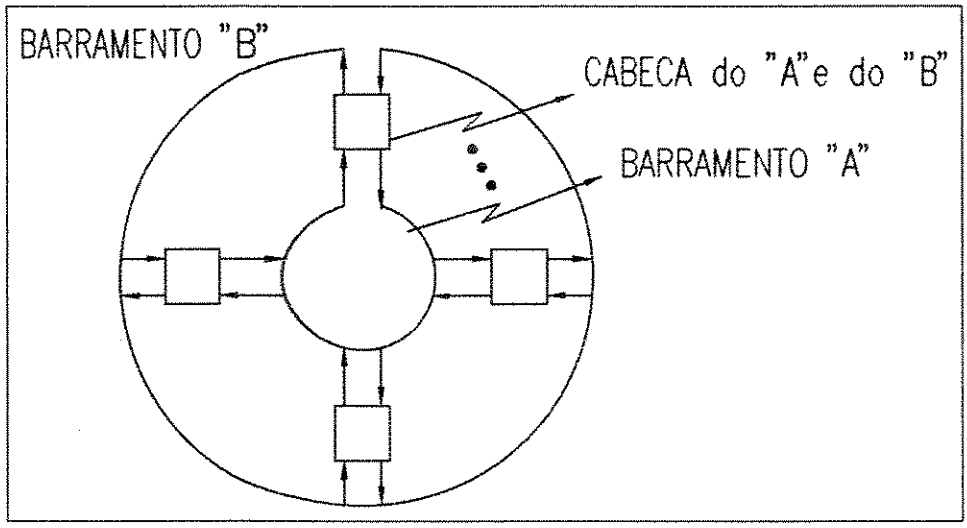


Figura 2.28: Anel DQDB

que queira enviar um pacote no barramento {A,B} solicitará um *slot* vazio no barramento {B,A} indicando o fato às estações entre ele e o final do barramento {B,A}. Este usuário, sabendo quantos usuários anteriores a ele no barramento {B,A} solicitaram previamente um *slot* vazio, deixará passar no barramento {A,B} o número de *slots* vazios necessário para atender esses usuários para finalmente transmitir a sua mensagem. No PA o nó cabeça do barramento reservará uma determinada quatidade de *slots* que serão partilhados pelos diversos nós do barramento, conforme definido pelo cabeça do barramento. Ao necessitar do acesso PA o usuário indica à DQDB qual a largura de faixa que deseja, i.e., quantos `bits` irá necessitar a cada 125 μs. O nó solicitará o recurso ao cabeça do barramento desejado que indicará ao nó solicitante qual/quais os *slots* e em que posição destes *slots* o solicitante poderá utilizar para enviar a mensagem, garantindo o oferecimento do *slot* periodicamente. É possível utilizar uma configuração onde as duas cabeças de barramento estejam na mesma estação, formando um anel duplo (Fig. 2.28), não havendo porém a circulação de *slots* de um cabeça de barramento para o final do barramento.

No DQDB há três seviços oferecidos ao seu usuário: (1) sem conexão provido à LLC (IEEE 802.2); (2) isócronos; e (3) orientado a conexão. O serviço sem conexão propicia a segmentação e empacotamento da *MAC SDU* (M-SDU). O serviço isócrono utiliza-se do PA. E o serviço com conexão está ainda em estudo.

Para o acesso via QA, utilizado pelos serviços sem e com conexão, são previstos três níveis de prioridade, de tal forma que as M-SDUs de maior prioridade serão enviadas sempre anteriormente às M-SDUs de prioriade mais baixa no determinado instante.

O método de acesso QA como foi descrito privilegia as estações mais próximas do cabeça do barramento. Buscando corrigir esta distorção é previsto que cada nó poderá ocasionalmente deixar passar um *slot* vazio para uso das estações no final do barramento. Este mecanismo é chamado de *balanceamento da largura de faixa.*

Três PHYs já são previstas no [IEE90]: DS3, com taxa nominal de 44736`Mbits/s`; CCITT G.703, com taxas de 34368`Mbits/s` e 139368`Mbits/s`; e CCITT G.707-9, *S*ynchronous *D*igital *H*ierarchy (SDH). Sendo que as duas últimas ainda estão em estudo.

A arquitetura funcional de um nó DQDB prevê a existência dos seguinte blocos funcionais:

- *MAC C*onvergence *F*unctions (MCFs);
- *C*onnection *O*riented *C*onvergence *F*unctions (COCFs);
- *I*sochronous *C*onvergence *F*unctions (ICFs);
- QA Convergence Function;
- PA Convergence Function;
- Common Functions;
- *P*hysical *L*ayer *C*onvergence *F*unctions (PLCFs); e
- *L*ayer *M*anagement *E*ntities (LMEs).

As MCFs e COCFs provêem os serviços com e sem conexão utlizando-se da QA Convergence Function. As ICFs provêem o serviço isócrono utlizando-se do PA. As Common Functions provêem o acesso à PHY. As PLCFs operam independentemente do meio físico utilizado, sendo preciso uma PLCF para cada tipo de meio físico utilizado, proporcionando sempre a mesma interface para as Common Functions. Finalmente são previstas as LMEs que provêem as funções de gerência de rede para a PHY e para a MAC.

Método de Acesso ao Meio no Acesso Básico da RDSI

O quadro do Acesso Básico da RDSI possui os campos `B1` e `B2` que são utilizados como canais para tráfego de informação do usuário a 64`kbits/s`, o canal `D`, 16`kbits/s` é utilizado para sinalização e tráfego de informação.

O acesso ao canal D é feito via um método de acesso ao meio CSMA/CD, onde garante-se que sempre um terminal terá acesso ao canal no caso de colisão. Um TE, aguarda o instante em que o canal D esteja vazio, i.e. esteja trafegando apenas bits %1s (ausência de sinal), mandando então sua mensagem, cada bit é recebido pelo NT e ecoado no barramento com sentido NT→TE no próximo campo E. Se o TE tiver enviado um %0 ele sempre receberá um %0 (pulso) no E, supondo que a transmissão está correta. Se o TE enviar um %1 ele poderá receber um %0 no campo E se outro TE estiver simultaneamente utilizando o canal D, quando então detectará a colisão e passará a monitorar novamente o canal D esperando que fique vazio. Note-se que no caso de detecção de colisão sempre haverá um TE que não notará o fato, o mesmo acontecendo com o NT. É previsto também um esquema de prioridade forçando-se o TE que deseja transmitir uma informação a aguardar um determinado número de %1 variável em função da prioridade da mensagem a ser transmitida. Os campos F, F_A e M são utilizados para sincronismo. O campo A é utilizado para ativar os TEs, o campo M está sendo proposto como uma canal para transmissão de dados em baixa taxa e a utilização do S é para estudo futuro ([CCI89h, Bap90, AAG+90]).

ATM

O ATM apresenta-se como a entidade de enlace/física a suportar a *RDSI-F*aixa *L*arga (RDSI-FL). Em [Pry91, Tam89, Mot83] encontramos introduções ao ATM. Basicamente o ATM proporcionará aos seus usuários um meio de comunicação para uma variedade muito grande de aplicações. A informação numa rede ATM é transferida via células de 53 bytes cada, onde a largura de faixa necessária para cada serviço é previamente requisitada a rede que poderá ofercê-la ou não, e em caso de oferecê-la deverá policiar para ver se o usuário está utilizando o recurso desejado, podendo, em caso de excesso, permitir a utilização do recurso adicional ou bloquear o acesso à rede, visando não prejudicar outros usuários.

A divisão em camadas do ATM (Fig. 2.29) prevê a existência de: (1) uma camada dependente do meio físico, a PHY; (2) uma camada que propicia a transmissão da informação em células ATM, a chamada camada ATM; e (3) uma camada que provê a interface com o usuário, a *A*TM *A*daptaion *L*ayer (AAL). A PHY é sub-dividida em *P*hysical *M*edia *D*ependent (PMD), dependente do meio físico adotado e em TC, responsável pela transmissão/recepção correta dos bits. A camada ATM utiliza-se de dois conceitos: o *V*irtual *C*hanel (VC), e o *V*irtual *P*ath (VP). O VC identifica um canal entre dois nós ATMs adjacentes e o VP pode ser visto como um conjunto de VCs utilizado para transferir a informação entre dois pontos da rede. A camada ATM deve fazer a multiplexação/demultiplexação necessária, iden-

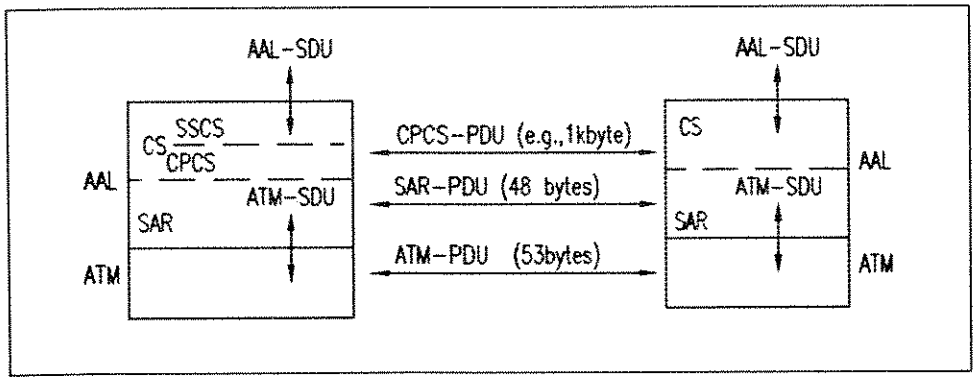


Figura 2.29: Sub-divisão em Camadas do ATM

tificando diferentes conexões pelo *V*irtual *C*hanel *I*dentifier (VCI); retirar/adicionar o cabeçário ATM antes/depois de encaminhar/receber a célula ao/do AAL; identificar o VC necessário; e fazer o controle de fluxo das células. São prevista três tipos de AALs. A AAL tipo 1 oferece ao usuário serviços classe A, onde é provida conexão, taxa constante de `bits` para um serviço isócrono. A AAL tipo 2 prevê os serviços da tipo 1 porém com taxa de `bits` variável. E a tipo 3/4, para classe C e D provê serviços não isócronos.

2.3.3 Entidades de Rede

A camada de rede, que possui como principal função o roteamento, pode ser orientada à conexão, como é o caso do padrão CCITT X.25, ou pode ser sem conexão, como por exemplo o protocolo ISO 8473. Nesta seção introduziremos estes dois protocolos além do IP.

X.25

O padrão X.25, largamente utilizado em redes públicas como padrão para serviço de rede com conexão, refere-se a três camadas inferiores da rede: a camada PHY, que é o padrão X.21 descrito na seção 2.3.1; a camada de enlace, que é o LAPB, que aproxima-se, segundo Andrew Tanenbaum, do LLC descrito na seção 2.3.2; a camada de rede: a X.25 *P*acket *L*ayer *P*rotocol (PLP), objetivo desta seção. Uma descrição mais detalhada do X.25 PLP pode ser encontrada em [Tan88] e a sua descrição formal em [CCI89e].

O X.25 provê duas formas de conexão: (1) *virtual calls*, e (2) *permanent virtual circuits*. Na primeira há o estabelecimento da conexão e de um canal virtual, que deixa de existir com o encerramento da conexão. Na segunda o canal virtual fica estabelecido independentemente de haver ou não dados para serem trafegados, como se fosse uma linha dedicada.

O endereçamento X.25 é estabelecido pelo CCITT através da recomendação X.121, composto de no máximo 14 dígitos decimais, nos moldes da numeração telefônica, onde os primeiros três dígitos indicam o país, o 4º indica a rede dentro do país e os outros 10 dígitos ficam para utilização interna em cada rede. Há países, como os EUA, que necessitam de alguns conjuntos de redes para cobrir todo o seu parque.

O formato da PDU X.25 PLP é dependente do tipo de PDU. Por exemplo os campos de endereçamento estão presentes apenas na PDU `Call Request`. Todas as PDUs possuem dois campos `Group` e `Channel`, como 12 `bits` no total, cujos conteúdos vão identificar o canal da conexão. A PDU de dados, `Data Packet` prevê o seqüenciamento e a fragmentação. As três fases de uma conexão X.25 estão esquematizadas na Fig. 2.30.

É previsto no X.25, como deve ser num protocolo para redes públicas, a tarifação.

IP

O IP é um protocolo de rede sem conexão proveninente de um projeto da *A*dvanced *R*esearch *P*rojects *A*gency (ARPA), agência do *D*epartment *o*f *D*efense (DOD), que tornou-se um padrão de fato para a camada de rede, amplamente difundido e o mais utilizado em todo o mundo. Uma intrudução ao IP pode ser encontrada em [Hed88, Tan88], e sua descrição formal em [Pos81c].

O IP foi projetado para proporcionar a troca de mensagens, os chamados datagramas, numa rede com qualquer topologia. Para alcançar o seu destino pode ser necessário o roteamento dos datagramas, como foi explicado na seção 2.2.2, além disto o IP proporciona a fragmentação e empacotamento dos datagramas, para transmissão via entidades de enlace com mensagens menores do que as suas (menor MTU).

O quadro do IP, representado na Fig. 2.31, prevê a existência dos seguinte campos:

`Version`: indica a versão do IP; atualmente utiliza-se a 4;

*I*nternet *H*eader *L*ength (IHL): tamanho do *header* em palavras de 32 `bits`;

`Type of Service`: permite entre outras coisas definir os seguintes parâmetros de qualidade de serviço:

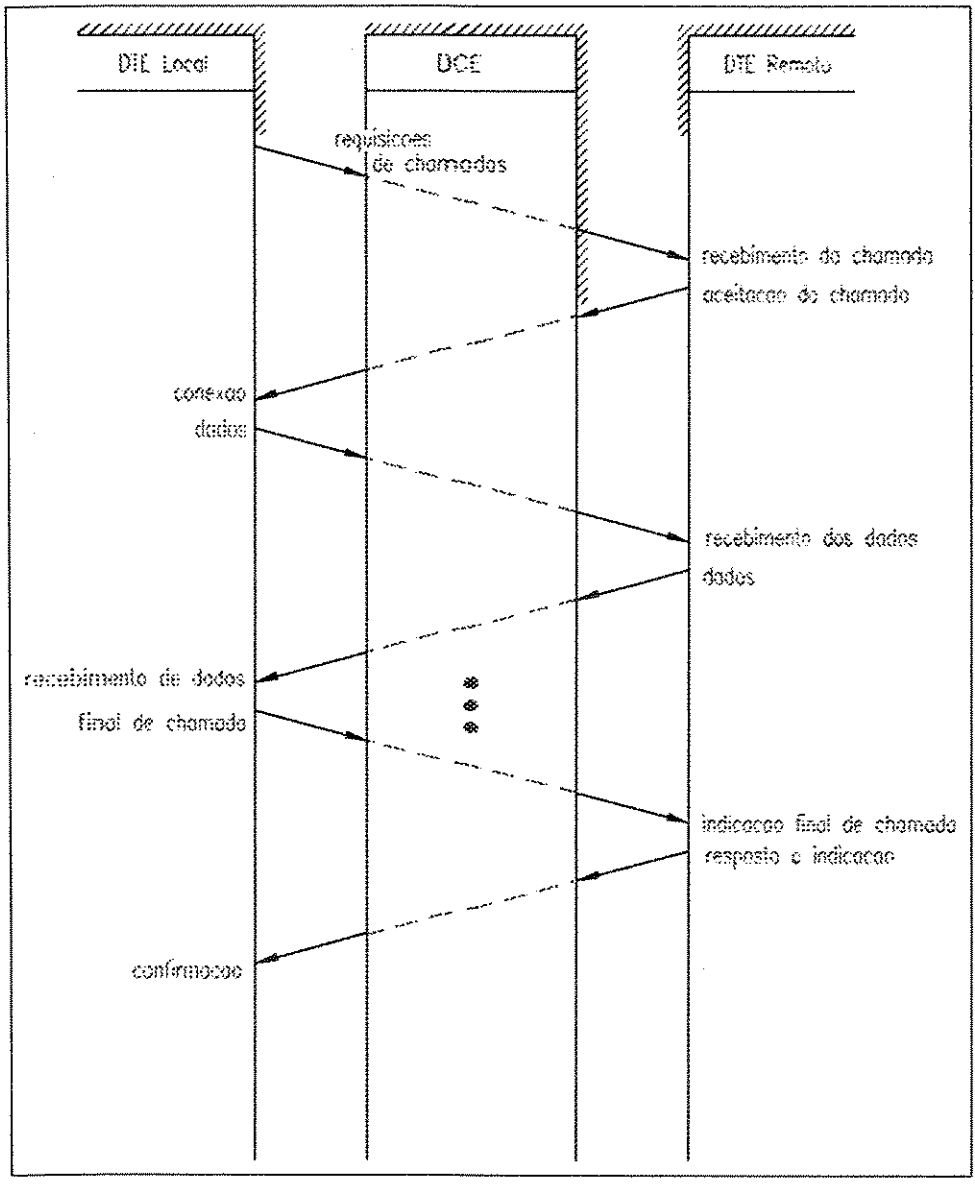


Figura 2.30: Fase de uma Conexão X.25

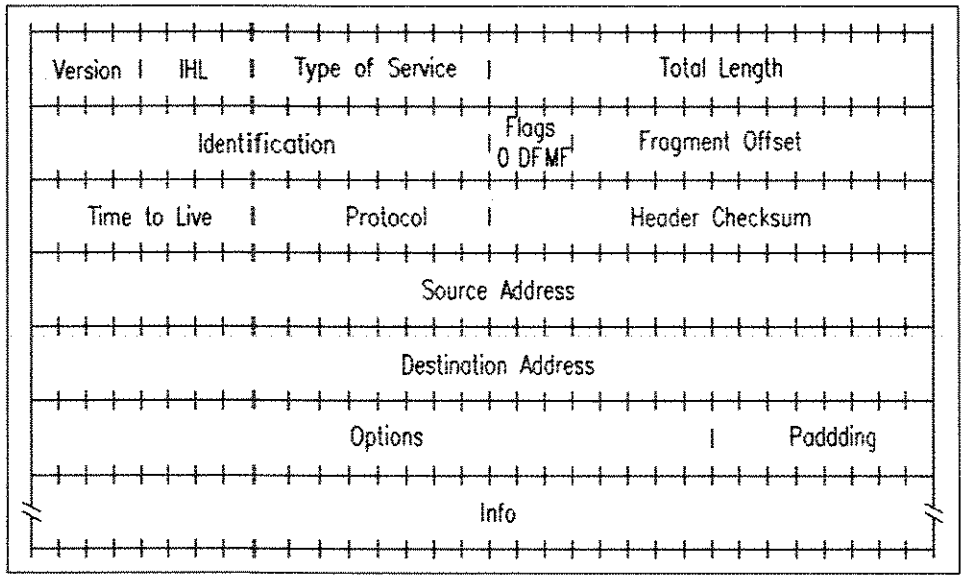


Figura 2.31: Quadro do IP

- atraso normal ou baixo;
- taxa normal ou alta; e
- confiabilidade normal ou alta;

Total Length: tamanho do datagrama, em octetos, no máximo 64 **kbits**;

Identification: identificação para o caso de ser necessária a fragmentação;

Flags: o 0 não é utilizado, os outros:

Don't Fragment (DF): quando %1 indica que não deve haver fragmentação, caso a entidade de enlace não permita o MTU desejado o pacote será descartado ou enviado por outra rota;

More Fragments (MF): quando %1 indica que há mais fragmentos da mensagem;

Fragment Offset: *offset* do fragmento, em 8 octetos;

Time to Live (TTL): por cada nó por onde o datagrama passa pela rede este campo é decrementado em pelo menos uma unidade, se o nó gastar mais do que 1 segundo no processamento deste datagrama o

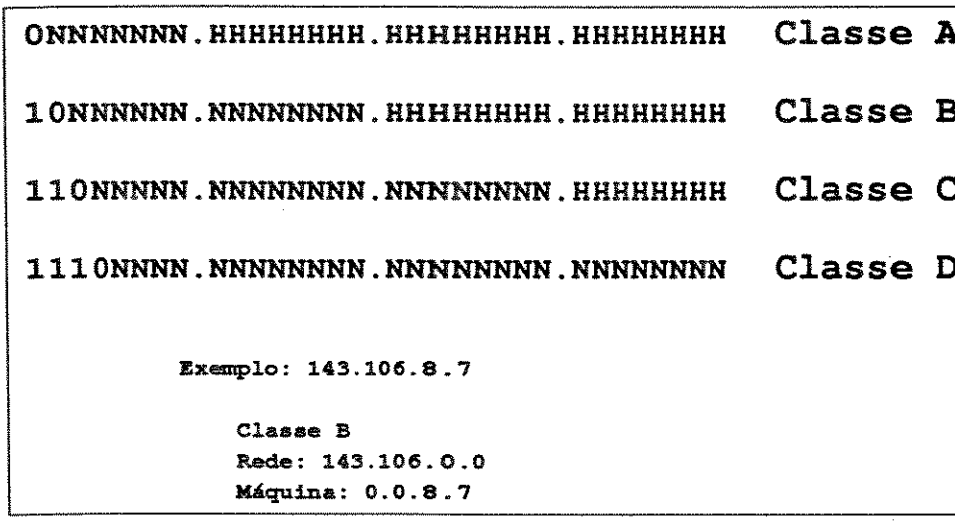


Figura 2.32: Endereço Internet

TTL deverá ser decrementado do número de segundos gasto. Assim garante-se um tempo de vida máximo de 255 s por cada datagrama na rede;

Protocol: indica o protocolo de transporte utilizado. Está definido em [Pos81a];

Header Checksum: o *check sum* do cabeçário, deve ser recalculado a cada enlace, posto que há campos que mudam a cada enlace;

Source Address: endereço Internet do remetente;

Destination Address: endereço Internet do destinatário;

Options: opções, utilizadas para roteamento a partir da origem, segurança, notificação de erro, *debug*, *time stamp*, etc;

Padding: sempre 0 para que o cabeçário tenha sempre um múltiplo de 32 **bits**; e

***Information* (Info):** mensagem da entidade de transporte.

O endereçamento do IP é feito por 4 *bytes*, geramente representados na forma decimal separados por pontos, que indicam a que rede o endereço pertence além de identificar a máquina (*host*) na rede (Fig. 2.32). Quando

o bit mais significativo é %0, um endereço classe A, indica que os próximos 7 bits indicam a rede e que os 24 últimos indicam a máquina. Quando os bits mais significativos são %10, um endereço classe B, os 14 próximos bits indicam a rede e os 16 últimos a máquina. Quando os bits mais significativos são %110, um endereço classe C, indica que os próximos 21 bits indicam a rede e que últimos 8 indicam a máquina. Finalmente quando os bits mais significativos são %1110, um endereço classe D, é utilizado experimentalmente para endereço *multicast*, i.e., que designa um grupo de máquinas que podem estar em sub-redes distintas. É comum a expansão do campo de rede, num endereço IP, a critério de uma administração local, nos limites da rede administrada, permitindo a criação de diversas sub-redes utilizando-se uma única classe IP.

Podemos dizer que uma rede Internet é um conjunto convexo de máquinas, já que entre duas máquinas quaisquer pertencente a mesma rede há sempre uma rota entre elas passando apenas por nós da própria rede. Utiliza-se este fato para fazer o roteamento por rede pela Internet. As informações de roteamento são distribuidas de uma forma automática por uma rede IP através de protocolos específicos. Maiores informações sobre o roteamento na Internet podem ser encotradas em [Nar89].

O IP prevê também a existência do *I*nternet *C*ontrol *M*essage *P*rotocol (ICMP) que envia mensagens indicando erros, redirecionamento, a existência do endereço, ou para controle de fluxo, etc. Este protocolo está definido em [Pos81b].

ISO 8473

O grande problema do IP é a utilização de endereçamento de tamanho fixo que hoje permite a existência de pouco menos de 4 bilhões de máquinas, teoricamente. Ocorre que pelo fato desses endereços serem distribuidos por rede, i.e., em conjuntos de pelo menos 256 endereços (uma classe C), é possível o seu esgotamento futuro. Para suprir esta deficiência do IP existe o padrão ISO 8473 que prevê os mesmos campos do IP, porém o endereçamento, assim como no X.25, possui um campo que indica o tamanho do endereço ([Tan88]).

2.3.4 Entidades de Transporte

A camada de transporte provê à entidade de sessão um serviço de comunicação livre das imperfeições das camadas inferiores. Falaremos nesta seção dos protocolos TCP, *U*ser *D*atagram *P*rotocol (UDP), ISO 8073, e ISO 8473. Os dois primeiros apresentam-se como um padrão de fato,

utilizado no mundo TCP/IP. Os dois últimos como a proposta com e sem conexão da ISO para redes de pacotes.

TCP

O TCP é o protocolo de transporte confiável, com conexão, utilizado nas redes TCP/IP. Ele propicia ao seu usário um canal de comunicação para transmissão de uma seqüencia ilimitada de bytes, fazendo, para isso, as necessárias fragmentações e empacotamento, além da validação da mensagem e confirmação de recebimento de cada fragmento. Introduções ao TCP podem ser encontradas em [Hed88, Ste90, Tan88], uma descrição mais detalhada em [Com91] e sua especificação formal em [Pos81d].

Antes de estabelecer uma conexão, o usuário que toma a iniciativa, usuário ativo, reserva uma porta TCP local, indicando em seguida o endereço IP da máquina remota, e a porta remota com a qual se deseja comunicar. A essa porta remota, já estará ligado, um usuário passivo remoto, em geral, um provedor de serviços, ou servidor. É comum a associação de uma porta pré-definida para um determinado serviço. Posto que uma conexão é individualizada pela quintupla (endereço IP local, porta local, protocolo utilizado (no caso TCP), endereço IP remoto e porta remota), é possível que uma porta esteja sendo simultaneamente utilizada por diversas conexões. Neste caso costuma-se criar uma instância do processo para cada conexão.

UDP

O *set* de protocolos TCP/IP provê também um protocolo de transporte sem conexão e *inseguro*: o UDP. Uma introdução ao UDP poderá ser encontrada também em [Hed88, Ste90, Tan88], uma descrição mais detalhada em [Com91] e sua especificação formal em [Pos80], que é um documento enxuto como o UDP.

O UDP é um protocolo sem confirmação, sem fragmentação e a validação dos dados é facultativa. O tamanho das UDPs SDUs, mensagens transmitidas pelo UDP, é limitado em cerca de 64`kbytes`. O leitor deverá estar se perguntando porque utiliza-se um protocolo deste quando dipõe-se do TCP. Qual seria a vantagem do UDP sobre o TCP? A simplicidade e conseqüente menor *overhead*, tornado-se adequado para aquelas aplicações que ou não necessitam de alta taxa de confiabilidade: como o *D*omain *N*ame *S*ervice (DNS), e o *N*etwork *T*ime *P*rotocol (NTP); ou que dispondo de camadas de enlaces confiáveis, necessitam de altas taxas, i.e., baixo *overhead*: como o *T*rivial *F*ile *T*ransfer *P*rotocol (TFTP), usado no *boot* de *diskless*, ou o *N*etwork *F*ile *S*ystem (NFS) responde a essa questão.

TPs

O ISO 8073 e o ISO 8473 especificam os protocolos com e sem conexão, respectivamente, da ISO. Uma introdução a esses protocolos pode ser encontrada em [Tan88, Ros90] e suas especificações formais em [ISO86b, ISO86a], respectivamente.

A ISO classifica os serviços de Transporte em 5 tipos: *T*ransport *P*rotocol (TP) 0, TP1, TP2, TP3, e TP4. O TP0 é utilizado sobre entidades de rede perfeitas, posto que ele apenas propicia o estabelecimento e liberação de conexão (quando utilizada a conexão) e a transferência das *T*ransport *PDUs* (T-PDUs).

O TP1 propicia além dos serviços do TP0 a recuperação de erros da entidade de rede.

O TP2 propicia além dos serviços do TP0 a multiplexação, i.e., é possível o envio de mais do que uma T-SDU numa unica conexão de rede.

O TP3 propicia simultaneamente os serviços do TP1 e do TP2.

Finalmente o TP4 é o protocolo mais completo e também o mais custoso dos cinco. Projetado para ser utilizado sobre uma camada de rede não confiável, sem conexão, onde pode haver perda de pacotes, atrasos, duplicação, erros, desordem, o TP4 provê a camada de sessão numa não rede confiável, assim é necessário fazer o sequenciamento, confirmação e a validação.

2.3.5 Entidades de Sessão

A camada de sessão provê além do mapeamento para a camada de Transporte os sincronismos necessários para transmissão da mensagem. O padrão TCP/IP possui como camada de sessão a *R*emote *P*rocedure *C*all (RPC) e a ISO padronizou o ISO 8327 como protocolo de sessão. Nesta seção intruduziremos os dois protocolos.

RPC

A RPC foi proposta pela Sun Microsystem, Inc. como padrão à comunidade Internet, e tornado público. Ele propicia o mapeamento automático nos *ports* (*T*ransport *SAP* (T-SAP)) da camada de transporte. O RPC proporciona aos seus usuários a utilização de rotinas remotas sem a utilização direta das funções de acesso ao serviço de transporte. O *E*xternal *D*ata *R*epresentation (XDR) é utilizado como sintaxe para representação dos dados no RPC. Uma introdução ao XDR e ao RPC pode ser encontrada em [Ste90], a descrição formal do XDR na [Mic87] e a descrição formal do RPC na [Mic88].

Um usuário da camada de transporte TCP/IP, TCP ou UDP, tipicamente indica não apenas o endereço IP do sistema com o servidor desejado, mas também o número do *port* relacionado ao serviço em questão. Alguns serviços possuem *ports* padronizados, e.g., o *login* remoto, o *S*imple *M*ail *T*ransfer *P*rotocol (SMTP), o *F*ile *T*ransfer *P*rotocol (FTP), mas um serviço não padronizado, digamos, uma aplicação desenvolvida localmente, exige que seja estabelecido previamente um *port* fixo reservado para este serviço. A utilização do RPC possibilita ao servidor requisitar um *port* livre a um *portmap* automaticamente, e um cliente RPC questiona o *portmap* quando quiser saber qual o *port* do serviço desejado.

A idéia do RPC é propiciar ao seu usuário uma interface com a rede similar a encontrada em funções/*procedures* em liguagens de programação. Desta forma uma função RPC ativada pelo chamador do serviço, encaminha os dados necessários via argumentos e aguarda a resposta via o retorno da função, ou também pelos seus argumentos. O RPC encaminha os dados ao servidor, ocupando-se de todo o processo necessário para utilizar os serviços de transporte. O servidor, que fica aguardando uma mensagem, faz o processamento necessário, devolvendo o resultado ao cliente RPC. Este, por sua vez, retorna o valor ao programa, que continua a execução normal. Assim o usuário, no caso uma aplicação, utiliza-se da comunicação pela rede como se estivesse acessando uma função local, com as seguintes ressalvas:

- é necessário tratar eventuais problemas na rede;
- não é possível a um processo remoto acessar a uma memória local ou a uma variável global;
- o desempenho de uma função local, via de regra, é superior a uma função RPC; e
- é necessário prover um mecanismo de autenticação.

ISO 8326/8327

O protocolo de Sessão da ISO propicia o fornecimento de muito mais serviços do que o RPC, sendo também muito mais complexo. Em muitos pontos foi buscada a compatibilidade com o serviço teletex padronizado pelo CCITT ([Tan88]). Suas principais características serão introduzidas nesta seção. Uma descrição mais completa e didática poderá ser encontrada em [Hal88]. [Ros90] faz um detalhamento maior. A especificação formal dos serviços pode ser encontrada em [ISO87c, ISO87d, ISO87e] e a especificação formal do protocolo em [ISO87f, ISO87g, ISO87h, CCI89e].

São os seguintes os serviços especificados pela ISO 8326 prestados pela camada de Sessão à de aplicação:

- estabelecimento de uma conexão com outra entidade de sessão possibilitando a troca de dados, e a finalização da conexão de uma maneira ordenada;
- gerência de *token*;
- controle do diálogo;
- gerência de atividade; e
- notificação de exceção.

O estabelecimento de conexão, tranferência de dados e finalização da conexão é propiciado via serviços da entidade de Transporte.

O número de aplicações onde a troca de informações é feita um sentido por vez, i.e., *half-duplex*, justifica a existência de um mecanismo na camada de Sessão que facilita este tipo de serviço. Tal mecanismo utiliza-se de *tokens* que são passados para o usuário da Sessão que deverá enviar os dados. É previsto também um esquema de prioridades.

O controle do diálogo permite a inserção de pontos de sincronismo numa mensagem longa que possibilitará em caso de falha da entidade de Transporte, o envio apenas do restante da mensagem ainda não enviada a partir do último ponto de sincronismo cuja recepção tenha sido confirmada. Assim poupam-se recursos de rede e tempo quando uma mensagem é interrompida por falha na rede, não sendo necessário a sua re-transmissão completa.

A gerência de atividade permite a interrupção de um diálogo, e posterior continuação a partir do ponto interrompido, para um envio de alguma mensagem de mais alta prioridade. Aqui é previsto também o serviço de quarentena, onde uma entidade de Sessão envia uma mensagem para a entidade par, mas que deverá aguardar uma ordem da entidade original para que tal mensagem seja encaminhada ao usuário da Sessão par.

A notificação de exceção propicia a informação de alguma falha à camada superior.

2.3.6 Entidade de Apresentação

Como já foi visto na seção 2.2.2, à camada de apresentação cabe a negociação de um contexto e a conversão dos dados de uma sintaxe abstrata para uma sintaxe concreta, além de prover à entidade de aplicação, os serviços da entidade de sessão, fazendo para isso um mapeamento direto.

Assim, para melhor entendimento dos serviços prestados pela entidade de Apresentação aqui introduzida, falaremos sobre o conceito de contexto, citando a *A*bstract *S*intax *N*otation *1* (ASN.1), como sintaxe abstrata e as BER como sintaxe concreta. Apresentaremos enfim os serviços da entidade de apresentação da ISO, descritos pelo documento ISO 8822 ([ISO88b, CCI89c]). O protocolo desta entidade é descrito pelo documento ISO 8823 ([ISO88c, CCI89c]). Descrições mais aprofundadas do que as apresentadas nesta seção e mais didáticas do que as apresentadas pelos documentos oficiais poderão ser encontradas em [Ros90, Ina90, Hal88].

Um contexto pode ser definido como um par sintaxe abstrata e sintaxe concreta, identidicado pelo ISO 8823 como *P*resentation *C*ontex *I*dentifier (PCI). No estabelecimento de uma conexão é negociado um conjunto de contextos, chamado *D*efined *C*ontext *S*et (DCS), que serão usados durante o diálogo. A existência de uma DCS possibilita a utilização de diversos contextos durante o diálogo. Podemos imaginar uma aplicação que faça transferência de arquivos texto, digamos *I*nternational *A*lphabet Number *5* (IA 5), e que permita a manipulação desses arquivos remotamente, e que a transferências do conteúdo desses arquivos se dê utilizando-se uma sintaxe concreta que faça compactação de dados, porém a tranferência dos atributos desses arquivos deve ser feitas sem compactação. Assim torna-se necessária a possibilidade de troca de contexto, cabendo à aplicação especificar qual a hora de utilizar-se um contexto com uma sintaxe concreta com compactação e sem.

A ASN.1 prevê a existência de tipos, i.e., unidades de informação, classificados em:

`UNIVERSAL`: tipo genérico, como boleano, inteiro, etc;

`CONTEXT_SPECIFIC`: relacionado a um contexto específico;

`APPLICATION`: comum a toda uma entidade de aplicação; e

`PRIVATE`: definido pelo usuário.

Os tipos universais podem ser primitivos, como `BOOLEAN`, `INTEGER`, `BITSTRING`, `OCTETSTRING`, `IA5String/GraphString`, `NULL` — nenhum tipo definido, e `ANY` — tipo a ser definido a posteriori; ou podem ser construidos, como `SEQUENCE` — como uma `structure` em C ou um `record` em pascal, `SEQUENCEOF` — um vetor de um determinado tipo, `SET` — como o `SEQUENCE`, porém não ordenado, `SETOF` — como o `SEQUENCEOF`, porém não ordenado, e o `CHOICE` um tipo escolhido de um conjuto de tipos definidos previamente. Em [Hal88] encontra-se uma descrição didática do ASN.1 e em [ISO87a, ISO87b] sua descrição formal.

Está padronizada uma sintaxe concreta para a ASN.1: as BER. Nelas cada elemento é representado por três campos, o primeiro formado por um octeto, indica o tipo do elemento; o segundo, formado por no mínimo dois octetos indica o tamanho de elemento em bytes; e o terceiro é o elemento propriamente dito. Desta forma as BER permitem a representação dos elementos descrito pela ASN.1 em uma seqüência de bytes na sintaxe concreta. Em [Hal88] encontra-se uma descrição didática das BER e em [ISO87i, ISO87j] sua descrição formal.

A camada de sessão da ISO é dividida em três unidades funcionais:

- *kernel*;
- *context management*; e
- *context restoration*.

A primeira unidade, de uso e implementação obrigatórios, provê o estabelecimento de conexão, a transferência normal de dados, a finalização organizada da conexão, o aborto da conexão pelo usuário, e o aborto pelo provedor. A segunda unidade, cuja utilização é negociada previamente com o usuário, provê a adição e remoção de contextos. E a terceira unidade, cuja utilização é negociada previamente com o usuário, e que requer o uso do *context management*, provê a restauração de contextos.

2.3.7 Entidades de Aplicação

A camada de apresentação difere-se das outras por não prover serviços às camadas superiores, posto que essas inexistem. Existem aqui os chamados *A*plication *S*ervice *E*lement*s* (ASEs), que correspondem a entidades que agrupam funções que provêem determinado recurso de interconexão em rede a uma instância de alguma entidade de aplicação. Cada instância de um ASE comunica-se com uma única instância par, utilizandos-se apenas uma associação (conexão), i.e. há apenas um canal virtual entre duas instâncias ASEs.

Os ASEs são divididos em *C*ommon *A*plication *S*ervice *E*lement*s* (CASEs) e em *S*pecific *A*plication *S*ervice *E*lement*s* (SASEs), esquematizados na Fig. 2.33. Os primeiros agrupam aquelas funções que são utilizadas por diversos SASEs, já os segundos contêm as funções de cada serviço específico. O ISO/IEC DIS 9545 ([II89]) detalha o OSI-RM, com relação à camada de aplicação. Um *A*plication *P*rocess (AP) utiliza-se de uma ou várias SASE que utilizam as CASEs.

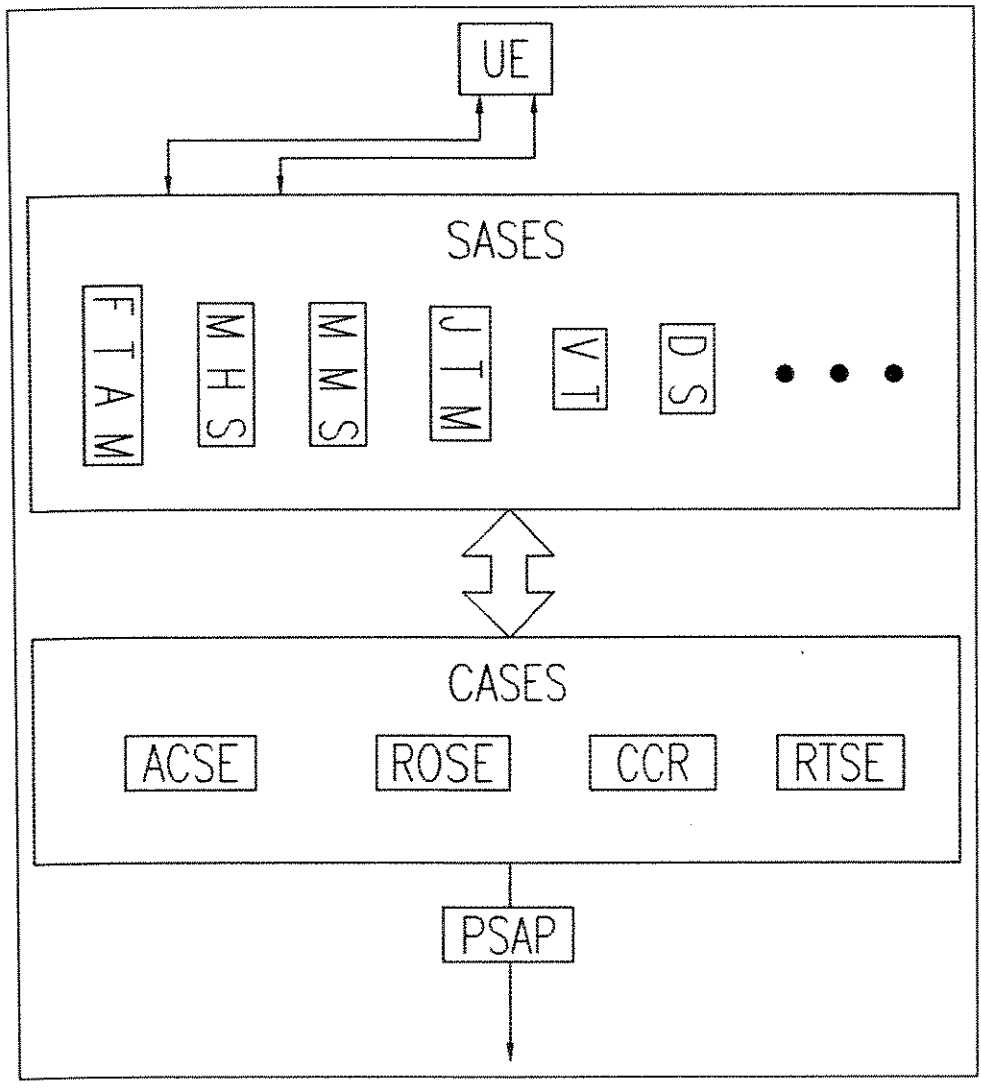


Figura 2.33: ASEs

CASEs

Os CASEs definidos pela ISO são *A*ssociation *C*ontrol *S*ervice *E*lement (ACSE), *R*emote *O*perations *S*ervice *E*lement (ROSE), *C*ommitment, *C*oncurrency, and *R*ecovery (CCR), e *R*eliable *T*ransfer *S*ervice *E*lement (RTSE) ([Hal88, Men90]). Listaremos as principais funções de cada um a seguir.

O ACSE apresenta-se como o núcleo básico dos CASEs, estando portanto sempre presente em qualquer aplicação. [Hal88, Men90] resume os seus serviços e protocolos, [Ros90] apresenta-os mais detalhadamente, o ISO/IEC 8649 ([ISO88e]) especifica formalmente os seus serviços e a ISO/IEC 8650 ([ISO88d]) o seu protocolo.

Um SASE iniciador estabelece uma associação (conexão) com um SASE respondedor via ACSE, depois deste estabelecimento o ACSE volta a agir apenas para interromper ou encerrar a associação. O processo de *binding* de uma associação é estabelecido em duas partes, na primeira o SASE, após consultar a SASE *D*irectory *S*ervice (DS), indica ao ACSE com qual entidade é desejada a associação; e a segunda corresponde ao estabelecimento da associação com a entidade desejada, via ACSE respondedor.

O ROSE tem como objetivo iniciar uma operação e receber o resultado dessa operação numa e de uma SASE remota. Uma explanação didática sobre o ROSE pode ser encontrada em [Ros90] e a especificação formal dos seus serviços e protocolo em [ISO88k, ISO88l, CCI89c].

Por suportar RPC (2.3.5), utilizado pela maioria das aplicações, Marshall T. Rose ([Ros90]) atribui o sucesso do ROSE. O ROSE que requisita uma operação é chamado de evocador e o que responde a essa requisição de empreendedor (*performer*). É possível qualquer combinação com os termos iniciador e provedor do ACSE. Assim quando o iniciador é o evocador diz-se uma associação classe 1, quando o respondedor é o empreendedor diz-se uma associação classe 2 e finalmente quando o iniciador e o respondedor são evocadores e empreendedores diz-se uma associação classe 3. O ROSE suporta *linked operations*, i.e., a passagem no nome da operação desejada — da função — como argumento da evocação.

O CCR é utilizado por SASEs que necessitam de uma ação átoma, i.e., não interrompível, para comunicar-se com um ou mais SASE. Uma ação átoma é definida como (1) uma seqüencia de operações que não é interferida por nenhum SASE que não faça parte do processo; e (2) uma operação completada com sucesso por *todos* os SASEs atuantes no processo ou então todas as operações envolvidas terão seus efeitos cancelados, voltando os dados aos seus estados iniciais. Um descrição didática do CASE CCR pode ser encontrada em [Hal88].

Um AP iniciador de uma ação átoma via CCR é referido como *mestre*,

ou *superior*, e as outras aplicações envolvidas na transação são chamados de *escravos* ou *subordinados*. Os dados afetados pela ação são chamados *bound data* que no inicio da operação estão no *estado inicial* e no final da operação, se bem suscedida, estarão no *estado final*. Cabe ao CCR mestre autorizar a passagem para o *estado final* do *bound data* local e dos escravos em caso de sucesso da transação ou então o retorno ao *estado inicial* no caso de alguma falha com algum AP.

O RTSE propicia a transferência de grandes volumes de dados entre APs. O RTSE provê ao seu usuário as facilidades da camada de sessão, fornecendo um serviço de fácil utilização e permitindo a transferência de dados tipados, tipicamente tipos ASN.1, não apenas bytes. O RTSE utiliza-se dos serviços da ACSE. Uma descrição didátida do seu serviço e do seu protocolo pode ser encontada em [Ros90] e a descição forma dos seus serviços e protocolo em [ISO88i, ISO88j, CCI89d].

SASEs

Os SASEs mais comuns definidos pela ISO são: *F*ile *T*ransfer, *A*ccess, and *M*anagement (FTAM), *M*essage *H*andling *S*ystem (MHS), *M*anufacturing *M*essage *S*tandard (MMS), *J*ob *T*ransfer and *M*anagement (JTM), *V*irtual *T*erminal (VT), e DS. Serão abordados introdutoriamente a seguir.

FTAM

Prover sistemas de arquivos remotos, onde acessa-se a um disco de um servidor de arquivos como se fosse um disco local, é certamente o serviço mais requisitado em LANs. Redes proprietárias como a NetWare, PathWork, Lan Manager, DECNet, *S*ystem *N*etwork *A*rchitecture (SNA) e tantas outras oferecem esse serviço. O NFS é o padrão de fato no mundo TCP/IP e também aqui a ISO possui seu padrão: o FTAM. Uma introdução ao FTAM pode ser encontrada em [Hal88], descrições mais detalhadas em [Tan88, Ros90]e a sua descrição formal em [ISO88a].

O sistema de arquivos FTAM é amorfo, não hierarquizado, ao contrário do do Unix, por exemplo, que prevê o agrupamento hirárquico dos arquivos em diretórios, podendo o nome do arquivo indicar a sua situação na estrutura de diretórios. No FTAM garante-se apenas a uniqüidade do nome do arquivo. Por sua vez um arquivo não é apenas uma seqüência de bytes como no Unix, mas possui uma divisão interna hierárquica, composta pelas chamadas *F*ile *A*ccess *D*ata *U*nits (FADUs). Esta característica possibilita o seu mapeamento em qualquer sistema de arquivos. Tal padronização fica fora do escopo da especificação do FTAM. São previstas várias operações diretamente nas FADUs. Os arquivos FTAM possuem uma série de atributos, tais como o nome, as operações permitidas, o

controle de acesso, datas de criação, modificação, leitura, e modificação dos artributos, dono, chave de criptografia, tipo de arquivo, qualificação legal do arquivo, etc. O mapeamento dos arquivos FTAM e seus atributos em um determinado sistema de arquivo fica a critério de cada implementação.

MHS

O MHS é seguramente o serviço mais utilizado em MANs e WANs, sendo ou *I*nter*p*ersonal *M*essage (IPM), o correio eletrônico, o serviço do MHS mais popular. A seguir introduziremos o MHS, exemplificando os serviços que podem utilizá-lo. Apresentaremos também a sua estrutura. O MHS, chamado pela ISO de *M*essage *O*riented *I*nterchange *S*ystems (MOTIS), foi padronizado pelo CCITT através da recomendação X.400 ([CCI89b] equivalente ao padrão [ISO88h]). Descrições mais didáticas podem ser encontradas em [UIPF, Ros90].

O princípio básico do MHS é a transmissão de mensagens via *store and forward*, onde a mensagem, partindo da sua origem, atinge o destino, via uma série de sistemas intermediários, que a transmite não necessariamente *on line*. O MHS pode ser visto como a versão eletrônica dos Correios, que propicia, não apenas o envio de cartas, mas também de encomendas, fax, telegramas, etc. Também o MHS não limita-se apenas ao IMP. O seu uso pode ser para transferência de qualquer tipo de mensagem no esquema *store and forward*. É prevista, por exemplo, sua utilização para *E*lectronic *F*unds *T*ransfer (EFT), i.e., para a tranferência de fundo entre, por exemplo, instituições financeiras, e para *E*lectronic *D*ata *I*nterchange (EDI), i.e., para a transferência de documentos entre, por exemplo, empresas.

O modelo funcional do MHS prevê a existência de três ambientes: o ambiente do usuário, que pode ser uma pessoa, ou um processo, diretamente conectado ao MHS ou via outro sistema de comunicação; o ambiente do MHS; e o ambiente do MTS. O último é responsável pela transferência das mensagens que trafegam entre *M*essage *T*tranfer *A*gents (MTAs) até os seus destinos. O ambiente MHS é composto pelo ambiente MTA e pelos *U*ser *A*gents (UAs), responsável pela interface entre o usuário e o MTS. É possivel também a existêcia o ambiente MHS de unidades de armazenagem de mensagens, entre um UA e o MTS. É possivel também a interligação do MHS com redes de comunicação não eletrônicas, como os Correios, via dispositivos apropriados, os *P*hysical *D*elivery *A*ccess *U*nit (PDAUs).

MMS

O MMS é um SASE para utilização num ambiente industrial totalmente automatizado, provendo um protocolo padronizado entre os níveis de processo e atuadores/sensores (veja 2.1.4). Em [Hal88] encontramos uma introdução ao MMS e em [EIA86] a sua especificação formal.

O MMS provê a comunicação de um controlador de célula de manufatura (Fig. 2.5) com os controladores de chão de fábrica, como *A*uto-*G*uide

*V*ehicles (AGVs), CLPs, CNCs, etc. Entre os níveis de processo e de sistema a comunicação pode ser feita via FTAM por exemplo. Assim o controlador de uma célula de manufatura recebe, via transferência de arquivo, a programação, a nível de sistema, e executa essa programação controlando e monitorando os controladores da sua célula via MMS. O MMS provê o estabelecimento de uma associação com um determinado controlador, a leitura pelo controlador de um arquivo de dados com as instruções de operação pertinentes, o carregamento de um programa no controlador, o controle remoto de um controlador, leitura e modificação de variáveis associadas ao programa do controlador, a requisição ao controlador dos serviços por ele suportado.

JTM

O JTM SASE propicia a execução e manipulação de processos remotamente. Aqui, assim como no FTAM, a integração com os sistemas reais fica fora do escopo desse padrão, cabendo-lhe apenas prover os mecanismos para JTM sem entrar no mérito dos processo propiamente dito. Em [Hal88] encontramos uma boa introdução ao JTM e em [ISO85] a sua especificação formal. Especificam-se aqui vários *agentes*: o agente iniciador, que inicia o processo; o agente executor, que propicia a execução do processo; o agente originador, que propicia fornece os dados necessários para a execução do processo; o agente monitor, que fornece um relatório ao agente sorvedor. Na prática muitos desses agentes estarão nos mesmos sistemas.

VT

Um dos serviços muito utilizados em redes é o *login* remoto, onde acessamos um computador remotamente como se estivéssemos num dos seus terminais. Prover esse serviço requer resolver alguns problemas como: a falta de padronização dos terminais e a necessidade de simular que o terminal remoto seja local, para o perfeito funcionamento dos *software*. Os terminais alfanuméricos geralmente aceitam receber seqüencias especiais de bytes que manipulam a tela. O problema é que, apesar dos esforços da ANSI, não há uma padronização de fato para essas seqüências. No Unix utiliza-se uma base de dados que indica para cada terminal conhecido quais as seqüências de caracteres especiais utilizadas. Isso permite que as aplicações comportem-se corretamente apenas sabendo qual o terminal do usuário. Também no Unix o segundo problema foi resolvido criando-se terminais virtuais, que permitem as aplicações sejam escritas como se fossem ser utilizadas em terminais locais, cabendo aos *servidores* de *logins* remotos a configuração correta desses terminas. Em [Ste90] encontramos um capítulo dedicado a discussão desse problema, implementando um exemplo de servidor de *logins* remoto. Em [ISO86c] encontramos a especificação formal do VT da ISO.

XWindow

Um padrão de fato e vastamente utilizado no mundo TCP/IP é o X Window System[3], um sistema de janelas orientado à rede, que propicia a execução remota de aplicativos gráficos via IP. Desenvolvido pelo *M*assachusetts *I*nstitute of *T*echnology (MIT), o X Window System preve a existência de um servidor de janelas, que estaria rodando na máquina local, podendo ser até um terminal dedicado - um *X Terminal*, os aplicativos utilizam-se de primitivas da biblioteca *Xlib* para enviar e receber informações para o servidor, informções como o botão direito do *mouse* foi precionado, ou trace uma reta entre dois determinados pontos. Sobre o *XWindow* foram contruidas algumas bibliotecas, chamadas *tool kits*, que permitem a utilização pelas aplicações de chamadas de mais alto nível, como faça uma janela de tal tamanho, ou coloque um determinado título em determinado lugar. A *tool kit* Motif[4] além de propocionar uma interface mais fácil com a *Xlib* possui o padrão *look and feel* Motif da *O*pen *S*oftware *F*oundation (OSF). A *tool kit* XView[5] possui o padrão *look and feel* OPEN LOOK[6], da Sun Microsystems, sendo de domínio público. [Sch87] é a *R*equest *f*or *C*omments (RFC), documento oficial da Internet, sobre o X Window System e em [X/O89] encontramos sua documentação completa.

DS

Como vimos na seção 2.2.2 a utilização de um serviço de uma determinada camada é feita via SAP dessa camada. Uma aplicação ao requisitar um serviço deverá passar então o endereço completo formado por:

$$\text{endereço de AP} = \text{P-SAP} + \text{S-SAP} + \text{T-SAP} + \text{N-SAP}$$

mas tanto para os usuários programas quanto para os usuários humanos é mais fácil guardar nomes simbólicos do que códigos como *P*resentation *SAP* (P-SAP) ou *N*etwork *SAP* (N-SAP). Visando converter nomes simbólicos em endereços de aplicação é previsto o DS, o seu equivalente no mundo TCP/IP é o DNS, que não prevê tantos recursos quanto o DS. Não abordaremos o DNS nesse texto. O leitor interessado poderá encontrar informações em [Sta87, Lot87, Moc87a, Moc87b]. Um introdução mais detalhada do que a nossa sobre o DS pode ser encontrada em [Hal88], uma descrição mais detalhada e didática poderá ser achada em [Ros90] e sua descrição formal em [ISO88m].

Apesar do mapeamento de nomes simbólicos em endereços de aplicação ser a principal função do DS, não é a única. O DS pode prover a seu usuário informações com respeito à rede, e.g., ao MHS poderá prover o mapeamento

[3] X Window System é marca registrada do MIT.

[4] Motif é marca registrada da Open Software Foundation, Inc..

[5] XView é marca registrada da Sun Microsystems, Inc..

[6] OPEN LOOK é marca registrada da AT&T.

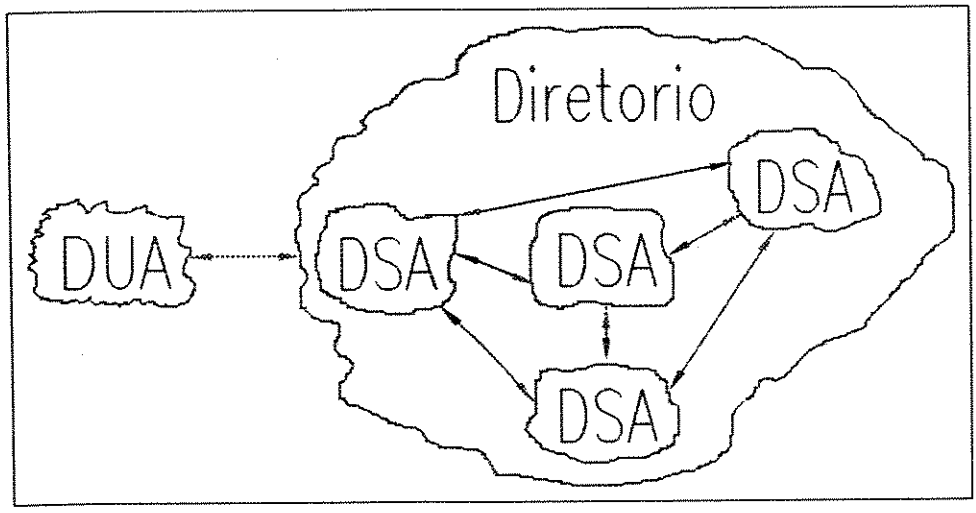


Figura 2.34: Modelo Funcional do DS

de nomes simbólicos em endereços de correio eletrônico, poderá informar a um usuário o endereço eletrônico de alguém chamado, digamos, Gorgonio que trabalha na UNICAMP. São apresentados a seguir quatro modelos de DS, que devem ser vistos como formas diferentes para descrição do mesmo serviço: o modelo de informação, o modelo funcional, o modelo organizacional e finalmente o modelo de segurança.

O modelo de informação descreve a base de dados do DS: a *D*irectory *I*nformation *B*ase (DIB). A DIB é formada por unidades de informação: as ***entradas***. Cada entrada é composta por atributos, que por sua vez são formados por um tipo e por valores. O tipo do atributo descreve o tipo da informação ali contida, e.g., caracteres, inteiro, e a sua sintaxe, descrita em ASN.1. Um atributo pode ter diversos valores, e, sendo assim, um deles é chamado de valor destinto. Um dos atributos da entrada, chamado `objectClass`, determina a espécie do objeto contido na entrada, e.g., se é uma pessoa, uma empresa, um número de telefone. Aqui é previsto o conceito de herança de classe e de especialização, utilizado em *O*bject *O*riente *D*esign (OOD) ([TLX90]). As entradas relacionam-se hierarquicamente via uma estrutura chamada *D*irectory *I*nformation *T*ree (DIT), para tanto há uma atributo chamado *R*elative *D*istinguished *N*ame (RDN) que identifica de uma forma não ambigua uma entrada numa DIT. É prevista uma entrada chamada ***alias*** que aponta para uma outra entrada dentro da DIT.

O modelo funcional descreve as interações dentro do diretório. A DIB não é uma base de dados centralizada; há agentes chamados *D*irectory *S*ervice *A*gent (DSA), que descentralizadamente, a controlam. Um *D*irectory *U*ser *A*gent (DUA) querendo saber uma determinada informação do DS consulta o DSA mais próximo diretamente; o DSA não possuindo essa informação indica ao DUA qual o DSA que a possui ou vai buscar a informação nesse DSA, repasando-a em seguida para o DUA.

O modelo organizacional define como a DIB está dividida entre os DSAs, definindo o *D*irectory *M*anagement *D*omain (DMD) como o domínio responsável por determinada parte da DIB. Um DMD pode ter uma ou mais DSA, que abrigam a parte da DIB sobre aquela administração; pode conter algumas DUAs e como as diversar DSAs devem ser vistas externamente ao DMD.

O modelo de segurança descreve o serviço em termos de segurança e autenticação. São prevista três formas de autenticação de um usuário: nenhuma, leve ou pesada. A leve é baseada no fornecimento de uma senha pelo usuário requisitor de determinado serviço e a pesada é baseada e criptografia por chave pública.

CMISE

A gerência de rede vista hoje como uma tarefa exercida a nível de camada de aplicação torna-se de fundamental importância para a gerência de redes geograficamente distribuidas e com diversidade de equipamentos. Uma grande experiência prática foi adquirida na Internet que levou a IAB a recomendar em 1988 ([Cer88]) direções técnicas para resolver o problema, o que resultou na criação de três grupos de trabalho a nível de *I*nternet *E*ngineering *T*ask *F*orce (IETF): (1) o grupo do *M*anagement *I*nformation *B*ase (MIB), (2) o grupo do *S*imple *N*etwork *M*anager *P*rotocol (SNMP), e (3) o grupo do *CM*IP *O*ver *T*CP/IP (CMOT). O grupo (1) teve como objetivo a definição dos objetos contidos na MIB e da própria MIB. A MIB contém todas as informações necessárias à gerência da rede descrita utilizando ASN.1/BER, introduzidas na seção 2.3.6, documentando o resultado dos seus trabalhos em [RM88, MR88]. O grupo (2) aprimorou um antigo protocolo de gerência existente na Ineternet, o *S*imple *G*ateway *M*anager *P*rotocol (SGMP) definindo o SNMP, um protocolo simplificado para gerência de rede e amplamente utilizado no mundo TCP/IP. Sua especificação encontra-se em [CFSD88]. Finalmente, o grupo (3), o grupo de trabalho *Netman* buscou a implementação do *C*ommon *M*anagement *I*nformation *P*rotocol (CMIP), protocolo de gerência de rede da ISO sobre o TCP/IP, o CMOT. Sua especificação formal encontra-se em [WB89]. A seguir faremos uma introdução ao CMOT. O leitor interessado poderá encontrar a especificação formal dos *C*ommon *M*anagement *I*nformation *S*ervice (CMIS)/CMIP em [ISO88f]/[ISO88g].

O CMIP prevê a existência de dois tipos de processo: os gerentes e os agentes. O gerente é o processo que controla a gerência monitorando e agindo sobre os agentes. Os agentes são os porcessos que interagem, no objeto gerenciado, com os agentes. Os agentes e o gerente trocam informações via CMIP. O `proxy`, é um agente especial que permite a interface com objetos que não falam o CMIP, como por exemplo pontes e modens, que não possuem a camada e aplicação, ou por exemplo objetos que utilizam outro protocolo de gerência de rede, como por exemplo o SNMP. O gerente utiliza o CMIP para solicitar a um agente a realização de alguma operação no objeto gerenciado. Essa operação pode ser a leitura ou a modificação de um parâmetro, a realização de alguma ação do objeto gerenciado, como por exemplo a realização de auto-teste. É previsto também ao agente enviar assincronamente informações ao gerente, via eventos ou interrupções (*traps*).

Há três modelos para o CMIP: (1) o modelo organizacional, (2) o modelo funcional e (3) o modelo informativo. O modelo organizacional introduz o conceito e domínios administrativos, onde um domínio é um sub-conjunto da rede sob uma mesma administração. O modelo funcional descreve 5 áreas específicas na gerência de rede: gerência de falhas, gerência de configuração, gerência de desempenho, gerência de contabilidade e gerência de segurança. O modelo informativo considera que todas as informações relevantes para a gerência de rede encontram-se numa MIB.

Para prover os serviços de gerência de rede é previsto um SASE: o *C*ommon *M*anagement *I*nformation *S*ervice *E*lement (CMISE). O CMISE utiliza-se do ROSE e do ACSE, além de uma parte dos serviços do DS.

2.4 Tendências Futuras

Algumas questões são levantadas aqui com respeito às tendências futuras da telecomunicação, em especial envolvendo as redes de computadores e a telefonia.

2.4.1 RDSI

O CCITT, antevendo a necessidade de prover mecanismos de comunicação mais eficientes e baratos, vem aprimorando a especificação de uma rede digital de comunicação que tem como meta substituir a atual rede telefônica, provendo uma variedade muito grande de serviços: a RDSI. O projeto prevê a gradual digitalização da rede telefônica, visando suportar a transmissão de informações digitais criando "uma rede totalmente digital, composta de máquinas de comutação (ou nós) inteligentes, que podem

cursar vários serviços simultaneamente e aos quais os usuários tem acesso através de algumas interfaces padronizadas" ([CCI89g][Araújo90]). A idéia é prover os usuários com uma rede com interface padronizada, como ocorre hoje na rede elétrica, onde liga-se qualquer tipo de aparelho via uma das interfaces padronizadas: tomadas de dois polos ou tomadas de três polos, por exemplo, requisitando determinado serviço: o fornecimento de energia a uma diferença de potencial determinada de 127 V ou 220 V, por exemplo. Assim o usuário ligaria o seu TE; que pode ser um telefone, um computador, um terminal gráfico, uma televisão, um video-fone, um fax, entre outros; à rede para transmissão e ou recepção de informações.

A implantação da RDSI prevê uma primeira fase onde seriam oferecidos serviços à taxa de até 2 Mbits/s, chamada de *RDSI-Faixa Estreita* (RDSI-FE) e numa segunda fase seriam incorporados à RDSI os serviços da RDSI-FL com taxas de até 600 Mbits/s. A RDSI-FE prevê dois tipos de acesso ao usuário: (1) o acesso básico, e (2) o acesso primário. O acesso básico oferece dois canais de 64 kbits/s e um de 16 kbits/s. Descrevemos a sua camada de enlace nas seção 2.3.2. O acesso primário oferece taxas de até 2 Mbits/s.

A RDSI-FL prevê a utilização de redes ATMs. Esta utilização, que já começa a ocorrer ([Wu93, BSF+93]), prevê uma migração gradual para uma rede ATM pura utilizando-se redes síncronas (*Syncronous Transfer Mode* (STM)) para transmissão de células ATMs. Ao contrário da previsão, a RDSI-FL/ATM parece ganhar adeptos mais rapidamente do que a RDSI-FE sendo impulsionada pela tendência de utilização de redes locais com altas taxas e pela necessidade de interligação de redes locais. Um indicativo da ascensão da ATM é o custo previsto para 1996 de US$500 por interface ATM de 150 Mbits/s, o ([LMT93]).

Uma série de serviços são previstos pela RDSI, mas antes de descrevê-los explicaremos a classificação adotada pelo CCITT ([CCI89h, RP89]). Os serviços são classificado pelo CCITT como sendo de suporte ou tele-serviço. Os serviços de suporte são aqueles providos a nível das três primeiras camadas do OSI-RM, transformando a RDSI numa rede de transporte, onde a informação a ser transmitida fica a encargo dos usuários. Os telesserviços são aqueles serviços providos pela RDSI a nível das sete camadas do OSI-RM, assim a informação trafegada pela rede passa a ser *entendida* pela rede. Há também a classificação em serviços básicos e suplementares. Os serviços básicos são aqueles cuja ausência implica na ausência da comunicação (um exemplo: um circuito virtual para a telefonia). Os serviços suplementares são aqueles cuja existência fornece apenas uma facilidade para o usuário, não sendo imprescindível; alguns exemplos de serviços suplementares seriam: discagem abreviada, apresentação do número de origem e destino, grupo fechado de usuários, tranferência de chamada, transferência de informação de usuário a usuário, discagem direta

a ramal, indicação da tarifação ao usuário, chamada em espera, suspensão de chamada e conversão do protocolos.

A RDSI-FE prevê os seguintes serviços de suporte: serviços em modo circuito a 64kbits/s, 384kbits/s, 1536kbits/s, 1920kbits/s e a 2 x 64kbits/s, serviços em modo pacote com/sem conexão a 16kbits/s e 64kbits/s, serviços de sinalização usuário a usuário a 16kbits/s, e serviço de modo circuito para transferência de voz. E os seguintes telesserviços: telefonia, telex, facsímile, modo misto, video texto e MHS.

A RDSI-FL prevê os serviços interativos ou por distribuição. Os interativos são aqueles em que o usuário interfere no fluxo de informação, os por distribuição o usuário pode optar por receber ou não a informação, mas não pode interferir na sua distribuição. Os serviços interativos podem ser de conversação, de mensagem ou de recuperação. Nos serviços de conversação os usuários trocam informação entre si, alguns exemplos destes serviços são video-telefonia, video-conferência, video-vigilância, serviço de informação por video/audio, programas de audio multi-canal, transmissão de dados digitais em alta taxa, transferência de arquivos de grande volume, tele-ações a alta velocidade, telefax a alta velocidade, serviço de comunicação com imagens de alta resolução, serviço de troca de documentos. Nos serviços de mensagem os usuários trocam informações entre si via mecanismo *store and forward*, alguns exemplos de serviços de mensagem são serviço de correio eletrônico com video e correio eletrônico com documentos. Nos serviços de recuperação o usuário pode recuperar informações armazenadas em algum centro de informações; alguns exemplos são *videotex* — educação e treinamento remoto, telecompras — video remoto, busca de imagens de auto definição, busca de documentos, *telesoftware* . Os serviços de distribuição podem ser sem controle do usuário ou com o controle do usuário. No primeiro caso a informação é transmitida continuamente independente do usuário estar recebendo-a ou não, como por exemplo televisão a cabo, televisão de alta definição, distribuição de jornais e revista, distribuição de dados irrestritamente. Nos serviços de distribuição com o controle do usuário as informações são transmitidas ciclicamente de forma que o usuário pode controlar o início e a ordem da apresentação, como distribuição de audio/video repetidamente.

2.4.2 Os Protocolos da ISO

A literatura sempre aponta os protocolos da ISO como o padrão a ser seguido para redes de computadores, porém as suas qualidades são também os seus pontos baixos. Os protocolos da ISO propiciam uma vasta gama de serviços para cada camada, tornando-se bastante completos, e também complexos. A sua complexidade dificulta a sua implementação e

torna a comunicação mais *cara*, com o considerável aumento do *overhead*. Lembramos que o que realmente importa para os usuários são os padrões *de facto*. O padrão *de facto* hoje para redes de computadores é o TCP/IP. Marshall Rose escreve: "*Before users will transit from the Internet suit of protocols to the OSI suite, they will have to see added value in OSI applications. Users are interested in services, not protocols*" ([Ros90]). A utilização de redes públicas baseadas no TCP/IP já é bastante difundida e tende a ser cada vez mais popular ([Lit93]). Dado que a quantidade de endereços Internet é limitada (4 bytes) e que trata-se de uma rede que cresce a taxa de 7% ao ano, discute-se a substituição do IP por algum outro protocolo de rede, que pode ser o *C*onnection-*L*ess *N*etwork *P*rotocol (CLNP) da ISO. Os protocolos de sessão, apresentação e aplicação da ISO aumentariam em muito a versatilidade e confiabilidade da rede, dado o acréscimo de serviços em relação ao TCP/IP, seu *overhead* poderá ser compensado pela crescente largura de faixa das redes de computadores, mas do seu sucesso dependerá o investimento em produtos que propiciem mais serviços para os usuários finais.

Capítulo 3

A MAC DA RALFO

Neste capítulo exibimos a RALFO dando uma maior ênfase à sub-camada MAC. Introduzimos à RALFO, apresentando os serviços cobertos por esta rede e suas principais características, além do método de acesso ao meio. Expomos a subdivisão em camadas e sub-camadas, resumindo a camada de APL, a sub-camada LLC/voz e a sub-camada LLC/dados. Apresentamos a arquitetura do PP, o computador utilizado no nosso protótipo. Finalmente detalhamos a MAC proposta: descrevendo o seu quadro, os serviços da MAC e a especifcação estrutural desta sub-camada.

3.1 Definições

nó: chamamos os nós da RALFO de estação ou também de PP, em referência ao computador utilizado no nosso protótipo;

enlace: ligação unidirecional entre dois nós;

anel: a RALFO utiliza como meio físico dois anéis ópticos;

quadro: cada anel possui um quadro de transmissão de tamanho fixo e sincronizado pela estação monitora;

***gap*:** bits de sincronismo que completam o quadro, utilizando para isto o espaço não preenchido pelos *slots*;

***slot*:** canal que pode estar vazio ou preenchido por um pacote, tem tamanho fixo e é subdividido em campos;

campos: nos *slots* cheios, transmitem símbolos;

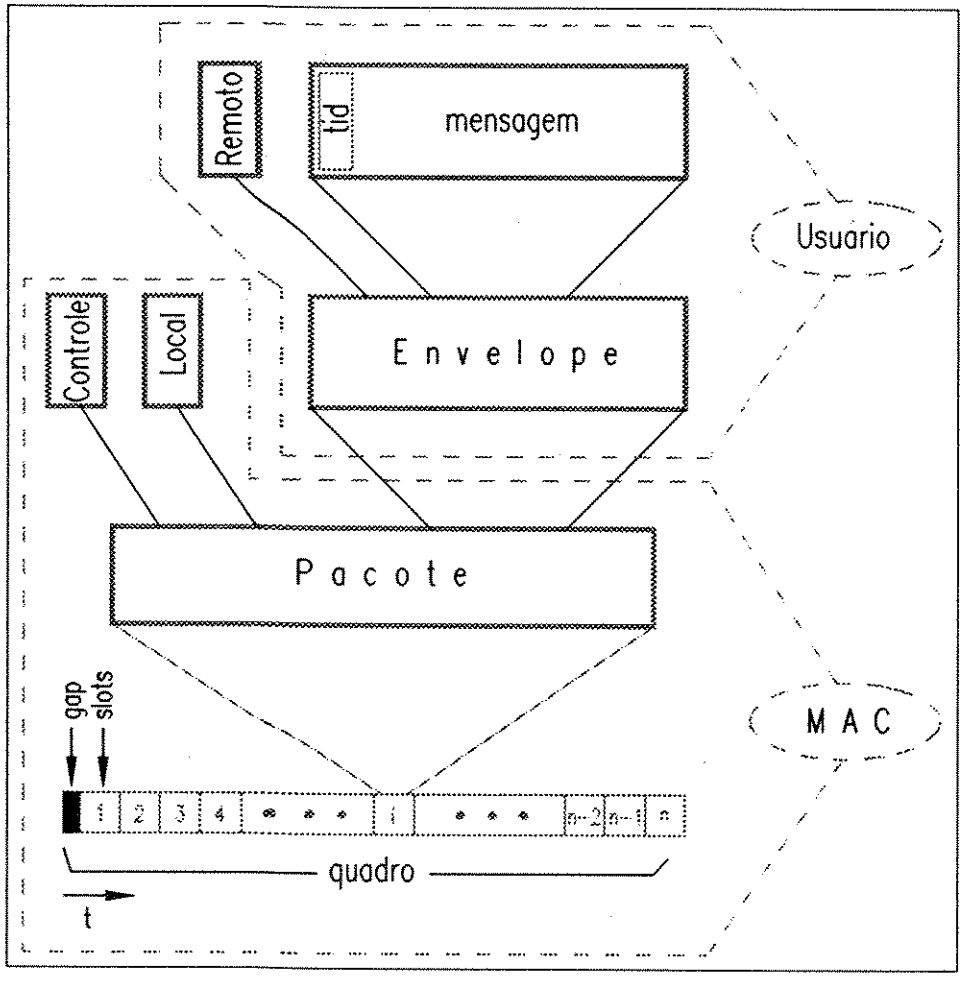


Figura 3.1: Encaminhamentos de mensagens/pacotes

símbolo: unidade de informação a nível de MAC;

usuário: quando nos referimos a usuário de uma camada, ou de um serviço oferecido por uma camada, estamos nos referindo à camada superior;

mensagem: quando um usuário solicita um serviço à MAC ele dispõe de uma mensagem, que poderá ser fragmentada em envelopes pelo próprio usuário para ser transmitida em diversos *slots*;

surto de voz: amostras de voz entre dois trechos de silêncio, uma mensagem da APL corresponde a um surto de voz;

***S*ource Address (src):** endereço da MAC remetente do pacote;

***T*arget Address (tgt):** endereço da MAC destinatária do pacote;

***T*erminal *Id*entifier (tid):** número de uma conexão da APL, estabelecida pela LLC/voz, associado ao terminal originador da mensagem. O `tid` só existe nas mensagens da APL, e está incorporado à própria mensagem;

envelope: mensagem do usuário acrescida do endereço do nó remoto;

pacote: corresponde à informação do usuário, acrescida dos símbolos de controle do pacote e do endereço do nó local.

A mensagem de um usuário da MAC é acrescida, pelo próprio usuário, do `tgt`, formando o que chamamos de envelope. A MAC recebe o envelope do usuário e acrescenta os símbolos de controle do pacote e o símbolo de endereço MAC local formando o pacote, que é alocado no *slot* conveniente pela MAC e encaminhado para a MAC par. A MAC par faz o serviço inverso, encaminhando o envelope para o usuário, com a ressalva que, no destino, o envelope corresponde à informação do usuário acrescida do endereço da MAC remoto, i.e. o endereço do remetente.

A compreensão destes termos é facilitada pela Fig. 3.1. Veremos mais adiante que as mensagens do usuário APL poderão ser fragmentadas, pelo próprio usuário, e transmitidas em quadros consecutivos.

Adotamos a seguinte nomeclatura para especificar a base de um número:

123: decimal;

0x7b: hexadecimal; e

%1111011: binária.

3.2 Introdução

A RALFO é uma rede, baseada na rede Anel de Cambridge ([NH82, HW93]), de serviços integrados, com dois serviços já implementados: o serviço telefônico e o de transmissão de dados. Teremos dois anéis, cada um possuindo um único quadro de 250 μs subdividido em *slots*. Os envelopes dos usuários concorrerão aos *slots* vazios que passarem pelo nó local. Tomamos o cuidado de priorizar a transmissão das mensagens de voz, em relação às outras, e dos envelopes da LLC/voz em relação aos da LLC/dados. Utilizamos alguns *slots* ociosos para transmitir envelopes da própria MAC com o objetivo de garantir a recuperação automática da rede em caso de falhas, como também a sua auto-configuração.

A RALFO foi proposta inicialmente por Paulo Guardieiro ([GM89]), como uma rede tipo anel de Cambridge modificada, visando ser uma rede de serviços integrados, que, na sua versão inicial, oferecerá ao usuário a possibilidade de transmitir dados e voz numa mesma rede. Com relação à rede de Cambridge, a RALFO inovou, não só por abranger o serviço telefônico, assim como por utilizar, por uma questão de confiabilidade, um anel duplo. E o fez, por uma questão tanto de confiabilidade a nível físico assim como visando poder atingir taxas mais elevadas nas suas futuras implementações. O meio físico adotado é fibra ótica. A Fig. 3.2 esquematiza a sua topologia.

Em [Gua91], Paulo Guardieiro propõe duas redes de serviços integrados baseadas no anel de Cambridge; uma com largura de faixa preestabelecida para transmissão de dados, chamada versão 2; e outra onde toda a faixa pode ser utilizada tanto para dados como para voz, que foi a adotada no nosso trabalho, a versão 1. Em conjunto com a descrição da rede proposta, Guardieiro faz uma análise de desempenho tanto para voz como para dados, apresentando resultados de modelos teóricos e de simulações. Na transmissão de voz ele leva em conta que haverá uma certa taxa de descarte para cada surto de voz. Esta taxa poderá chegar até a 1% sem comprometer a qualidade do serviço, posto que num surto de voz temos uma certa redundância, como descreve [Gon83]. Na transmissão de dados ele procura dimensionar o chamado Atraso de Tempo Médio (atm), que indica o tempo médio de espera para um envelope de dados ser transmitido, em função do fluxo de serviço da rede. Veremos mais tarde que estes estudos foram importantes na definição do quadro utilizado.

Nossa proposta tem algumas características interessantes:

- é aderente ao modelo de referência OSI;
- prevê o descarte de amostras de voz;

- transmite apenas surtos de voz;
- o método de acesso ao meio físico é totalmente descentralizado; e
- é uma rede tolerante a falhas físicas nos enlaces/nós.

3.3 Método de Acesso ao Meio

Utilizamos dois métodos de alocação dos *slots* em função do tipo de envelope a ser transmitido. Chamaremos estes métodos de tipo 1 e tipo 2. O tipo 1 destina-se à transmissão de mensagens de voz e o tipo 2 destina-se aos outros tipos de mensagens.

Para transmissão de mensagens do tipo 1 levamos em conta as seguintes características da telefonia:

- um usuário da APL intercala numa conversação surtos de voz com períodos de silêncio, com valores típicos variando respectivamente de $1,4 \pm 0,4$ s e $1,8 \pm 0,6$ s, conforme [Bra68];
- o atraso máximo ponto-a-ponto, dispensando o uso de supressores de eco, segundo [Gon83] é de 22,5 ms;
- a existência de redundâncias num surto de voz permite o descarte de até 1% das amostras de voz sem comprometer a qualidade telefônica, como pode ser visto em [Gon83]; e
- tem freqüência de amostragem de 8 kHz.

Assim uma mensagem telefônica é naturalmente fragmentada pela APL, posto que suas amostras são feitas a cada 125 μs, estas amostras são acumuladas pela APL, anexadas ao endereço do terminal remoto e, na forma de envelope, encaminhadas à MAC, que, tratando-se de uma mensagem nova, encaminhará para a MAC par no próximo *slot* vazio. Caso não haja nenhum *slot* vazio no intervalo correspondente ao período de um quadro, o envelope será desprezado. Um *slot* uma vez alocado a uma mensagem ficará reservado até o final dela, i.e. até que a APL pare de mandar envelopes provenientes de um determinado terminal, quando será liberado como vazio para o próximo nó.

Para as mensagens do tipo 2 levamos em conta as seguintes características das mensagens da LLC/voz, da LLC/dados e da MAC:

- impomos um MTU fixo, portanto, para a mensagem do usuário, o que implicará que a MAC não fará fragmentação;
- o serviço deverá ter baixo BER.

Haverá para os usuários deste serviço a confirmação do recebimento do pacote pela MAC par, cabendo porém, a este usuário, a tarefa de retransmitir a mensagem caso não receba a confirmação.

Assim os envelopes do tipo 2 aguardarão sempre a chegada de um *slot* livre, que será alocado à mensagem mais velha deste tipo, prevalecendo a seguinte ordem de prioridade:

1. envelope da LLC/voz,
2. envelope da LLC/dados, e
3. envelope da MAC;

desde que não haja alguma mensagem da APL aguardando um *slot* vazio. Os envelopes do tipo 2 nunca serão descartados, aguardando o tempo que for necessário para serem encaminhados à rede.

Para ambos os tipos de mensagem um *slot* sempre será encaminhado, pelo nó que originou a mensagem, como vazio, para o próximo nó do anel, ao término da mensagem, proporcionando assim maior eqüidade na distribuição dos *slot*s livres na rede.

É prevista a existência de uma estação monitora, eleita automaticamente pelas MACs de um anel, que tem os seguintes objetivos:

- possuir a camada PHY responsável pelo sincronismo dos quadros;
- retirar do anel os pacotes perdidos, i.e., os pacotes que deram mais de uma volta no anel.

3.4 As Camadas e Sub-camadas

Adotamos também na RALFO uma estrutura de camadas aderente ao modelo de referência OSI ([ISO82]), porém simplificado, esquematizada na Fig. 3.3:

- Camadas físicas,
- Sub-camada MAC,
- Sub-camada LLC/dados,
- Sub-camada LLC/voz,
- Camada de APL/dados, e
- Camada de APL.

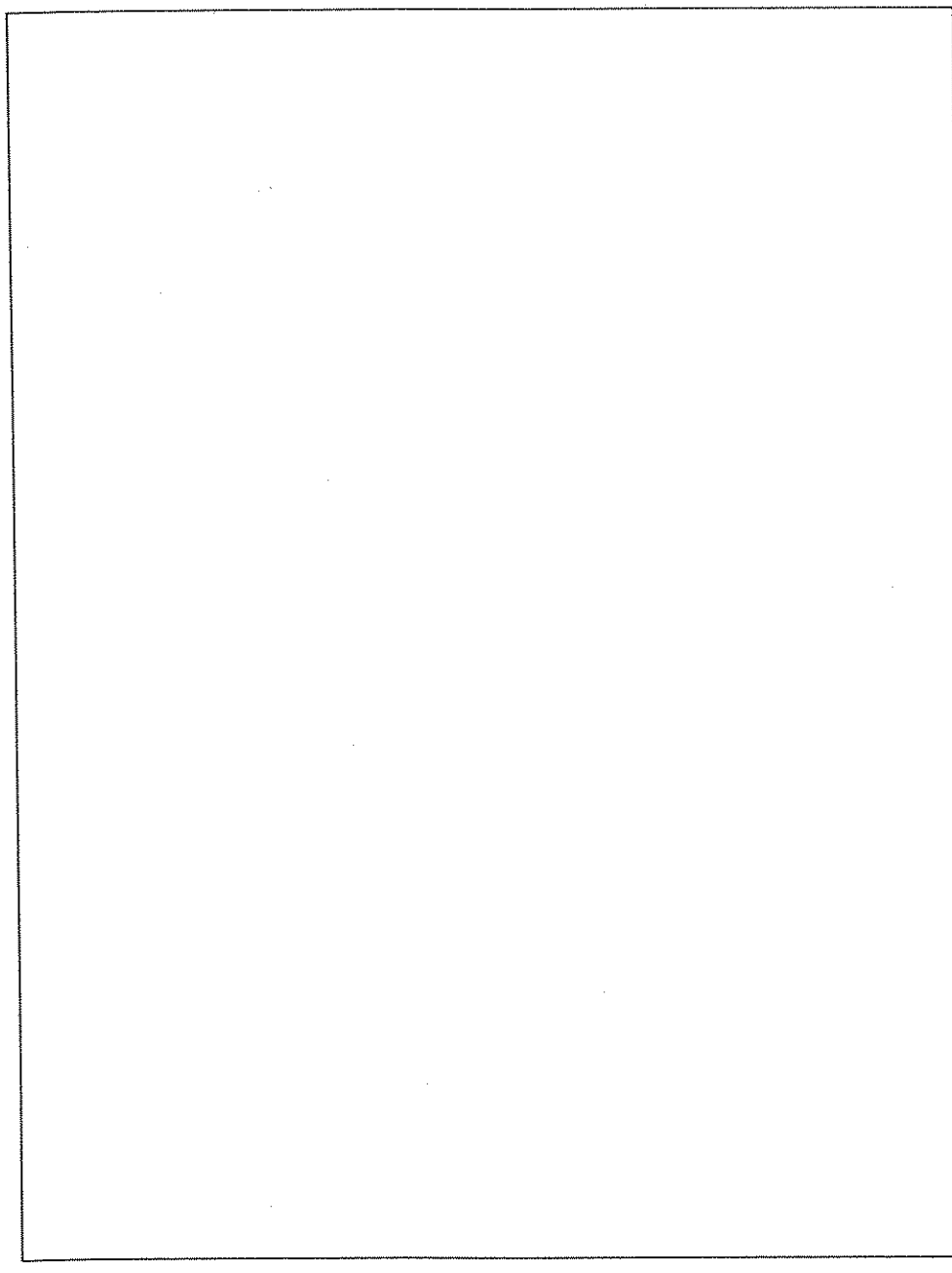


Figura 3.2: Topologia

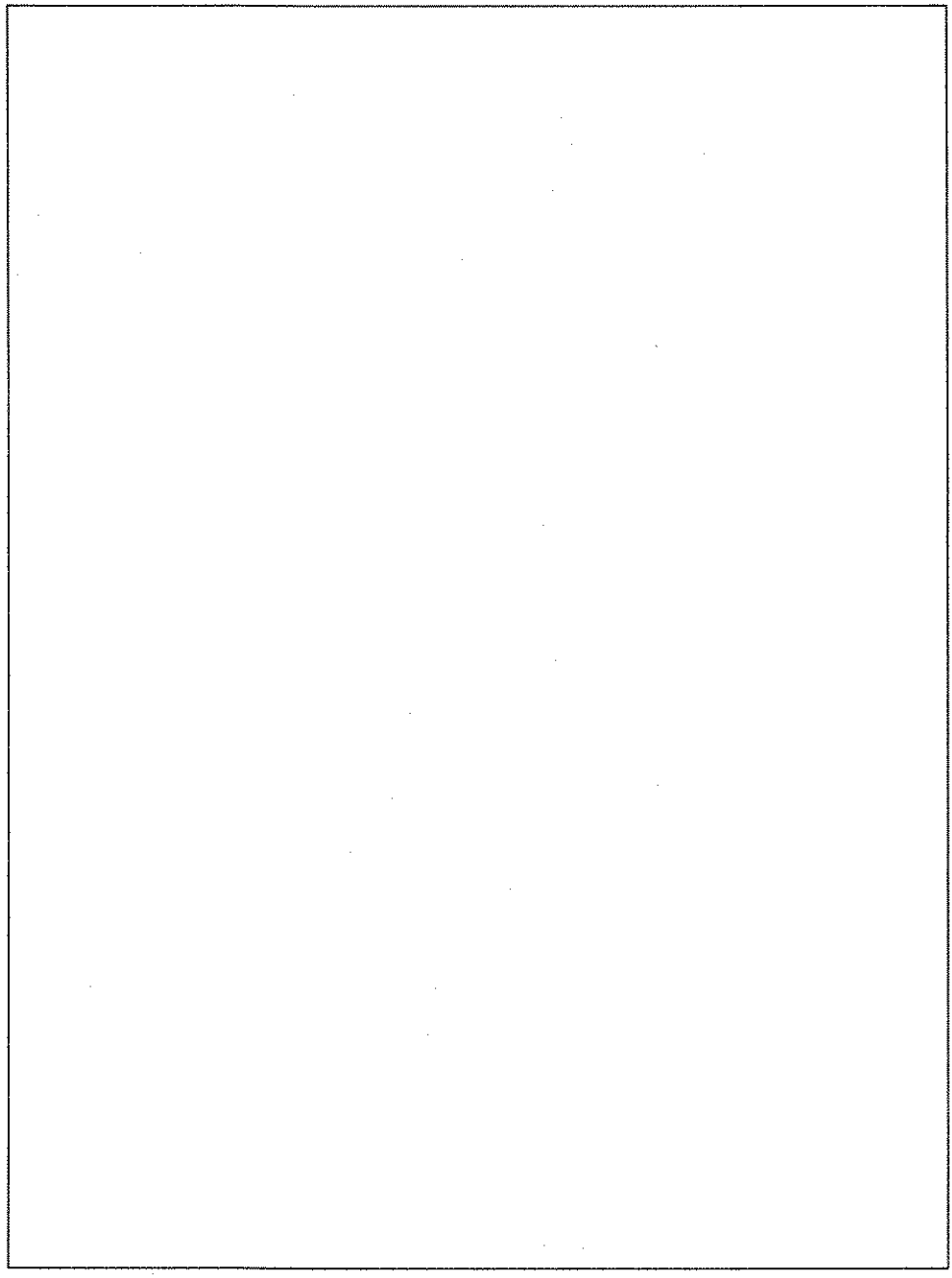


Figura 3.3: Camadas e sub-camadas

3.4.1 A Camada Física

Temos duas camadas físicas idênticas, uma para cada anel, responsáveis (1) pelas interfaces físicas, portanto conversão ótica/eletrônica e eletrônica/ótica, e (2) pela geração do relógio para a MAC, portanto pelo sincronismo e também pelo sincronismo a nível de quadro e *slots*. Cabe assim à camada PHY garantir um número de *slots* existentes num quadro. Para tal ela será informada, pela MAC, sobre a quantidade de nós existentes no anel, o que permitirá, conhecendo o tempo de propagação no meio, prover o atraso necessário para cada *slot*. O sincronismo a nivel de bit será estabelecido em todo um anel por uma única camada PHY, a da estação monitora, definida pelas camadas MACs dos anéis.

3.4.2 A Sub-camada MAC

A sub-camada MAC será única para cada nó da RALFO, cabendo-lhe controlar o método de acesso ao meio, distribuir os envelopes provenientes dos seus usuários nos dois aneis, encaminhar os envelopes provenientes dos usuários remotos, estabelecer o endereço MAC de cada nó, estabelecer o nó monitor dos aneis, detectar falhas nos enlaces ou nós e, quando necessário, reconfigurá-los. A MAC oferece dois serviços, os chamados:

- serviço sem conexão, ou tipo 2; e
- serviço isócrono, ou tipo 1

utilizando-se para isso de, no máximo, apenas três primitivas, que são:

- `MA-data.request`;
- `MA-data.indication`; e
- `MA-data.confirm`;

A descrição detalhada da MAC é o principal objetivo deste capítulo e sua descrição a nível de projeto eletrônico será vista no capítulo 5.

3.4.3 A Sub-camada LLC/dados

À sub-camada LLC/dados cabe o controle dos enlaces lógicos de dados. Seguimos aqui a padronização estabelecida pelo IEEE no seu padrão 802.2 ([IEE85b]), sendo utilizada a LLC tipo 2, que provê serviços com conexão, controle de fluxo e recuperação de erros. Em [Uss93], Maria Cristina Ussami detalha os serviços oferecidos, a implementação utilizando *C*CITT *High Level Language* (CHILL) ([CCI89a, Tel90b]) e faz uma análise de desempenho da LLC/dados.

3.4.4 A Sub-camada LLC/voz

A sub-camada LLC/voz é responsável por todo protocolo de sinalização telefônica, provendo, para tal, basicamente o serviço de estabelecer e encerrar uma conexão, sendo portanto capaz de detectar quando o terminal remoto está livre, ocupado, quando é inexistente, quando o usuário atendeu e quando encerrou uma conexão. É utilizado para isto um protocolo estabelecido em [Car91] e em [Pes91] entre a LLC/voz e a APL. Cabe também à LLC/voz fazer a tarifação telefônica. A sub-camada LLC/voz está descrita por Cláudia Carneiro em [Car91].

3.4.5 A Camada APL

A camada de APL é responsável pela digitalização do sinal telefônico, requisição do estabelecimento e encerramento de conexão à sub-camada LLC/voz. Deverá portanto gerar os sinais: linha disponível, destino ocupado, destino sendo chamado e destino inexistente no terminal do usuário. Deverá também gerar o sinal necessário para o toque da campainha no terminal telefônico; tudo isto em função do protocolo estabelecido com a LLC/voz. Uma vez estabelecida uma conexão, a APL utiliza diretamente os serviços da MAC para a transmissão dos surtos de voz. Caberá a APL fazer a detecção dos intervalos de silêncio do seu usuário, de forma a enviar para a MAC envelopes com amostras de voz apenas nos surtos de voz. A epecificação eletrônica dos circuitos da APL assim como de toda a placa da APL encontra-se em [Pes91].

3.5 O Nó

No protótipo da RALFO, ora em desenvolvimento no DT, estamos usando como nós da rede computadores PPs desenvolvidos pelo CPqD da Telebrás não só por dispormos, ao utilizá-lo, da infra-estrutura necessária para o projeto: acesso aos projetos de suas placas, especificação do barramento, sistema operacional multitarefa com recursos para aplicações em tempo real (*S*istema *O*peracional do *PP* (SOP)), processamento paralelo, linguagem de programação de alto nível (CHILL) e ambiente de programação (*A*mbiente de *P*rogramação do *C*HILL *C*PqD (APCC)) mas também devido a interoperabilidade com as centrais Trópicos da Telebrás. Uma discussão mais detalhada sobre os aspectos de *software* que nos levaram a optar pelo SOP e pelo CHILL pode ser envontrada em [Uss93]. A descrição formal do CHILL pode ser encontrada na norma [CCI89a].

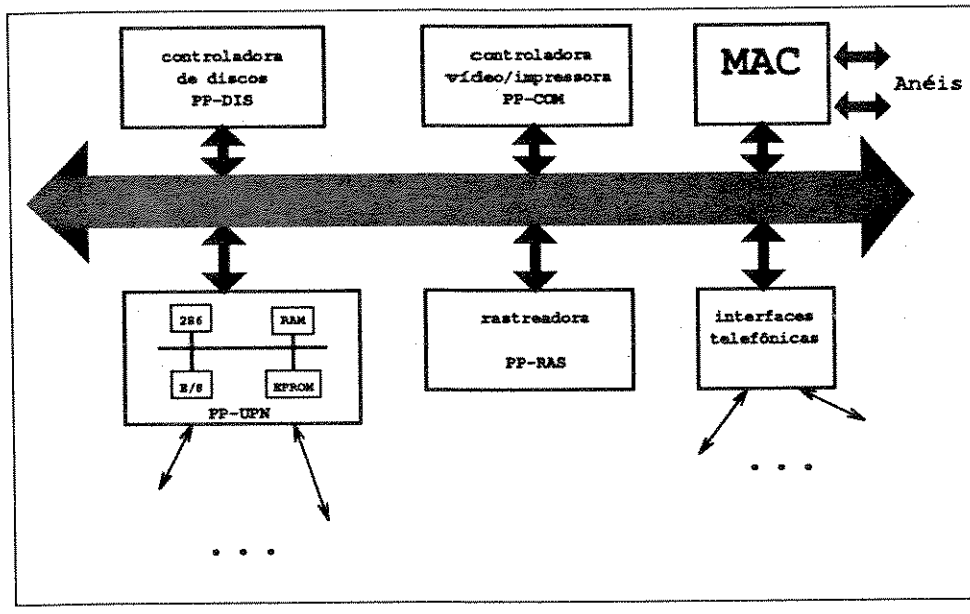


Figura 3.4: Arquitetura de um nó

O PP é um computador baseado na família Intel 80X86[1] com um barramento, cujo acesso pode ser feito numa das duas configurações: serial (*dasy-chain*) ou paralela. A descrição completa do PP poderá ser encontrada no manual [Tel90a] e as especificações de cada placa estão indicadas a seguir. O nosso protótipo, esquematizado na Fig. 3.4, possui 8 *slots* para serem utilizados pelas seguintes placas ([Pes91]):

Unidade de Processamento Numérico (UPN): ([Tel88b, Tel88c]) placa Central Process Unit (CPU) constituida por:

- 1 processador 80286;
- 1 co-processador 80287;
- 512 kBytes de Random Access Memory (RAM);
- 64 kBytes de Erasable and Programable *ROM* (EPROM); e
- 2 interfaces RS232;

DIS: ([Tel90d, Tel90c]) placa controladora de disco, com capacidade para controlar até:

[1] Diponível hoje com os μ-processadores IAPx-80286 e IAPx-80386 ([Pes91, Uss93])

- 4 discos rígidos de até 288 MBytes cada;
- 4 unidades de discos flexíveis de 8" e 5 1/4"; e
- 2 unidades de fitas cartucho;

COM: ([Tel87b, Tel89b, Tel90f]) placa:

- controladora de vídeo;
- controladora de duas interfaces RS232;
- que possui um bloco de memória dual-port que serve como memória comum a todas as placas do nó;

SER: ([Tel90e]) placa controladora de até 8 interfaces RS232;

RAS: ([Tel86, Tel87a, Tel88a, Tel89a]) placa rastreadora que permite acompanhar o fluxo de informações no barramento, simulando um analizador lógico

Acrescentamos mais duas placas a cada nó do PP:

APL/LLC/voz: Placa hospedeira da APL e LLC/voz com:

- 1 μ-processador 80286; e
- 16 interfaces para terminais de voz (telefones).

Nesta placa estão localizados todos os circuitos utilizados pela camada de APL e descritos em [Pes91];

MAC/PHY: Placa hospedeira da sub-camada MAC e da camada PHY com:

- 2 interfaces de entrada ótica; e
- 2 interfaces de saída ótica.

Esta placa é toda desenvolvida com circuitos dedicados detalhados no capítulo 5.

A sub-camada LLC/dados fica residente na UPN.

3.6 A MAC

A principal função de uma MAC é o controle de acesso ao meio físico. Apresentaremos o quadro proposto, os serviços da MAC e detalharemos estruturalmente essa sub-camada.

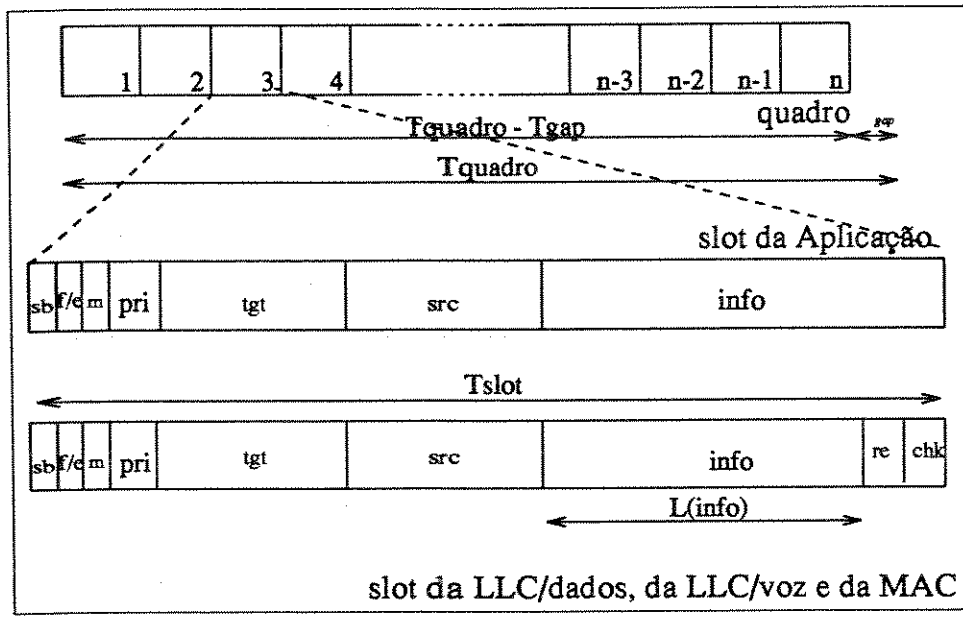


Figura 3.5: O quadro

3.6.1 O Quadro

Adotamos um método *sloted ring* que multiplexa diversos pacotes em células temporais denominadas *slot*s. Ao conjunto de todos os *slot*s presentes num determinado instante no anel chamamos de quadro. Detalharemos em seguida este quadro, seus *slot*s e seus campos.

O quadro da RALFO está esquematizado pela Fig. 3.5 onde temos os seguintes parâmetros:

$T_{\mathbf{quadro}}$: é o período do quadro;

T_{gap}: é o comprimento do *gap*;

T_{slot}: é o comprimento do *slot*;

L(*Information* (`info`)): é o comprimento do campo `info`; e

n: é a quantidade de *slot*s por quadro;

e os seguintes campos:

***Start Bit* (`sb`)**: para sincronismo do *slot*, 1 bit;

Full/Empty (`f/e`): indica se há ou não um pacote no *slot*, 1 bit;

`m`: indica quando o pacote já passou pela estação monitora, 1 bit;

Priority (`pri`): indica o tipo de serviço do pacote, 2 bits;

`tgt`: endereço MAC do destinatário, 8 bits;

`src`: endereço MAC do remetente, 8 bits;

`info`: mensagem, ou fragmentação da mensagem, do usuário, L(`info`) bits;

Response (`re`): indica se o pacote foi aceito ou não pelo destinatário, 2 bits; e

Check Sum (`chk`): *check sum* dos campos `pri`, `tgt`, `src` e `info`, 2 bits.

O período de um determinado *slot* no anel deve ser múltiplo do período de amostragem telefônica, garantindo assim que, no retorno de um *slot* tranportador de uma mensagem da APL, já haja mais envelopes desta mensagem aguardando o *slot* reservado. Desta forma teremos:

$$T_{\text{quadro}} = aT_{\text{t}}, \qquad a \in \mathbb{N}^* \tag{3.1}$$

onde:

T_{t}: é o período de amostragem telefônica (125 μs).

Assim teremos num envelope da APL pelo menos a amostras de voz da mensagem corrente. É facultado à camada APL utilizar um *slot* para transportar um número maior do que a de amostras de uma mensagem, isto poderá ocorrer em dois casos:

- apenas quando iniciamos uma mensagem, visando poupar os recursos da rede; ou
- quando a APL optar por concorrer a *slots* vazios a cada envelope enviado à MAC, acumulando amostras de voz até atingir o limite disponível para o `info`, visando também poupar recursos da rede.

Em ambos os casos o usuário da APL corre maior risco de ter seu envelope descartado.

Normalmente a APL utilizará um envelope para cada a amostras da sua mensagem e caberá à APL par forçar para que as amostras que vão chegando aguardem o seu tempo para serem enviadas para o usuário, o que chamamos na telefonia de *retime*.

Terminal	Taxa de transferência pico [kbps]	Utilização média [erlangs]	Vazão média [kbps]
telefone	128	0,13	8,0
proc. de texto	2,4	0,4	0,048
terminal de dados	2,4	1,0	0,12
facsimile	9,6	0,21	2,0
copiadora	48,0	0,036	1,7
impressora	9,6	0,36	3,5

Tabela 3.1: Utilização de Rede Local em Escritórios

Trabalhamos com um $a = 2$ buscando aumentar a eficiência (η) da rede, definida pela equação 3.3:

$$\mathrm{L(i)} \triangleq \text{Comprimento do campo } i, \quad [\mathrm{s}] \tag{3.2}$$

$$i \in \left\{ \mathtt{sb}, \mathtt{f/e}, \mathit{Monitor}(\mathtt{m}), \mathtt{pri}, \mathtt{tgt}, \mathtt{src}, \mathtt{re}, \mathtt{chk} \right\}$$

$$\eta = 1 - \frac{\sum_i \mathrm{L(i)}}{T_{slot}}. \tag{3.3}$$

Se o nosso objetivo fosse transmitir apenas mensagens telefônicas, optaríamos por *slot*s cujos campos `info`s fossem do tamanho exato para transmitir a amostras de voz, pois assim teríamos o maior número possível de *slot*s por quadro, o que permitiria o maior número possível de transmissões simultâneas de mensagens na rede, e conseqüente diminuição da fração de envelopes tipo 1 descartados pela rede (Φ_E). Uma análise teórica desta afirmação poderá ser encontrada em [GM89] e o resultado desta análise para o nosso quadro em [AYM93].

Se o nosso objetivo fosse transmitir apenas mensagens de dados, optaríamos por um campo `info` cujo L(`info`) levaria a uma η alta, buscando também otimizar o atm, que indica o tempo médio de espera para que um envelope do tipo 2 seja transmitido. Em [Gua91] encontramos uma análise teórica e simulações, tanto para atm assim como para Φ_E.

Em [RSS81], Richer, Steiner e Seigoku mostram alguns resultados empíricos para utilização de serviços de dados e de voz em empresas, resultados estes reproduzidos na tabela 3.1.

Isto nos leva a concluir que o tráfego de pacotes da APL é, em média, muito mais intenso do que o de pacotes da LLC/dados. Desconsideramos aqui o tráfego de pacotes da LLC/voz e da MAC por terem vazão desprezível em relação aos dois primeiros. Na tabela anterior não são considerados

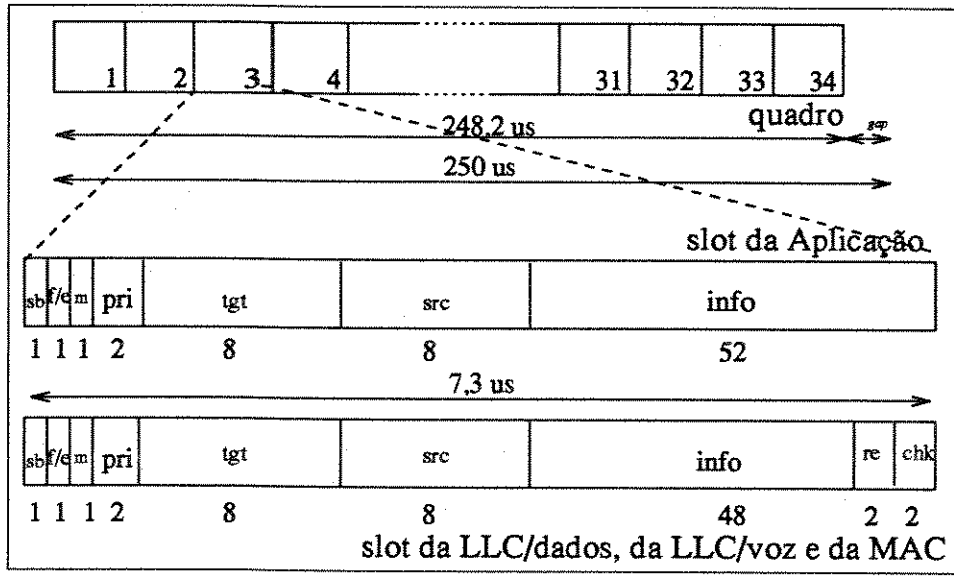


Figura 3.6: Quadro totalmente parametrizado

alguns serviços mais novos, como os serviços de sistemas de arquivos pela rede, que ocupam razoavelmente a largura de faixa de uma rede local. Veremos mais adiante (capítulo 6) os efeitos desta aproximação.

Buscando então privilegiar o tráfego telefônico optamos por utilizar um pacote cujo L(`info`) = 48bits, chegando então no quadro esquematizado pela Fig. 3.6

3.6.2 Os Serviços

A MAC provê três serviços para os seus usuários, esquematizados pela Fig. 3.7:

- `MA-data.request`;
- `MA-data.indication`; e
- `MA-data.confirm`.

O `MA-data.confirm` será provido apenas para a LLC/dados e LLC/voz, indicando o símbolo lido no campo `re` dos pacotes que retornam ao nó local.

Os envelopes, compostos pela mensagem, pelo endereço remoto e pelo `tid`, são encaminhados à MAC via SDU `MA-data.request`, este envelope é

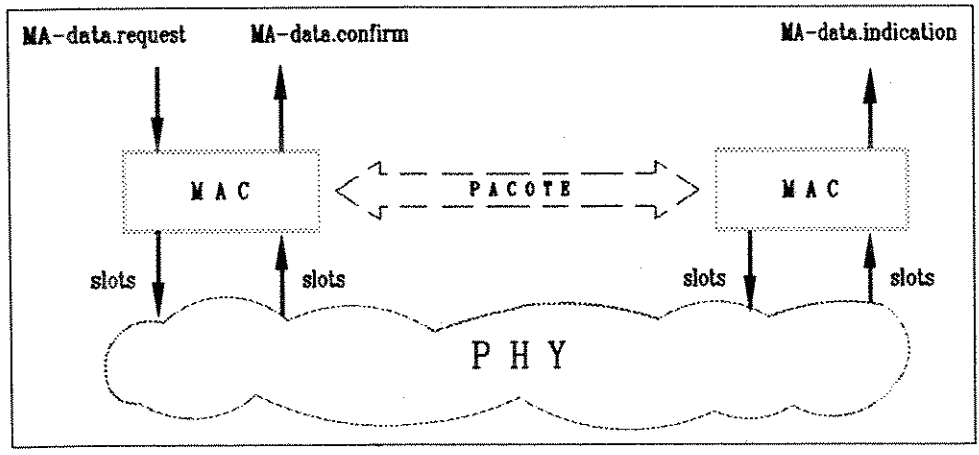


Figura 3.7: Serviços da *MAC*

convertido em pacote pela MAC e aguardará a chegada de um *slot* vazio, ou, caso seja um envelope da APL com um *slot* já reservado, aguardará o retorno deste *slot*. No nó remoto, a MAC receberá o pacote e enviará ao usuário uma SDU `MA-data.indication` com o envelope retirado do pacote. Caso o usuário seja a LLC/dados ou a LLC/voz, a MAC colocará o simbolo conveniente no campo `re` do pacote, que retornará ao nó original. De volta ao nó original, a chegada do pacote provocará o envio de uma `MA-data.confirm` para o usuário (pacote tipo 2) ou provocará o envio de outro pacote para o nó remoto (pacote tipo 1) sem confirmar o recebimento ou não do pacote anterior à APL.

A troca de SDU entre os usuários e a MAC dar-se-á por meio de filas *F*irst *i*n *f*irst *o*ut (FIFO) implementadas numa memória comum da estação. Estas filas terão as seguintes características:

- terão tamanho limitado;
- serão unidirecionais;
- serão compostas de linhas que conterão todo um envelope;
- suas leituras/escritas serão feitas por linha, i.e., um envelope nunca será lido ou escrito parcialmente.

Cada fila terá apenas um sorvedor de linhas e um provedor de linhas. O sorvedor será aquela camada/sub-camada que retirará os envelopes da fila e o provedor será aquela camada/sub-camada que colocará os envelopes na fila.

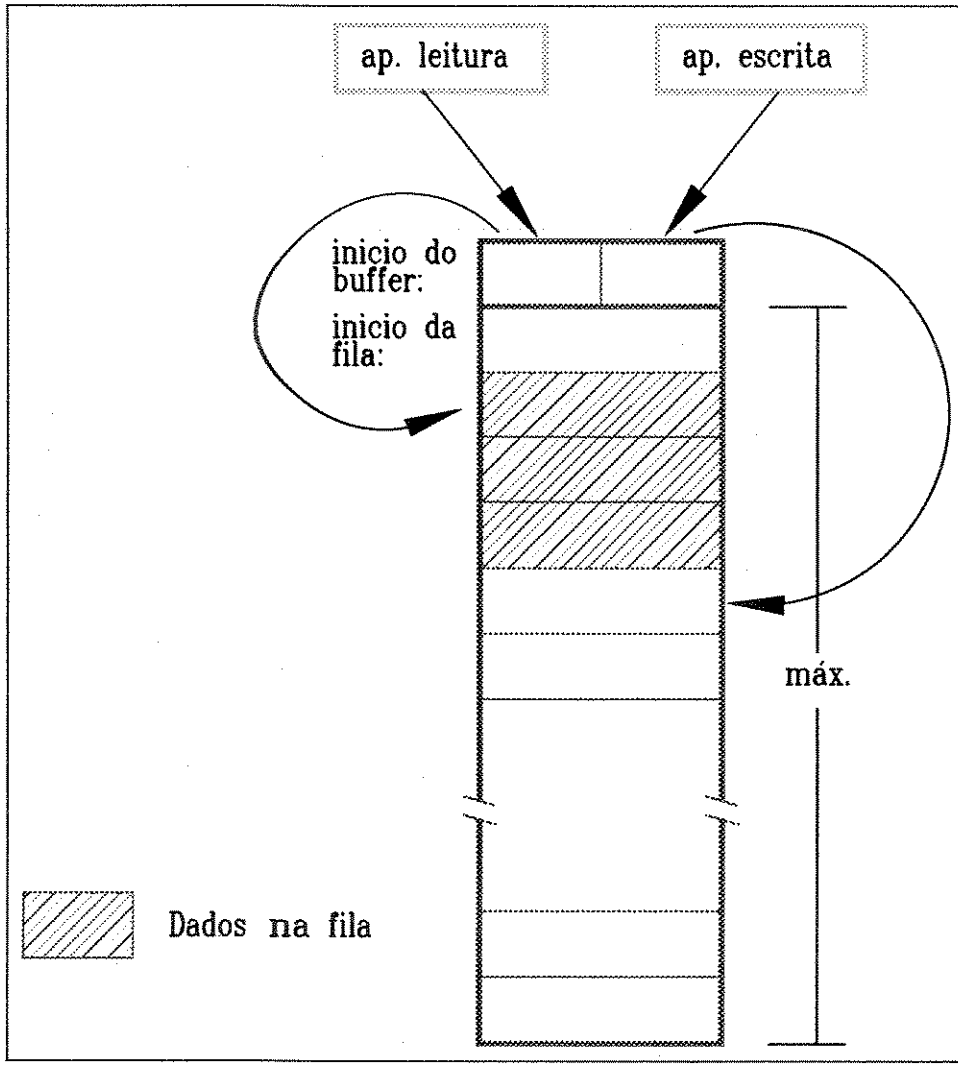


Figura 3.8: Buffers de comunicação com os usuários

Cada fila terá os seguintes parâmetros, esquematizados na Fig. 3.8, configurados no sistema:

- endereço inicial da fila;
- apontador de escrita;
- apontador de leitura;
- endereço do apontador de escrita;
- endereço do apontador de leitura; e
- tamanho máximo da fila.

Os apontadores de escrita e de leitura terão seu conteúdo modificados apenas pelo provedor e pelo sorvedor, respectivamente, da fila. Eles serão incrementados a cada escrita/leitura e, quando seus valores ultrapassarem o tamanho máximo da fila a eles será atribuído o endereço inicial desta fila.

O provedor, antes de encaminhar uma linha para a fila, verificará se o endereço no apontador de escrita não coincide com o do apontador de leitura incrementado, o que indicaria que a fila está cheia. Caso isto ocorra, o provedor tentará encaminhar a linha mais tarde.

O sorvedor, antes de fazer a leitura de uma linha da fila, verificará se o endereço no apontador de leitura não coincide com o do apontador de escrita, o que indicaria que a fila está vazia. Caso isto ocorra, o provedor tentará ler a fila mais tarde.

3.6.3 A Especificação Estrutural

A MAC é formada por 17 módulos, que podem ser de 7 tipos diferentes, esquematizados na Fig. 3.9:

- *Mó*dulo de *I*nterface como o *B*arramento (MoIB);
- *Mó*dulo de *Co*ntrole de *Fi*la de *E*ntrada (MoConFiE);
- *Mó*dulo de *Co*ntrole de *Fi*la de *S*aída (MoConFiS);
- *Mó*dulo de *Tra*tamento de *P*rioridades (MoTraP);
- *Mó*dulo de *D*istribuição de *En*velopes (MoDEn);
- *Mó*dulo de *C*ontrole de *A*cesso ao Meio (MoCA); e
- *Mó*dulo de *T*este e *I*niciação (MoTI).

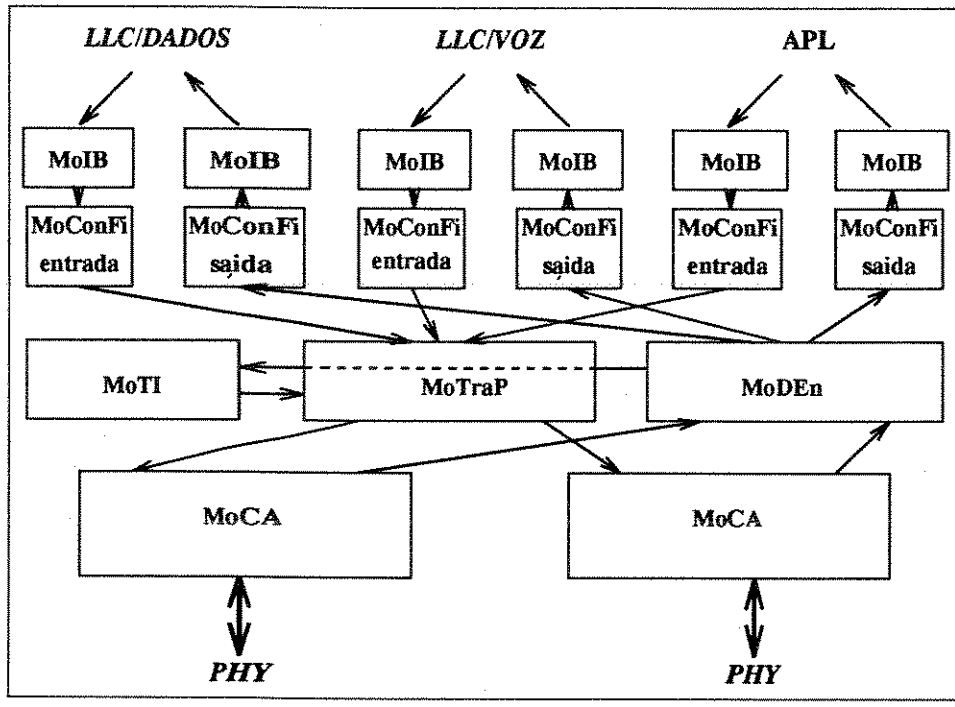


Figura 3.9: Os módulos

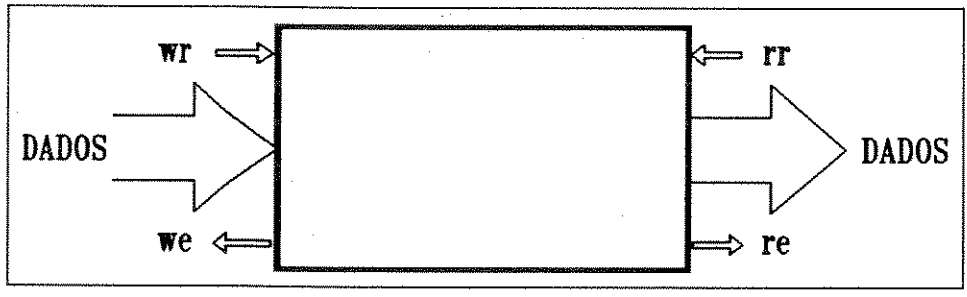


Figura 3.10: Convenção utilizada nos sinais

Descreveremos em seguida estruturalmente estes modulos.

Para a nomeclatura dos sinais adotamos a seguinte convenção, esquematizada na Fig. 3.10:

***W*rite *R*equest *I*nput ($\texttt{wr}_{in}$)**: sinal de solicitação para envio de dados ao componente;

***W*rite *E*nable *Out*put ($\texttt{we}_{out}$)**: sinal de habilitação do componente para receber dados;

***R*ead *R*equest *I*nput ($\texttt{rr}_{in}$)**: sinal de solicitação para envio de dados do componente;

***R*ead *E*nable *Out*put ($\texttt{re}_{out}$)**: sinal de habilitação para leitura dos dados do componente.

Este sinais serão utilizados para sincronismo de recebimento/envio de dados de/para um componente, em geral o $\texttt{wr}_{in}$ de um componente estará ligado a um $\texttt{re}_{out}$ de outro componente ou então o $\texttt{rr}_{in}$ de um estará ligado ao $\texttt{we}_{out}$ do outro. Raramente será necessária a existência dos quatro sinais entre dois componentes.

Estaremos também referenciando muitas vezes um barramento de entrada/saída de dados como sendo um único sinal para simplificar a descrição.

Nesta seção referenciamos um componente a que utilize um componente b como a sendo usuário de b.

Alguns sinais serão referenciados como:

$$\texttt{sinal}\{a,b\}$$

querendo dizer **sinal**a e/ou **sinal**b, dependendo do contexto. Assim quando nos referimos aos sinais $\texttt{sinal}\{a,b\}$ queremos dizer **sinal**a *e* **sinal**b, já quando nos referimos ao sinal $\texttt{sinal}\{a,b\}$ queremos dizer **sinal**a *ou* **sinal**b.

MoIB

O MoIB é o único módulo que é dependente do computador utilizado. Ele será responsável pelo acesso direto ao barramento do equipamento, recebendo, processando e enviando os sinais necessários ao barramento e simultaneamente interagindo com o *Mó*dulo de *Con*trole de *Fi*la (MoConFi) através de uma interface definida independentemente do barramento utilizado. No PP o MoIB cuidará da interface com o barramento do PP, que é um barramento que funciona na base do *Dasy Chain*, permitindo o acesso múltiplo e descentralizado.

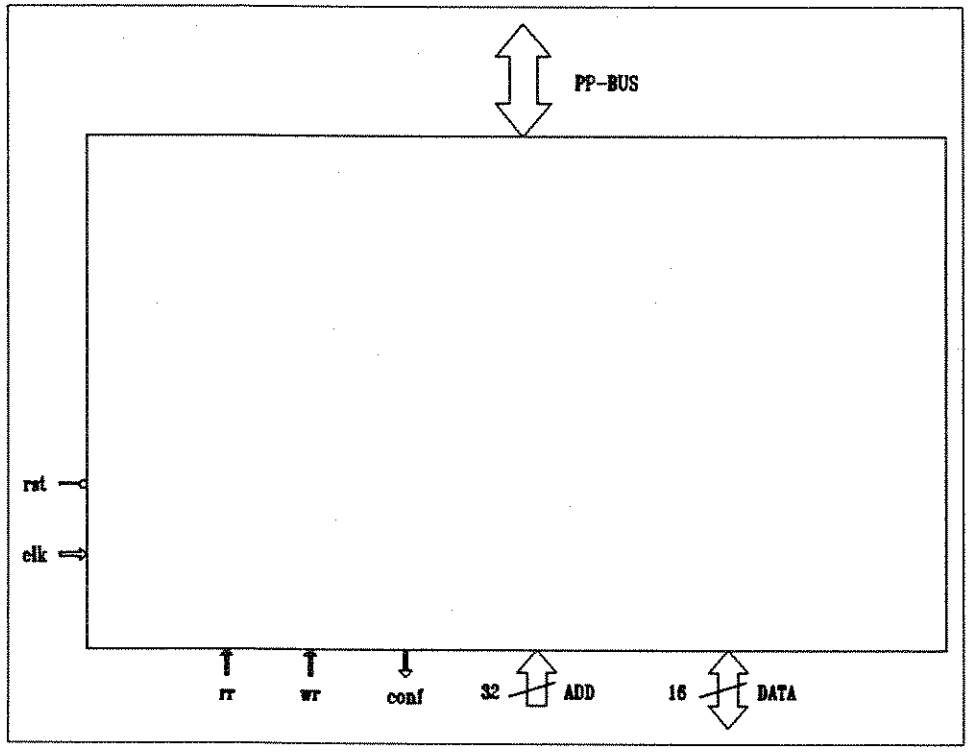


Figura 3.11: Símbolo do MoIB

O endereço inicial de cada fila de interface com o usuário é definido no respectivo MoIB.

O MoIB possui uma interface com o barramento do PP especificada pelo documento [Tel] e que não será detalhada neste trabalho, e uma interface com os MoConFis conforme esquematizado na Fig. 3.11. Para o MoConFi, o MoIB dispõe dos seguintes sinais:

$\texttt{rr}_{in}$: Quando ativo indica ao MoIB que o dado contido no endereço disponível no *Add*ress *In*put ($\texttt{add}_{in}$) deverá ser lido da memória, colocado no *Data* ($\texttt{data}_{in/out}$), e indicado no *Conf*irm *Out*put ($\texttt{conf}_{out}$) que o dado está disponível por dois ciclos do relógio;

$\texttt{wr}_{in}$: Quando ativo indica ao MoIB que o dado disponível no pino $\texttt{data}_{in/out}$ deverá ser escrito no endereço indicado na $\texttt{add}_{in}$. O usuário deverá manter os valores da $\texttt{data}_{in/out}$ e da $\texttt{add}_{in}$ até que o MoIB indique o sucesso da operação ativando o $\texttt{conf}_{out}$;

conf$_{out}$: Indica ao usuário que a operação solicitada, escrita ou leitura, está encerrada;

data$_{in/out}$: entrada/saída de dados; e

add$_{in}$: endereço para escrita/leitura.

O endereço apresentado na **add**$_{in}$ não é absoluto mas relativo ao endereço físico inicial da fila, conforme esquematizado na Fig. 3.8. Ficará gravado no MoIB o endereço incial do seu *buffer*, e caberá ao MoIB o mapeamento do endereço passado pelo usuário com o endereço físico.

MoConFiE

O MoConFiE cuidará do controle da fila de entrada de um determinado usuário para a MAC, verificando periodicamente se há envelopes para serem encaminhados à rede, i.e., se a fila não está vazia, indicando tal fato ao MoTraP. Para controle da fila o MoConFiE utilizará os seguintes parâmetros:

- Endereço inicial relativo da fila, sempre será 0x0000;
- Tamanho da fila: dependente do usuário;
- Largura da fila: 64 bits;
- Apontador de escrita: indica onde será feita a próxima escrita; e
- Apontador de leitura: indica onde será feita a próxima leitura.

O tamanho da fila será um parâmetro de cada MoConFi. O apontador de escrita será lido do endereço relativo 0x0004. Este apontador é modificado apenas pelo usuário. O apontador de leitura será controlado pelo MoConFiE e, toda vez que modificado, escrito no endereço relativo 0x0000. Todos os dois apontadores estão limitados em 4 bytes, como pode ser visto na Fig. 3.8.

O MoConFiE será responsável pela leitura da fila de entrada do usuário. Ele só fará a leitura se a fila não estiver vazia.

Num ciclo de leitura será lido todo o envelope, composto por 64 bits.

O apontador de escrita, controlado pelo usuário, será lido a cada início de ciclo de leitura e comparado com o apontador de leitura, que é mantido pelo MoConFi, e, após o ciclo de leitura, este endereço será incrementado (de 64 bits) e colocado no endereço relativo à disposição do usuário.

O MoConFiE possui uma interface com o respectivo MoIB e uma interface com o MoTraP esquematizadas na Fig. 3.12 e cujos sinais estão descritos a seguir:

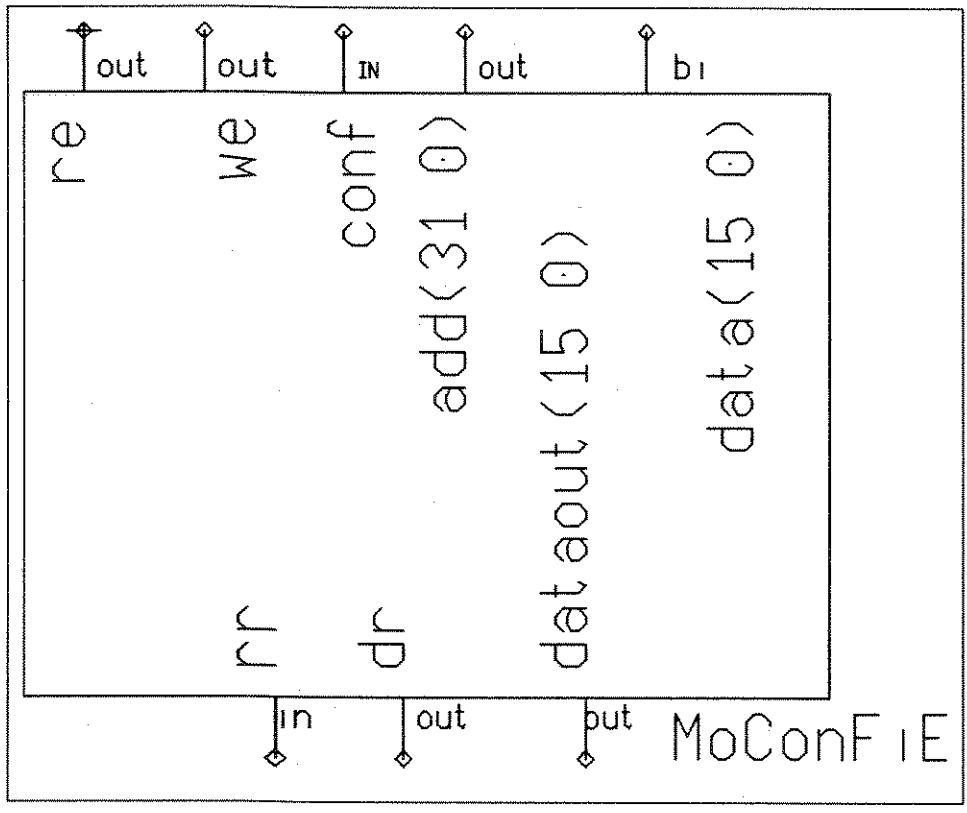


Figura 3.12: Símbolo do MoConFiE

$\mathtt{we}_{out}$: Quando ativo indica ao MoIB que o dado contido no $\mathtt{data}_{in/out}$ deve ser escrito no endereço relativo contido no *Add*ress *Out*put ($\mathtt{add}_{out}$). O MoConFiE assumirá que o ciclo de escrita foi finalizado com sucesso quando o *Conf*irm *In*put ($\mathtt{conf}_{in}$) estiver ativo;

$\mathtt{re}_{out}$: Quando ativo indica ao MoIB que o dado do endereço relativo contido no $\mathtt{add}_{out}$ deverá ser colocado no $\mathtt{data}_{in/out}$. Tal ciclo de leitura será considerado finalizado com sucesso quando o $\mathtt{conf}_{in}$ estiver ativo;

$\mathtt{conf}_{in}$: Sinal através do qual o MoIB indicará o sucesso da solicitação feita pelo MoConFiE;

$\mathtt{add}_{out}$: Sinal onde o endereço relativo é colocado;

$\mathtt{data}_{in/out}$: Sinal de entrada e saída de dados para o MoIB;

*Data Ready Out*put ($\mathtt{dr}_{out}$): Sinal ativado pelo MoConFiS indicando ao MoTraP que há envelopes na fila de entrada;

$\mathtt{rr}_{in}$: Sinal de solicitação do envelope ao MoConFiE;

*Data Out*put ($\mathtt{dataout}_{out}$): Sinal para passagem do envelope do MoConFiE para o MoTraP.

Quando o MoConFiE percebe a existência de um envelope na fila ele o lê, colocando-o num *buffer* interno, e só então indica ao MoTraP que a fila não está vazia, ativando o $\mathtt{dr}_{out}$. Assim, quando o $\mathtt{rr}_{in}$ é ativado o MoConFiE tem condições de passar todo o envelope, composto por 64 bits, para o MoTraP em 4 ciclos do relógio.

Periodicamente o MoConFiE verifica a existência ou não de envelopes na fila da seguinte forma: (1) solicita ao MoIB o conteúdo do apontador de escrita ($\mathtt{add}_{out}$ <= 0x0004 e $\mathtt{re}_{out}$ <= %1); (2) lê o apontador de escrita do $\mathtt{data}_{in/out}$ quando $\mathtt{conf}_{in}$ = %1; (3) compara-o com o valor do apontador de leitura, mantido internamente, se o apontador de leitura for igual ao de escrita será assumido que a fila está vazia, e o MoConFiE fará nova tentativa de leitura posteriormente. Caso contrário a leitura da fila será feita. A leitura da fila de entrada é feita da seguinte forma: (1) o MoConFiE colocará o valor do apontador de leitura no $\mathtt{add}_{out}$ e solicitará a leitura deste endereço ao MoIB ($\mathtt{re}_{out}$ <= %1); (2) com a ativação do $\mathtt{conf}_{in}$ os dados disponíveis no $\mathtt{data}_{in/out}$ (16 bits) serão guardados num *buffer* interno; (3) o apontador de leitura será incrementado de 2; (4) os passos (1), (2) e (3) serão repetidos mais três vezes; e (5) o valor do apontador de leitura será escrito no endereço relativo 0x0000 ($\mathtt{data}_{in/out}$ <= apontador de leitura, $\mathtt{add}_{out}$ <= 0x0000 e $\mathtt{we}_{out}$ <= %1), encerrando o ciclo de leitura.

Os apontadores, como pode ser visto na Fig. 3.8, possuem 32 bits de comprimento, sendo que o mais significativo será chamado de sinal e os 31 menos significativos de mantissa. Toda vez que o valor da mantissa for incrementado atingindo o tamanho máximo da fila, a mantissa será zerada e o sinal invertido.

MoConFiS

O MoConFiS cuidará do controle da fila de saída de um determinado usuário da MAC, encaminhando os envelopes da rede para o usuário quando a fila não estiver cheia. Se uma fila estiver cheia o MoConFiS entenderá que o usuário está ocupado, indicando tal fato para o MoDEn. Para controle da fila o MoConFiS utilizará os mesmos parâmetros utilizados pelo MoConFiE:

- Endereço inicial relativo da fila, sempre será 0x0000;
- Tamanho da fila: dependente do usuário;
- Largura da fila: 64 bits;
- Apontador de escrita: indica onde será feita a próxima escrita e
- Apontador de leitura: indica onde será feita a próxima leitura.

Aqui o apontador de escrita será controlado pelo MoConFiS e atualizado toda vez que se fizer necessário no endereço relativo 0x0004. Já o apontador de leitura será modificado apenas pelo usuário, e lido do endereço relativo 0x0000 pelo MoConFiS.

O MoConFiS só mandará os envelopes para a fila quando esta não estiver cheia.

Num ciclo de escrita será escrito todo um envelope.

O apontador de leitura, controlado pelo usuário, será lido a cada início de ciclo de leitura e comparado com o apontador de escrita, que é mantido pelo MoConFiS e, após o ciclo de escrita, este endereço será incrementado (de 64 bits) e colocado numa área da memória comum a disposição do usuário.

O MoConFiS possui uma interface com o respectivo MoIB e uma interface com o MoDEn esquematizadas na Fig. 3.13 e cujos sinais estão descritos a seguir:

we$_{out}$: Quando ativo indica ao MoIB que o dado contido no **data**$_{in/out}$ deve ser escrito no endereço relativo contido no **add**$_{out}$. O MoConFiS assumirá que o ciclo de escrita foi finalizado com sucesso quando o **conf**$_{in}$ estiver ativo;

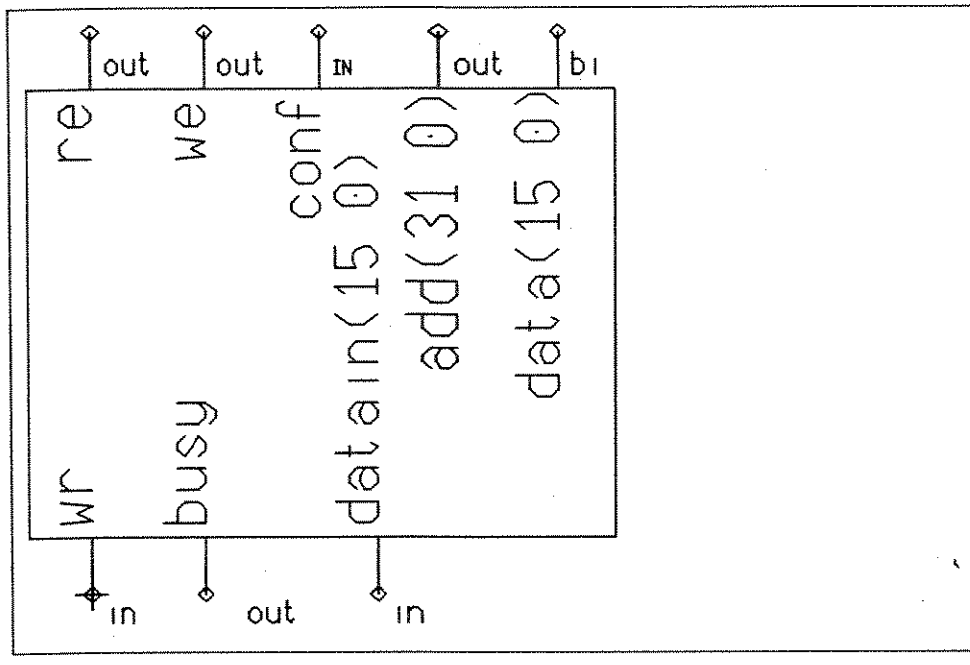


Figura 3.13: Símbolo do MoConFiS

re_{out}: Quando ativo indica ao MoIB que o dado do endereço relativo contido no add_{out} deverá ser colocado no $data_{in/out}$. Tal ciclo de leitura será considerado finalizado com sucesso quando o $conf_{in}$ estiver ativo;

$conf_{in}$: Sinal através do qual o MoIB indicará o sucesso da solicitação feita pelo MoConFiS;

add_{out}: Sinal onde o endereço relativo é colocado;

$data_{in/out}$: Sinal de entrada e saída de dados para o MoIB;

Busy Output ($busy_{out}$): Sinal ativado pelo MoConFiS indicando ao MoDEn que o usuário está ocupado, i.e., sem condições de receber novos envelopes;

wr_{in}: Sinal de indicação ao MoConFiS que um envelope está sendo encaminhado pelo *Data Input* ($datain_{in}$);

$datain_{in}$: Sinal para passagem do envelope do MoConFiS para o MoTraP.

O ciclo de escrita do MoConFiS é análogo ao ciclo de leitura do MoConFiE, ressalvando-se que o apontador controlado pelo MoConFiS é o apontador de escrita e que o MoConFiS assumirá que a fila está cheia quando a mantissa do apontador de escrita for igual à mantissa do apontador de leitura e os sinais forem invertidos.

O MoDEn encaminhará envelopes para o MoConFiS quando o sinal $busy_{out}$ do MoConFiS não estiver alto e fará isto em 4 ciclos de relógio, após o wr_{in} do MoConFiS ter sido ativado.

MoTraP

Ao MoTraP caberá o recebimento dos envelopes vindos dos MoConFiEs e do MoTI, controlando o fluxo destes envelopes, conforme a disponibilidade de *slots* vazios, seguindo a seguinte ordem de prioridade:

1. envelopes da APL[2];
2. envelopes da LLC/voz;
3. envelopes da LLC/dados e
4. envelopes da MAC.

[2]Maior prioridade

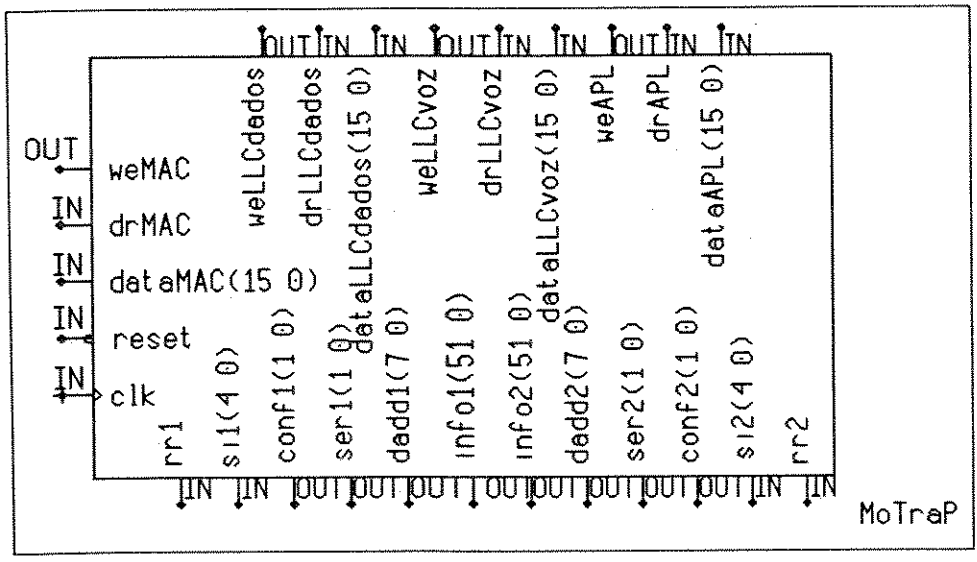


Figura 3.14: Símbolo do MoTraP

O MoTraP estará sempre apto a receber envelopes do MoConFiE da APL e manterá guardado num *buffer* interno o envelope que poderá:

- estar pleiteando um *slot* vazio para ser transmitido; ou
- estar aguardando a chegada do *slot* reservado para ele.

No primeiro caso o envelope será perdido, i.e., descartado, quando chegar o próximo envelope da APL e ele ainda não tiver sido enviado.

A chegada de um *slot* vazio será informada ao MoTraP, que estará capacitado para atender aos dois MoCAs simultaneamente, fazendo todo o processamento/encaminhamento necessário num tempo equivalente à metade do período de um *slot*, podendo processar, quando for necessário, dois *slots* consecutivamente.

O MoTraP será informado quando houver envelopes do MoConFiE da LLC/voz ou do MoConFiE da LLC/dados ou do MoTI, e requisitará estes envelopes quando não houver mensagem da APL aguardando um *slot* livre. Neste caso o envelope requisitado obedecerá ao esquema de prioridades citado no início desta seção.

O MoTraP possui interfaces com os MoConFiEs, com o MoTI e com os MoCAs esquematizadas na Fig. 3.14 e cujos sinais estão descritos a seguir:

we{APL,LLC/voz,LLC/dados,MAC}$_{out}$: Sinal através do qual o MoTraP indica ao seu usuário que lerá o envelope no **data**{APL,LLC/voz,LLC/dados,MAC}$_{in}$;

dr{APL,LLC/voz,LLC/dados,MAC}$_{in}$: Sinal através do qual o usuário indica ao MoTraP que há um envelope para ser encaminhado à rede;

data{APL,LLC/voz,LLC/dados,MAC}$_{in}$: Sinal através do qual os envelopes são encaminhados ao MoTraP;

rr{1,2}$_{in}$: Sinal através do qual o MoCA solicita um pacote ao MoTraP;

si{1,2}$_{in}$: Sinal de 5 bits através do qual o MoCA indica ao MoTraP se:

- %00XXX: O pacote desejado será um que esta aguardando um *slot* livre;
- %01XXX: O pacote desejado virá do MoTI, pois trata-se da análise de um pacote MAC;
- %1$b_3b_2b_1b_0$: O pacote desejado é do **tid** $b_3b_2b_1b_0$ da APL;

conf{1,2}$_{out}$: Sinal através do qual o MoTraP indica ao MoCA que:

- **%11:** o pedido está sendo respondido com sucesso,
- **%10:** não há pacote disponível a ser enviado,
- **%0X:** não há ainda resposta, indicando que o MoCA deverá aguardar;

ser{1,2}$_{out}$: Sinal através do qual os campos **pris** dos pacotes são encaminhados ao MoCA;

dadd{1,2}$_{out}$: Sinal através do qual os campos **tgts** dos pacotes são encaminhados ao MoCA;

info{1,2}$_{out}$: Sinal através do qual os campos **infos** dos pacotes são encaminhados ao MoCA.

Um MoCA{1,2} indicará, ativando o sinal **rr**{1,2}$_{in}$ do MoTraP que deseja um pacote do tipo especificado pelo sinal **si**{1,2}$_{in}$; se houver uma pacote deste para ser trasmitido, o MoTraP o colocará nos sinais **ser**{1,2}$_{out}$, **dadd**{1,2}$_{out}$ e **info**{1,2}$_{out}$ e ativará o **conf**{1,2}$_{out}$; se a resposta no **conf**{1,2}$_{out}$ não for positiva o MoCA assumirá que não há pacote deste tipo para ser transmitido.

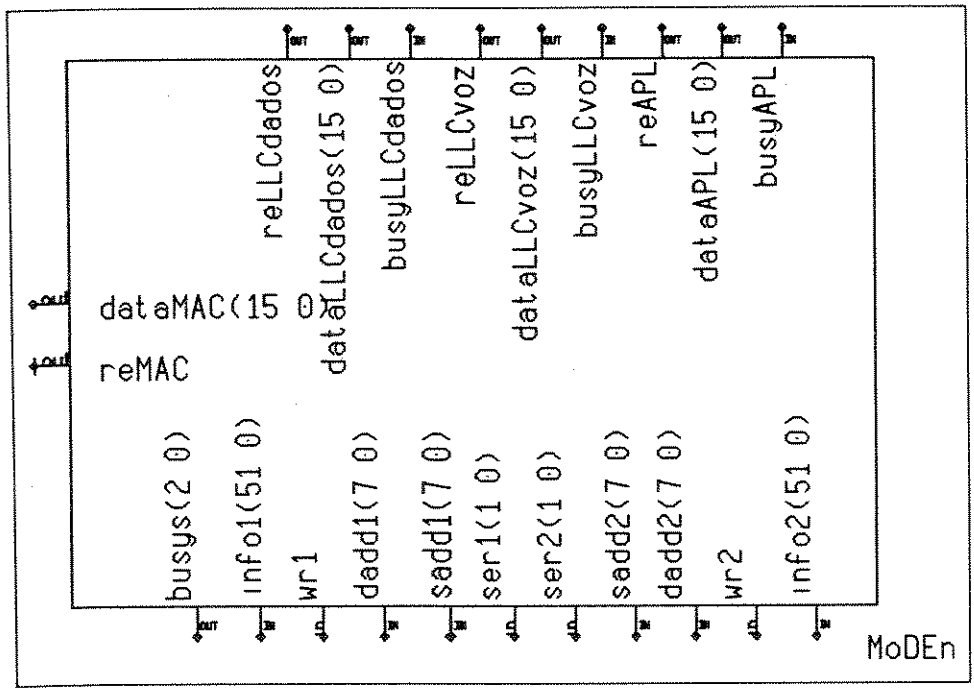


Figura 3.15: Símbolo do MoDEn

MoDEn

O MoDEn é o responsável pelo recebimento dos envelopes provenientes dos MoCAs e seu envio para o respectivo MoConFiS ou para o MoTI. Ele confirmará ao MoCA a aceitação do envelope ou a sua recusa.

O MoDEn possui interfaces com os MoConFiSs, com o MoTI e com os MoCAs esquematizadas na Fig. 3.15 e cujos sinais estão descritos a seguir:

re{APL,LLC/**voz**,LLC/**dados**,MAC}$_{out}$: Sinal através do qual o MoDEn indica ao seu usuário que há um envelope para ele no **data**{APL,LLC/voz,LLC/dados,MAC}$_{out}$, que será transferido em 4 ciclos do relógio;

busy{APL,LLC/**voz**,LLC/**dados**}$_{in}$: Sinal através do qual o MoDEn é informado de que o usuário não está disponível para receber o envelope;

data{APL,LLC/**voz**,LLC/**dados**,MAC}$_{out}$: Sinal através do qual o envelope é encaminhado ao {MoConFiS,MoTI};

busys$_{out}$: Sinal através do qual o MoDEn indica ao MoCA quais usuários não estão disponíveis para receber envelopes;

wr{1,2}$_{in}$: Sinal através do qual ao MoCA é indicado que um dado pacote estará disponível nos próximos 2 ciclos de relógio;

Destination Address Input (**dadd**$_{in}$): Endereço do nó local (pode ser o endereço 255 de *broadcast*). Encaminhado ao MoDEn e dependente do envelope a ser trasportado;

Information Input (**info**$_{in}$): Sinal através do qual o campo **info** do pacote é encaminhado ao MoDEn;

Source Address Input (**sadd**$_{in}$): Sinal através do qual o endereço do nó remoto é encaminhado ao MoDEn conforme encontrado no pacote recebido;

ser{1,2}$_{in}$: Sinal através do qual o MoDEn recebe o símbolo **pri**;

O MoTI estará sempre apto a receber pacotes, não havendo portanto a necessidade de um sinal **busyMAC**$_{in}$.

O MoDEn receberá do MoCA apenas os pacotes para os usuários que estejam aptos a recebê-los. Para tal o MoCA dispõe da informação contida no **busys**$_{out}$.

Ao receber um pacote do MoCA o MoDEn encaminhará o envelope imediatamente ao MoConFiS correspondente ou ao MoTI, não havendo a necessidade de "buferização".

MoTI

O MoTI cuidará da iniciação do sistema e dos testes garantindo a integridade dos anéis.

No chamado processo de iniciação da MAC, o MoTI enviará mensagens para todas as outras MACs, buscando contar a quantidade de nós ativos na rede, indicando, em seguida, esta quantidade à PHY, para que os atrasos necessários sejam inseridos em cada nó. Apesar de ser chamada de mensagem de iniciação da MAC esta mensagem será periodicamente repetida, permitindo mudanças dinâmicas no número de nós em cada anel.

O MoTI enviará um pacote para cada anel quando a MAC foi iniciada ou quando houver um rompimento de algum enlace ou mesmo quando não houver uma estação monitora na rede. Este pacote, esquematizado pela Fig. 3.16, permitirá:

- o teste da integridade de um anel;

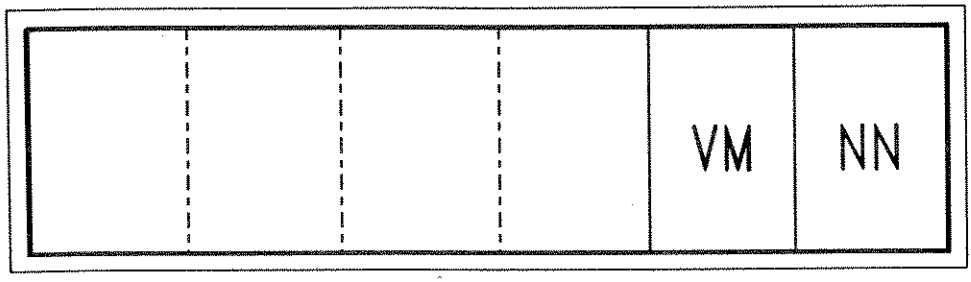


Figura 3.16: Pacote da *MAC*

- contar a quantidade de nós num anel; e
- definir quem será a estação monitora.

O campo `info` do pacote da MAC possuirá dois sub-campos:

***N*umber of *N*odes (`nn`):** Será usado para contar a quantidade de nós da rede; e

***V*otes to *M*onitor (`vm`):** Será usado para solicitar para ser a estação monitora.

A contagem de nós da rede será feita incrementando o valor em `nn`, que teve como valor inicial o 0, a cada nó por onde o pacote passará; assim, ao retornar à estação origem; o `nn` conterá o número de nós da rede.

O pedido para ser monitor de uma rede é feito da seguinte forma. A estação candidata lança um pacote MAC com o `vm` contendo $r + \overline{n}$, i.e. o tempo, em número de *slot*s, que o pacote levará para chegar na próxima estação e inicia um contador interno, o *C*ounter of *V*otes to *M*onitor (`cvm`), com este valor. Os parâmetros $\overline{n}$ e r, definidos pelas equações 3.4 e 3.5, são funções do N.

$$\overline{n} \triangleq \left\lfloor \frac{n}{N} \right\rfloor \tag{3.4}$$

$$r \triangleq n - N \cdot \overline{n} \tag{3.5}$$

onde:

$\overline{n}$: número de *slot*s por estação;

n: número de *slot*s do anel;

N: número de estações ativas no anel;

r: resto do $\overline{n}$

A cada pacote que passa pelo nó candidato a monitor o **cvm** é incrementado de um e por cada nó onde o pacote do candidato a monitor passa ele será incrementado de $\overline{n}$, se o nó em questão não for também um candidato. Os nós candidatos que receberem um pacote MAC de outro nó com o **vm** $\neq 0$ compararão o valor do **vm** com o do seu **cvm** e se:

vm < **cvm**: indica que o nó que está analisando o pedido já fez um pedido para ser monitor a mais tempo do que o nó remetente do pacote, assim o nó analisador zerará o **vm**, vetando a candidatura a monitor do nó remetente do pacote;

vm = **cvm**: indica que o nó que está analisando o pedido fez um pedido para ser monitor no mesmo instante que o nó remetente do pacote, assim o nó analisador comparará o seu endereço de MAC com o endereço de MAC do originador do pacote. O endereço de menor valor nominal será mantido como candidato;

vm > **cvm**: indica que o nó que está analisando o pedido fez um pedido para ser monitor posteriormente ao nó rementente do pacote, assim o nó analisador zerará o seu **cvm**, retirando a sua candidatura, e passará o pacote para o próximo nó, após incrementar o conteúdo do **vm** de $\overline{n}$.

Dissemos a alguns parágrafos atrás que os pacotes da MAC serão enviados quando um enlace for rompido. Este rompimento será detectado pela PHY que, não mais recebendo sinais da rede, assumirá que o enlace rompeu-se e indicará o fato para a MAC. Porém a PHY só tem condições de indicar um rompimento ocorrido num enlace que a ligue a um nó vizinho, então o MoTI enviará pacotes para a rede também quando notar que outras MACs o estão fazendo. Tal fato não sobrecarregará a rede porque a prioridade dos pacotes da MAC é a menor, e também porque teremos, num determinado instante de tempo, no máximo, um pacote de cada MAC na rede.

Dissemos também que os pacotes da MAC seriam enviados quando não houvesse uma estação monitora na rede. Este fato é notado por uma MAC quando os seus pacotes não estiverem retornando com o **m** ativo.

O MoTI possui interfaces com o MoTraP, com o MoDEn, com os MoCAs e com a PHY esquematizadas na Fig. 3.17 e cujos sinais estão descritos a seguir:

rr$_{in}$: Sinal através do qual o MoTraP indica que o MoTI deverá, nos próximos 4 ciclos de relógio, enviar o envelope pelo **dataout**$_{out}$;

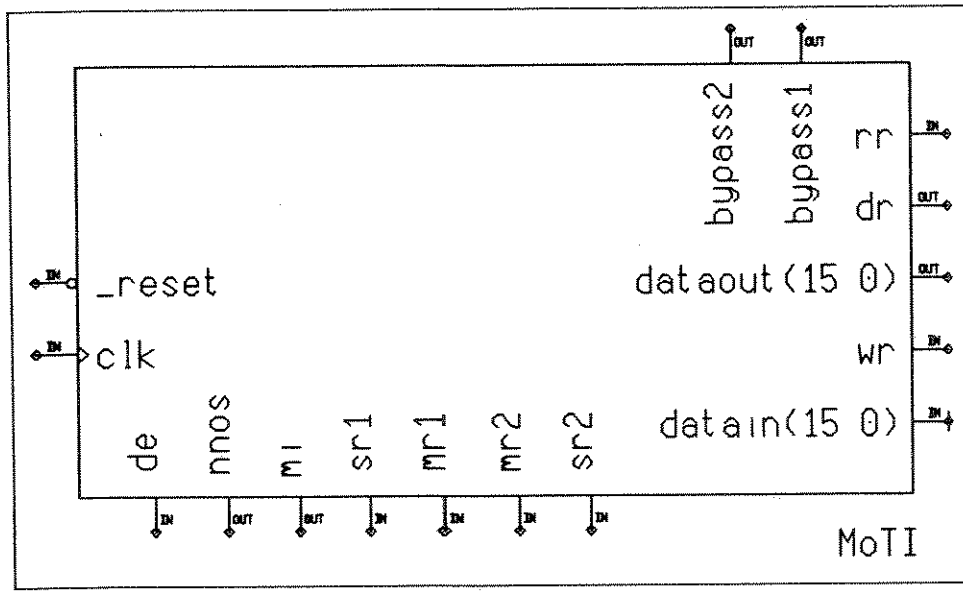


Figura 3.17: Símbolo do MoTI

dr$_{out}$: Sinal através do qual o MoTI indica ao MoTraP que possui um envelope para ser enviado à rede;

dataout$_{out}$: Sinal através do qual os envelopes são enviados do MoTI para o MoTraP;

wr$_{in}$: Sinal através do qual o MoDEn indica ao MoTI que será encaminhado um envelope nos próximos 4 ciclos de relógio, via **datain**$_{in}$

datain$_{in}$: Sinal através do qual o MoTI recebe os envelopes do MoDEn;

sr{1,2}$_{in}$: A cada *slot* recebido com um pacote de algum MoTI o MoCA{1,2} indicará o fato ao MoTI por este sinal;

mr{1,2}$_{in}$: O retorno de um pacote ao nó original, sem que o campo **m** esteja ativo, indicará uma provável ausência de uma estação monitora. Tal fato é indicado ao MoTI pelo **mr**{1,2}$_{in}$;

***M*onitor *I*ndicator *Out*put (mi$_{out}$)**: Sinal através do qual o MoTI indica aos MoCAs e às PHYs que o nó corrente é o monitor;

*N*úmero de *Nós* *Out*put (nnos$_{out}$): Sinal através do qual o MoTI indica às PHYs o atraso de propagação para o próximo nó, em tamanho do *slot*, este valor será $\overline{n}$ para todas as estações e $\overline{n}+r$ para a monitora;

*D*enyed *E*nlace (de$_{in}$): Sinal através do qual a PHY indica à MAC que um determinado enlace está inacessível;

By Pass{*1,2*} *Out*put (bypass{1,2}$_{out}$): Quando %1 indica ao MoCA{1,2} que ele deve trabalhar em modo *by pass*, i.e., os pacotes não devem ser analisados, apenas passados adiante. Este pino deverá inibir a solicitação de envelopes ao MoTraP por parte de determinado MoCA.

O MoTI só terá acesso aos envelopes provenientes do nó local, porém será sempre informado pelo MoCA quando houver algum pacote de algum outro MoTI na rede, o que indicará que há um provável rompimento de algum ponto da rede (enlace ou nó) o que fará com que o MoTI envie um pacote para rede, se já não o tiver feito. Os algoritmos de contagem do número de nós da rede e de escolha da estação monitora necessitam que pacotes proveninetes de MoTIs de outras MACs da rede sofram um certo processamento. Este processamento será provido pelo MoCA, evitando assim atrasos desnecessários no processamento dos *slots*.

MoCA

Ao MoCA caberá o recebimento, processamento e envio dos *slots* de/para o anel. Há dois MoCAs, um para cada anel.

Ao receber um *slot* da PHY, o MoCA terá o tempo correspondente ao tamanho do *slot*, 7,3μs, para reconhecer se este *slot* foi remetido pelo nó local ou não. Em caso positivo ele verificará qual o tipo de serviço prestado: tipo 1 ou tipo 2. Se for do tipo 1 o MoCA solicitará o próximo envelope de voz referente a esta chamada ao MoTraP. Se for do tipo 2 o MoCA verificará o campo re encaminhando a resposta encontrada ao MoDEn e liberará o *slot* como vazio para o próximo nó.

A cada *slot* recebido do tipo 2 o MoCA conferirá o chk e em caso do *slot* estar corrompido será descartado, tornando o *slot* vazio. Esta perda de informação deverá ser coberta pelas camadas superiores.

O MoCA, quando receber um *slot* vazio, solicitará ao MoTraP um envelope para ocupar o *slot* e, se o MoTraP não tiver nenhum envelope para encaminhar ao MoCA o *slot* será encaminhado como vazio para o próximo nó.

O relógio do sistema será proveniente da camada física e terá um ciclo de 100 ns.

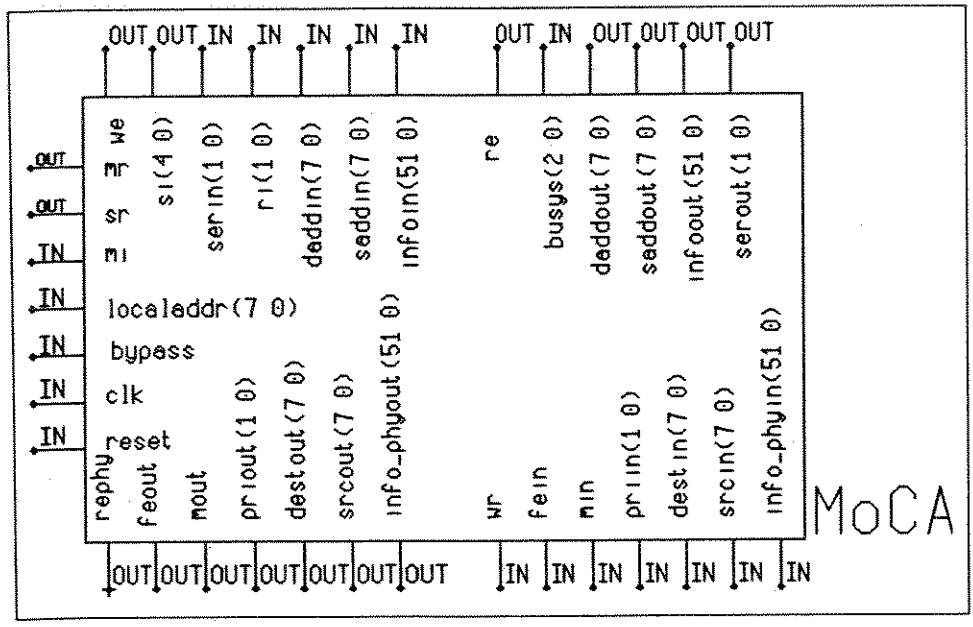


Figura 3.18: Símbolo do MoCA

O MoCA possui interfaces com o MoTraP, com o MoDEn, com a PHY esquematizadas na Fig. 3.18 e cujos sinais estão descritos a seguir:

dadd$_{in}$: Endereço do nó destino. Encaminhado pelo MoTraP e dependente do envelope a ser trasportado;

info$_{in}$: É a informação do usuário carregada pelo envelope;

Read Indicator Input **(ri$_{in}$):** Após solicitar um envelope ao MoTraP, o MoCA aguardará pela resposta vinda pelo **ri$_{in}$**, que pode ser:

%11: o pedido está sendo respondido com sucesso,

%10: não há pacote disponível a ser enviado,

%0X: não há ainda resposta, indicando que o MoCA deverá aguardar;

sadd$_{in}$: Endereço do nó origem. Será fixo (podendo ser definido pelo MoTI).

Service Input **(serin$_{in}$):** Indica o serviço que está sendo prestado:

%00: MAC,

%01: LLC/dados,

%10: LLC/voz e

%11: APL;

Slot Indicator Output **(si$_{out}$):** Indica o tipo de *slot* disponível ao MoTraP:

%00XXX: *slot* vazio,

%01000: *slot* reservado à própria MAC,

%1$b_3b_2b_1b_0$: *slot* reservado a APL, sendo $b_3b_2b_1b_0$ o **tid**;

we$_{out}$: Quando 1 indica ao MoTraP que há um *slot* do tipo descrito por **si$_{out}$**, aguardando para ser utilizado. O MoCA aguardará por uma resposta do MoTraP através do **ri$_{in}$**;

busys$_{in}$: Indica quando um usuário está ocupado não estando diponível para receber envelopes:

%XX1: LLC/dados não disponível para receber mensagens,

%X1X: LLC/voz não disponível para receber mensagens e

%1XX: APL não disponível para receber mensagens;

***Destination Address Output* (daddout):** Endereço do nó local (pode ser o endereço 255 de *broadcast*). Encaminhado ao MoDEn e dependente do envelope a ser trasportado;

reout: Indica ao MoDEn quando os dados (daddout, *Source Address Output* (saddout), *Information Output* (infoout) e *Service Output* (seroutout)) estão prontos para serem lidos;

infoout: Sinal através do qual o campo **info** do pacote é encaminhado ao MoDEn;

saddout: Endereço do nó remoto. Encaminhado ao MoDEn conforme encontrado no pacote recebido;

seroutout: O MoCA indicará o tipo de envelope ao MoDEn através do seroutout:

%00: MAC,

%01: LLC/dados,

%10: LLC/voz,

%11: APL;

***Local Address* (localaddr):** Endereço da MAC local;

***Full/Empty Input* (feain):** Contém o valor do campo **f/e** do pacote recebido;

***Destination Input* (destin):** Contém o valor do campo **tgt** do pacote recebido;

***Information Input to PHY* (info_phyin):** Contém o valor do campo **info** e do campo **re** (quando existir) do pacote recebido;

***Monitor Input* (min):** Contém o valor do campo **m** do pacote recebido;

***Priority Input* (priin):** Contém o valor do campo **pri** do pacote recebido;

***Source Input* (srcin):** Contém o valor do campo **tgt** do pacote recebido;

wrin: Quando %1 indicará ao MoCA que os dados nas portas feain, min, priin, destin, srcin e info_phyin estão prontos para serem lidos;

*F*ull/*E*mpty *Out*put (`feout`$_{out}$): Contém o valor do campo `f/e` no pacote transmitido;

*Des*tination *Out*put (`destout`$_{out}$): Contém o valor do campo `tgt` no pacote transmitido;

*Inf*ormation *Out*put to *PHY* (`info_phyout`$_{out}$): Contém o valor do campo `info` e %00 no campo `re` (se existir) no pacote transmitido;

*M*onitor *Out*put (`m`$_{out}$): Contém o valor do campo `m` no pacote transmitido;

*Pri*ority *Out*put (`priout`$_{out}$): Contém o valor do campo `pri` no pacote transmitido;

*Sou*rce *Out*put (`srcout`$_{out}$): Contém o valor do campo `src` no pacote transmitido; e

*R*ead *E*nable *Out*put to *Phy*sical (`rePHY`$_{out}$): Quando %1 indica à PHY que os dados nas portas `feout`$_{out}$, `m`$_{out}$, `priout`$_{out}$, `destout`$_{out}$, `srcout`$_{out}$ e `info_phyout`$_{out}$ estão prontos para leitura. O `rePHY`$_{out}$ ficará em %1 por dois ciclos de relógio;

*S*lot *R*eceived *Out*put (`sr`$_{out}$): A cada *slot* recebido com um pacote de algum MoTI o MoCA indicará o fato ao MoTI por este sinal;

*M*onitor *R*equest *Out*put (`mr`$_{out}$): O retorno de um pacote ao nó original, sem que o campo `m` esteja ativo, indicará uma provável ausência de uma estação monitora. Tal fato é indicado ao MoTI pelo `mr`$_{out}$;

*M*onitor *I*ndicator *In*put (`mi`$_{in}$): Sinal que quando %1 indica ao MoCA que o nó é monitor.

*By Pass In*put (`bypass`$_{in}$): Quando %1 indica ao MoCA que nenhum pacote deve ser processado como sendo do nó ou para o nó corrente. Este pino possibilita a utilização da MAC em *loopbacks*.

3.7 Conclusões

Neste capítulo apresentamos uma rápida descrição da RALFO, rede proposta em ([GM89]), como uma rede tipo anel de Cambridge modificada, visando ser uma rede de serviços integrados, oferecendo aos usuários a possibilidade de transmitir dados e voz numa mesma rede, utilizando anel duplo e fibra óptica como meio físico. Apresentamos a nossa proposta para a sub-camada MAC da RALFO, definindo o quadro, sua estrutura

em módulos e especificando funcionalmente cada um destes módulos. Este capítulo dará subsídios para o capítulo da implentação da MAC a nível de esquemático e na linguagem VHDL numa metodologia *top-down*.

Capítulo 4

A Metodologia de Projeto

Neste capítulo fazemos uma introdução ao processo de projeto de circuitos eletrônicos, notadamente os digitais. Na seção 4.1 descrevemos as principais tecnologias de circuitos digitais. Na seção 4.2, vemos diversas formas de modelamento e descrição de circuitos. Discutiremos duas formas de abordagem de projeto na seção 4.3: *botton-up* e *top-down*. Na seção 4.4, apresentamos o ciclo de projeto de circuitos. Na seção 4.5, apresentamos alguns conceitos importantes para projetos digitais. Na seção 4.6, apresentamos a VHDL, a captura esquemática e a *Q*uick*P*art *T*able (qpt) como exemplos de formas de descrição de circuitos. Finalmente, na seção 4.7, descrevemos o processo adotado neste projeto.

Aqui procuramos registrar a experiência adquirida por nós nesta área durante a confecção deste projeto. Portanto estamos sempre utilizando como exemplo as ferramentas da MGC, versão 8.*X*, para projeto de circuito. [Kru93] é um trabalho dedicado à análise de metodologias de projeto de micro-eletrônica, onde podem ser encontrados dados interessantes que justificam a atual tendência da área. [MGC93a] descreve mais detalhadamente os conceitos envolvidos no projeto de circuitos. [Tau84] é uma literatura básica sobre projeto de circuitos digitais.

4.1 Tecnologias de Projetos Eletrônicos

O advento da micro-eletrônica propiciou a proliferação do uso de circuitos digitais nas mais diversas técnicas. A modularidade e maior tolerância a variações dos sinais eletrônicos favorecem a utilização da

eletrônica digital, onde trabalhamos com apenas dois níveis lógicos. Das mais diversas técnicas para implementação de circuitos, destacaremos as seguintes: *P*rinted *C*ircuit *B*oard (PCB), *P*rogrammable *L*ogic *A*rray (PLA), *F*ield *P*rogramable *G*ate *A*rray (FPGA), *standard cell* e *full custom*. Aqui elas foram listadas em ordem crescente de complexidade. O uso de técnicas mais complexas, como por exemplo *full custom*, é justificado para projetos grandes, com alto grau de integração, i.e., elevado número de componentes, ou para projetos que almejam atingir o mercado de massa, pois a produção em larga escala de circuitos integrados dedicados diminui o custo final do produto e aumenta a sua confiabilidade.

Como o nome já sugere, o PCB utiliza-se de placas de circuito impresso sobre as quais são soldados os componentes, geralmente *C*ircuitos *I*ntegrados (CIs) e alguns componentes discretos, como osciladores, capacitores e resistores. Tal técnica permite a utilização de componentes comerciais encontrados a baixo custo no mercado, o que torna-a apropiada para a produção em baixa escala.

Na classe de projeto a base de *P*rogrammable *L*ogic *D*evices (PLDs) destacaremos duas tecnologias, as PLAs e as FPGAs. Os PLDs são aqueles dispositivos cuja funcão é especificada via uma programação. As PLAs diferenciam-se das FPGAs por necessitarem de uma programação feita em tempo de implementação do circuito, i.e., durante o processo de sua fabricação. Uma PLA uma vez programada não pode ser mais reprogramada. Já as FPGAs são programadas na iniciação do circuito, podendo ter sua lógica modificada a qualquer instante. Uma introdução aos PLDs pode ser encontrada em [Alf89, Col87, Fre87a, Fre87b, Mar87, Rob87].

O *standard cell* utiliza-se de células básicas de circuitos, já validadas, como unidades do projeto. Assim um projeto *standard cell* consiste na interligação des células básicas.

Um projeto *full custom* é totalmente desenvolvido, a nível de circuito integrado, para a sua aplicação específica. Tal técnica embora cara e demorada, produz circuitos mais eficientes.

4.2 Abstrações de Circuitos

Podemos descrever um circuito de diversas formas (Fig. 4.1). A descrição de mais alto nível é a comportamental, seguida das descrições *R*egister *T*ransfer *L*evels (RTLs), estruturais e físicas.

A descrição comportamental descreve o circuito em alto nível, podendo aqui ser utilizadas máquinas de estado, diagramas de fluxo de dados, linguagens de programação sequencial, *S*pecification and *D*escription *L*anguage (SDL), Estelle, Lotos ou mesmo VHDL.

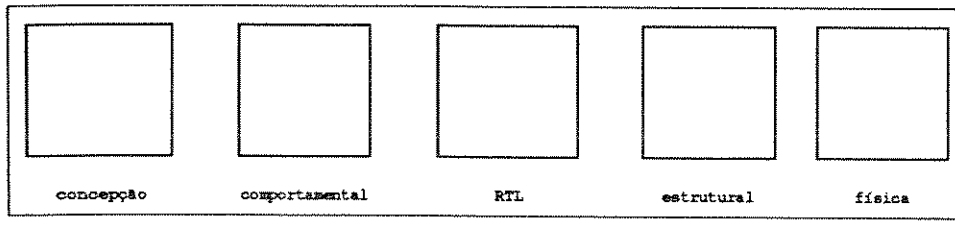


Figura 4.1: Tipos de Abstrações

A descrição a nível RTL estabelece células básicas como sendo registradores, somadores, memória, etc., e a lógica de ligação entre elas.

A descrição estrutural chega a nível de portas lógicas básicas como *and*, *or* e *nor* e é a descrição mais baixa antes da descrição física.

A descrição física estabalece o *layout* do circuito, a nível de máscara de circuito integrado.

4.3 Metodologias de Projeto

Podemos abordar um projeto a partir das células básicas, que são agrupadas em blocos cada vez mais complexos até a especificação do projeto final. Tal técnica possui a falha de não dispormos de mecanismos de validação do projeto como um todo antes do final da primeira especificação. Esta metodologia é conhecida como *botton-up*.

A abordagem oposta à *botton-up* é a *top-down*, onde descrevemos o projeto em alto nível, por exemplo, comportamentalmente, validamos esta descrição e a refinamos até o nível físico. Tal técnica hoje é facilitada pela existência de ferramentas de síntese, como o *AutoLogic* (`autologic`) ([MGC93b]) da MGC.

4.4 Ciclos de Projeto

O ciclo de projeto de circuito vai depender da tecnologia utilizada. Assim um projeto PCB difere-se de um projeto *full custom*. Mas, embora de formas diferentes, todos eles envolvem uma fase de criação do projeto, uma fase de checagem, uma fase de análise e verificação do projeto, uma fase de implementação, uma prototipagem e finalmente a produção. Buscando diminuir os custos, aumentar a velocidade e confiabilidade do projeto, e suprir a necessidade crescente de circuitos complexos e eficientes do mercado, há uma tendência da adoção de ferramentas de EDA, que além

de prover o projetista com ferramentas de suporte direto a cada fase do ciclo de projeto, ainda propiciam a sua execução concorrente, permitindo o trabalho em paralelo de uma equipe de projetistas. As ferramentas de EDA mais modernas, como a utilizada neste trabalho, suportam ainda o trabalho dos gerentes de projetos, permitindo-os obter a cada instante parâmetros importantes para a gerência, como medidas do volume do projeto, quantidade de componentes/células básicas utilizados em cada parte do circuito, custo total estimado, etc.

Como exemplo veremos com mais detalhes o ciclo de um projeto ([MGC93a]). O ciclo, esquematizado na Fig. 4.2, começa com a especificação, prossegue pela criação do projeto eletrônico e checagem, quando havendo erros poderá retornar à fase de criação ou, não havendo erros, passar para a fase de verificação e análise. Da fase de verificação e análise ele pode retornar para a criação, quando os resultados ainda não satisfazem a especificação, ou passar para a fase de fabricação. O projeto pode sofrer então novo processo de testes a nível do fabricante, sendo remetido, se aprovado, com as modificações, para o teste final, ou então retornar para o nível de verificação e análise ou mesmo para a fase inicial de criação.

A fase de criação pode envolver descrições em alto nível, comportamental ou RTL, utilizandos-se de linguagens de descrição formal. Neste caso há um processo de síntese, que consiste em transformar a descrição em alto nível para nível lógico. Tal tarefa é hoje facilitada por ferramentas de síntese que permitem, por exemplo, a conversão automática de VHDL para uma descrição lógica, como o `autologic` da MGC. Pode ser utilizada aqui também a captura esquemática, que permite ao usuário descrever o seu circuito via desenhos que o esquematize. É importante notar que em ambientes integrados, como o usado neste trabalho, é possível a mistura de técnicas, i.e., é possível fazer uma parte do projeto com VHDL e outra parte via captura esquemática. O projetista pode criar um *componente*, descrevendo-o funcionalmente em VHDL ou via esquemáticos, criando um símbolo gráfico que represente este componente e depois utilizá-lo em outro esquemático, da mesma forma que utiliza componentes de uma biblioteca do sistema.

A fase de criação do projeto é feita utilizando-se do *D*esign *A*rchitect (`da`) ([MGC93e, MGC93d]), onde há um ambiente para descrição VHDL, com um editor voltado para a linguagem, documentação, gerador automático de símbolos a partir de uma descrição VHDL e geração automática de entidades VHDL a partir de uma descrição gráfica do componente, o símbolo. Há também a possibilidade de geração automática de uma descrição VHDL para células básicas como *Flip-flops*, *latches*, multiplexadores, demultiplexadores, memória, etc. A captura esquemática também é feita

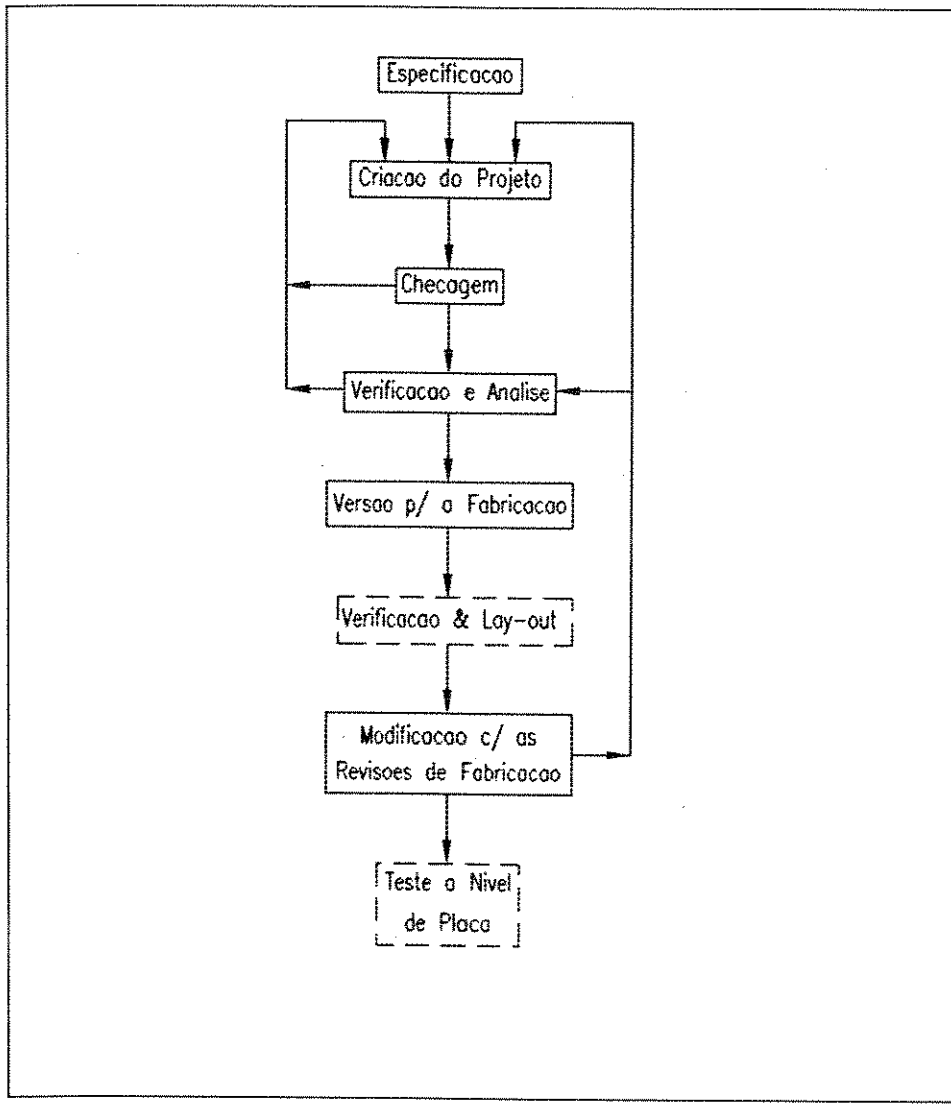


Figura 4.2: Ciclo de Projeto ASIC

via da, que possui uma série de bibliotecas prontas para diversas tecnologias de projeto além de uma biblioteca genérica, a *St*andar*d* *Lib*rary (`std_lib`), usada por nós nas especificações feitas por captura esquemática ([MGC93l]).

A fase de checagem do projeto consiste em verificar se a descrição do circuito está infringindo alguma regra, que impossibilite o seu entendimento em etapas subseqüentes do projeto. Aqui é verificado se o projeto está infringindo alguma limitação (*constraint*) tecnológica, ou imposta no próprio projeto. Num circuito descrito via esquemáticos é preciso verificar se não há sinais com nomes duplicados, se não há pinos inadivertidamente não utilizados, se os barramentos quando interligados, possuem as mesmas dimensões, etc. É necessário a criação de um *Desegn Viewpoint*, que pode ser entendido como a parte do projeto responsável pelo controle de suas propriedades. É necessário certificar-se de que os componentes utilizados estão acessíveis. É possível também checar se regras elétricas inerentes do processo físico estão sendo seguidas corretamente. Podemos entender também que a análise sintáxica de uma descrição VHDL, feita pelo compilador VHDL, faz parte da fese de checagem.

Todo este processo é feito em um único passo dentro do da, bastando ao projetista selecionar a opção desejada no *menu* `Check`.

Na fase de verificação e análise submete-se o projeto a simulações que visam verificar o seu comportamento, detectando falhas no projeto para que possam ser corrigidas. Nesta fase costuma-se criar vetores de teste como estímulos para a simulação, o que é possível com o uso de VHDL ou também de comandos do simulador lógico, *QuickSim* II (`quicksim`), automatizados via *A*dvanced *M*ulti-*P*urpose *L*anguage (AMPLE), uma linguagem utilizada para automação do uso das ferramentas da MGC. A simulação lógica pode ser feita usando três tipos de atraso: atraso unitário, atraso estimado, ou atraso de propagação. A simulação é feita com atraso unitário quando deseja-se testar a lógica do circuito, não levando em conta atrasos como tempo de resposta de uma porta, atrasos de mudanças de estado de um sinal, etc. A simulação utiliza atraso estimado, quando deseja-se verificar o funcionamento do circuito em condições mais próximas da realidade, i.e., com os atrasos inerente de cada componente e porta. É possível especificar no da atrasos mínimos, máximos e típicos de cada componente de uma biblioteca, usada em esquemáticos. Na simulação é possível trabalhar com qualquer um dos três valores. Com relação ao atraso estimado é possível especificar equações que representem o atraso desejado. Finalmente, utilizamos o atraso de propagação após recebermos do fabricante a informação de atraso estimado em cada linha.

A nossa participação neste projeto encerra-se nesta fase, pois objetivamos apenas a sua especificação, e não a sua síntese. Assim fizemos apenas simulações com atraso unitário e com atraso estimado.

Em circuitos onde necessitamos detectar os caminhos onde o tempo é crítico, utilizamos o *QuickPath* (quickpath) ([MGC93j]). Temos também ferramentas para análise de falhas, detectadas pela aplicação de um conjunto de vetores de teste: O *QuickGrade* II (quickgrade) e o *QuickFault* (QuickFault) ([Men93]). O quickgrade faz uma análise estatística dos erros e o QuickFault faz uma análise determinística. É possível também, nesta fase, a análise da dissipação de calor assim como a simulação do CI numa placa.

Antes da fase de fabricação, geralmente feita por terceiros, o projeto deve ser preparado para ser encaminhado para o fabricante. Aqui, se ainda não tiver ocorrido, pode haver uma nova fase de checagem, utilizandos-se o *QuickCheck* (QuickCheck) ([MGC93i]). Então é preciso criar uma lista de ligações (*netlist*) para ser encaminhada para o fabricante. A MGC provê a possiblidade de criar uma série de *netlists*, conforme a especificação do fabricante: *E*lectronic *D*esign *I*nterchange *F*ormat (EDIF), feito pelo *E*DIF *N*etlister *Write* (enwrite) ([MGC93h]); *V*erilog *Net*list (V-Net), feito pelo da; *Lsim Net*list (LsimNet), também feito pelo da; *Spice Net*list (SpiceNet), feito pelo da. É possível também o uso do *D*esign *F*ile *I*nterface (DFI) ou do *D*esign *D*ataport (DDP) para gerar *netlist*. Na fase de preparação para fabricação é preciso converter para o formato do fabricante os vetores de teste. É preciso fornecer ao fabricante também um diagrama com os nomes e disposição dos pinos do componente. Maiores detalhes sobre este processo podem ser encontrados em [MGC93f].

O fabricante, a seguir recebe do cliente os dados necessário para gerar o *layout* físco do *A*pplication-*s*pecific *I*ntegrated *C*ircuits (ASIC). Depois de gerar o *layout*, o fabricante fornece também ao cliente os dados relativos à capacitância e atraso inseridos nesta fase no projeto. É possível também ao fabricante fornecer uma *netlist* e vetores de testes atualizados ao cliente.

A fase seguinte consiste em atualizar o projeto com os dados fornecidos pelo fabricante. Para tal fim é possível a leitura de *netlists* em formato EDIF através do *E*DIF *N*etlister *Read* (enread) ([MGC93h]). Novas simulações incorporando os atrasos e capacitâncias do processo físico são necessárias. Utiliza-se o *D*esign *V*iewpoint *E*ditor (dve) para incorporar estes dados no *design viewpoint* do projeto ([MGC93g]).

Após o recebimento do ASIC é possível o seu teste já em uma placa, utilizando-se dos vetores de teste do projeto.

Analisando o ciclo de um projeto cuja tecnologia utilizada seja PCB (Fig. 4.3), veremos que as três primeiras fases são coincidentes com o ciclo de projeto ASIC, ressalvando-se que as primitivas utilizadas na descrição do circuito, via captura esquemática, por exemplo, podem ser de componentes comerciais, quando já tem-se definido no início do projeto a família que será adotada. Há aqui, terminada a fase de validação e análise do projeto,

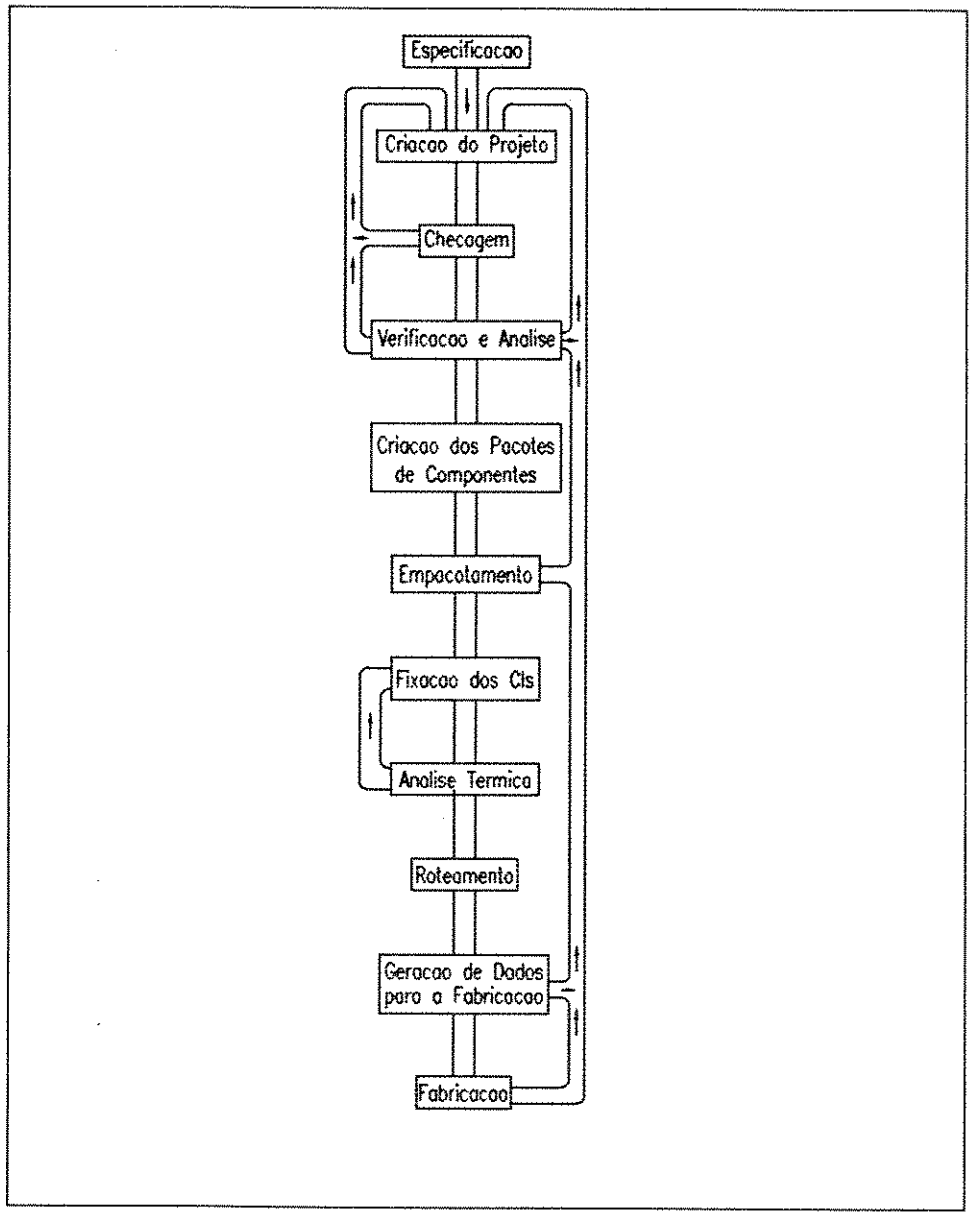


Figura 4.3: Ciclo de Projeto PCB

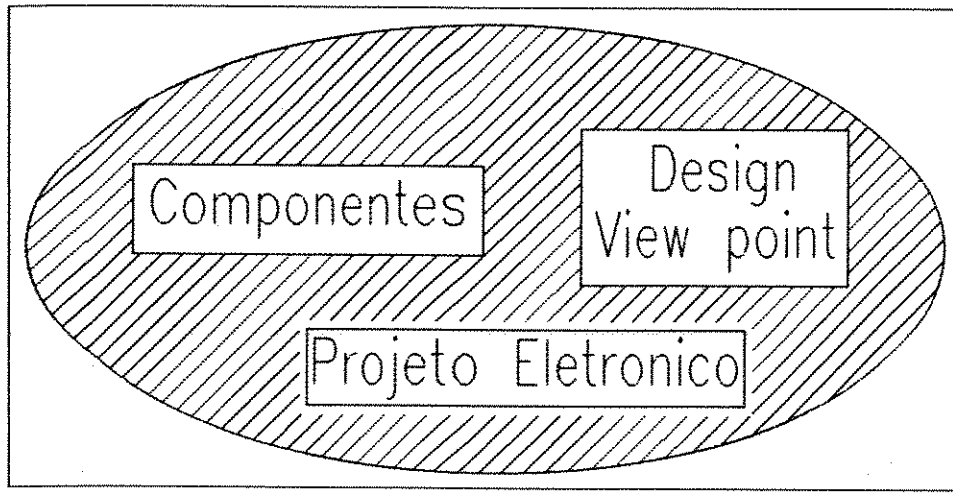


Figura 4.4: Projeto Eletrônico

fases características da tecnologia PCB. São elas: empacotamento dos componentes, atribuição de símbolos lógicos aos componentes, fixação dos componentes na placa, análise térmica, roteamento e finalmente a geração das informações necessárias à fabricação. O empacotamento corresponde a agrupar diversos símbolos num único *chip*, de acordo com os componentes encontrados no mercado. O roteamento consiste em especificar as trilhas que interligarão os diversos *chips* segundo o projeto descrito no(s) esquemático(s).

A nossa opção em fazer especificação deste projeto utilizando como componentes básicos, componentes genéricos, possibilitará a adoção que qualquer tipo de tecnologia no prosseguimento do projeto, i.e., sua síntese.

4.5 Conceitos Envolvidos num Projeto

Um projeto eletrônico, ou simplesmente projeto, consiste na descrição eletrônica de um dispositivo (*device*). Um dispositivo pode ser um circuito simples como uma porta lógica ou pode ser um sistema complexo, pode ser um componente de uma biblioteca que define parte de outro dispositivo, ou pode ser composto por diversos outros dispositivos. Um projeto eletrônico é composto por um componente e um *design viewpoint*, conforme esquematizado na Fig. 4.4. Um componente é um objeto que contém uma série de modelos que descrevem o projeto funcionalmente, graficamente,

além de aspectos tecnológicos e aspectos relacionados à temporização. O *design viewpoint* pode ser visto como o controlador da configuração do dispositivo. Detalharemos mais adiante estes dois conceitos.

Um conceito importante utilizado aqui é o de propriedades (*properties*) de um objeto. As propriedades de um objeto diferenciam uma instância de um objeto de outra, e caracterizam cada instância da forma conveniente. A propriedade **net**, por exemplo, serve para identificar uma instância de um objeto tipo barramento ou ligação e sua dimensão; uma propriedade pode servir para especificar o *timing* de um objeto, como o tempo de subida ou descida de um pino; outro exemplo é o uso de uma propriedade para especificar o número de vezes que um determinado bloco é repetido num objeto, facilitando ao projetista que esteja utilizando captura esquemática a utilizar, por exemplo, um único desenho de um *three-state* para interligar dois barramentos, no lugar de desenhar um *three-state* para cada trilha do barramento. O mecanismo de acrescentar informações ao projeto via propriedades é chamado de *anotação de propriedade* (*property annotation*). Algumas propriedades terão seu conteúdo definido em alguma fase posterior do projeto. Chamamos este processo de *forward annotation*. A atribuição ou modificação de valores a propriedades de objetos é chamada de *back annotation*. O projetista pode criar propriedades novas para uso durante o projeto ou pode utilizar as propriedades do sistema. Uma descrição completa de todas as propriedades do sistema pode ser encontrada em [MGC92].

A utilização de hierarquia num projeto feito em metodologia *top-down* é de fundamental importância para não perder-se o controle do projeto. Como já vimos na definição de dispositivos, um dipositivo pode ser composto de outros dipositivos, que por sua vez poderão ser compostos de outros dipositivos, e assim sucessivamente até chegarmos a dispositivos descritos via primitivas de modelamento. O uso de níveis hierárquicos num projeto permite ao projetista especificar blocos de forma que cada bloco não possua uma complexidade muito alta, permitindo sua análise e compreensão mais fácil.

A configuração de um projeto é feita segundo uma série de regras guardadas no *design viewpoint* do projeto. Essas regras podem ser de quatro tipos: (1) parametrizadas, (2) primitivas, (3) para propriedades visíveis e (4) de substituição. A atribuição de valores a variáveis não resolvidas no processo hierárquico é feita pelas regras *parametrizadas*. A definição dos blocos primitivos de um projeto, i.e., dispositivos que não são formados por outros dispositivos, é feita pelas regras de configurar primitivas. A configuração de propriedades para as fases subseqüentes do projeto é feita pelas regras de configuração de propriedades visíveis. Finalmente as regras de substituição permitem a mudança de valores das

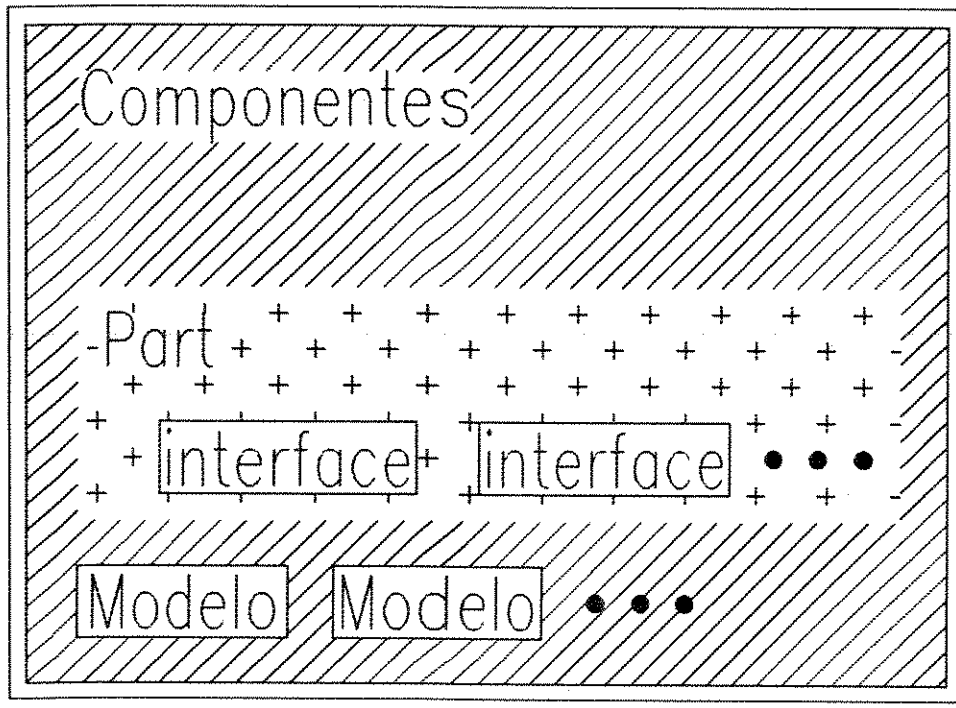


Figura 4.5: Componente

propriedades do projeto.

É possível a manipulação do *design viewpoint* via **da** e via **dve**.

Um componente contêm uma ou mais interfaces e um ou mais modelos, conforme esquematizado na Fig. 4.5. A **part** é o objeto que abriga as interfaces do componente. A interface pode ser vista como a ligação com os diversos modelos que descrevem funcionalmente o componente. A interface é composta por uma lista de pinos, uma lista de propriedades do corpo do componente e uma lista de modelos do componente. Os pinos de um componente permitem a sua ligação com o mundo externo. Para tal é necessário que haja uma consistência entre os pinos da interface e os pinos dos modelos, i.e., que haja uma relação um-p'ra-um entre os dois conjuntos. Pode haver propriedades ligadas ao corpo do componente, i.e., propriedades ligadas ao símbolo do componente. Estas propriedades estão na lista de propriedades do corpo do componente. Finalmente, a tabela de modelos lista todos os modelos do componentes. Como veremos na seção seguinte, há diversas formas de especificar um componente funcionalmente.

Destacamos a descrição VHDL, a descrição via esquemático e via qpt, como veremos mais adiante. Na especificação não funcional, destacamos o modelo que descreve o símbolo gráfico do componente, i.e., a forma com que o componente será representado em esquemáticos.

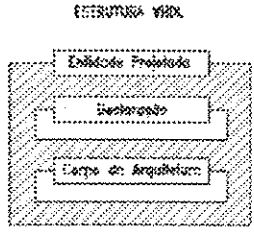
O *C*omponent *I*nterface *B*rowser (cib) ([MGC93c]) é a ferramenta disponível para manipulação da interface de um componente.

4.6 Formas de Modelamentos Funcionais

Há diversas técnicas para modelamento de circuito. Dentre elas destacaremos as *B*ehavioral *L*anguages *M*odel (BLMs), o formato *H*ardware *M*odel *L*ibrary (HML), o qpt, a captura esquemática e a VHDL. C e Pascal são conhecidas linguagens de programação que podem ser usadas como BLMs para descrição de circuitos. Um recurso muito interessante e importante em alguns tipos de projeto é o uso da HML, que permite a interação do modelo do circuito com componentes reais, tornando a análise de projetos complexos que utilizem componentes comerciais complexos, como CPUs, factível. O qpt permite especificação de componentes simples a partir de uma tabela de estado. Os circuitos podem ser descritos também via esquemáticos, utilizando-se o editor de esquemático do da ou via VHDL, que é uma liguagem de descrição de *hardware* padronizada pelo IEEE e analisada em seguida. Detalharemos a seguir estas três últimas formas.

4.6.1 A VHDL

No final da década de 70 e início da década de 80, no projeto *V*ery *H*igh *S*peed *I*ntegrated *C*ircuit (VHSIC) do DOD americano, que visava desenvolver uma nova geração de circuitos integrados, foi proposta a VHDL como liguagem de descrição de circuitos. Buscava-se com isso uma forma de descrever circuitos que suportasse a crescente complexidade dos projetos. Padronizada em dezembro de 1987 pelo IEEE, o VHDL tornou-se também um padrão de fato da industria, que criava assim uma forma de descrição de circuitos intercambiável entre diferentes instituições, i.e., uma forma padronizada. Suportada por todos os fabricantes de ferramentas de EDA, o VHDL permite ao projetista a especificação de circuitos com todas as facilidades encontradas nos tradicionais métodos de captura esquemática. Permite o intercâmbio de descrições de circuito, como é feito hoje via *newsgroup* comp.lang.vhdl do *netnews* da Internet. Propicia a utilização de uma única ferramenta, o próprio VHDL, em diversos níveis da abstração: comportamental, RTL e estrutural. É possível a geração automática de blocos canônicos via, por exemplo da. É possível a síntese automática



```
ENTITY interface IS
    [PORT(port_list);]
END [interface];
ARCHITECTURE modelo OF interface IS
    -- declaracoes
BEGIN
    -- acoes
END [modelo];
```

Figura 4.6: Estrutura VHDL

de circuitos a partir de descrições VHDL, via `autologic`. Faremos nesta seção uma análise da VHDL comparando-a com o método de captura esquemática. Um maior detalhamento da linguagem poderá ser encontrada em [Per91, IEE87, MGC91].

A VHDL oferece ao projetista alguns recursos importantes para a descrição de circuitos eletrônicos. Destacamos as sentenças concorrentes, as sentenças seqüenciais, os sinais, a temporização, a estrutura em forma de entidades, a possibilidade de utilizar diversas arquiteturas, os blocos, os recursos de configuração e as *resolution functions*. Oferece também recursos comuns às liguagens de programação como diversos tipos de variáveis, funções e procedimentos, operadores lógicos, aritméticos e relacionais, pacotes, *overloading* de operadores e manipulação de textos e arquivos.

O projetista dispõe de dois tipos de sentenças: a concorrente e a seqüencial. A descrição concorrente é mais natural numa descrição de circuitos, já que aqui é natural o funcionamento concorrente. Mas dispomos também de sentenças seqüenciais, onde uma sentença é processada por vez.

Os sinais podem ser vistos como variáveis cujo conteúdo é função do tempo.

A temporização possibilita ao projetista atribuir valores a sinais com um certo atraso.

A estrutura de uma descrição VHDL, esquematizada pela 4.6, prevê a existência de uma interface e vários modelos, aqui chamados de arquitetura, por componente. Na declaração da entidade VHDL, são especificadas as `ports`, seus tipos e o sentido do fluxo dos dados em cada `port`. É no corpo da arquitetura que reside a descrição funcional da entidade.

Na captura esquemática, quando buscamos um componente de uma biblioteca e o colocamos no esquemático, estamos criando uma instância deste elemento no nosso projeto. Algumas propriedades são atribuidas a esta instância, como por exemplo as suas ligações no esquemático. A

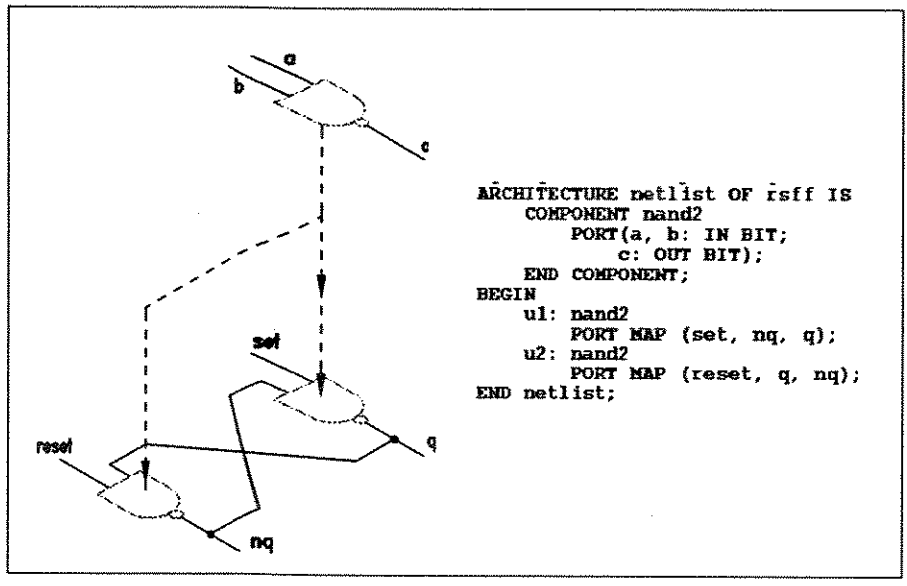


Figura 4.7: Esquemático X Arquitetura

VHDL permite também a instanciação de um componente no esquemático, representado aqui por uma ARCHITECTURE, como ilustra a Fig. 4.7.

Os *blocos* da VHDL permitem ao projetista a organização do projeto de forma a facilitar a sua compreensão, de forma análoga às folhas (*sheets*) na captura esquemática. A estrutura dos blocos está mostrada a seguir:

```
label:  BLOCK [(guarda)]
        -- declaracoes
    BEGIN
        -- sentencas concorrentes
    END BLOCK label;
```

Outro recurso importante para o projetista é a *resolution function*, onde o projetista pode definir o que deve acontecer quando dois valores são simultaneamente atribuidos a um sinal. A Fig. 4.8, exemplifica o recurso.

4.6.2 Captura Esquemática

A captura esquemática permite ao projetista criar componentes a partir de outros componentes. Num esquemático podemos usar componentes de alguma biblioteca do sistema, componentes de uma biblioteca do

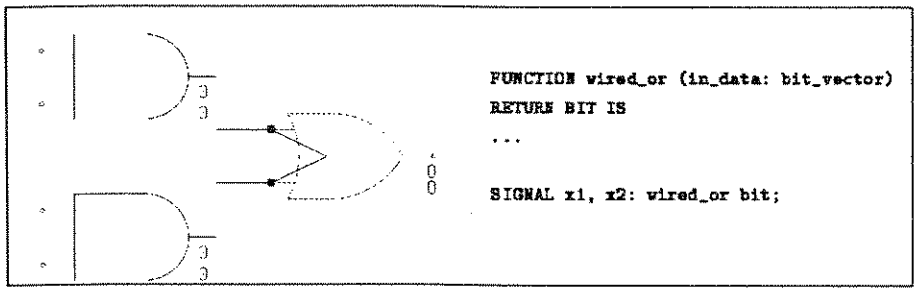


Figura 4.8: *Resolution Function*

próprio projetista, ou de sua equipe, podemos utilizar também outros componentes especificados via captura esquemática ou alguma outra forma de modelamento, como VHDL. Esta possibilidade permite a hierarquização do projeto, indispensável quando este possui um alto grau de complexidade.

A captura esquemática no ambiente utilizado por nós é feita pelo `da`, assim como a geração de símbolos, a geração automática de blocos canônicos em VHDL, a descrição VHDL, a criação da interface e a checagem do projeto. O fato de haver apenas uma ferramenta para a fase de criação do projeto agiliza este processo.

A existência de macros a nível de captura esquemática pode agilizar o trabalho do projetista, tornar o projeto reutilizável e facilitar a sua interpretação. É possível fazer construções como `FOR i=1 TO N` (Fig 4.9), (sendo `N` uma propriedade do componente) ou outras construções como `IF` e `CASE`. Assim podemos repetir uma célula num esquemático, o número de vezes que for necessário, apenas atribuindo o valor correto à propriedade `N` desta célula.

Algumas tarefas do projeto são automaticamente feitas pelo `da`. Pode-se gerar automaticamente símbolos a partir de esquemáticos. Em geral utilizamos esta facilidade para gerar um símbolo inicial, que depois é modificado pelo projetista conforme seu desejo. A geração de símbolos a partir de descrições VHDL tem a mesma função da geração a partir de esquemáticos (geração de *entities* VHDL a partir de símbolos). A geração automática de especificações VHDL de componentes canônicos é possibilitada através da definição do componente desejado e de suas características via *menus*. Assim o projetista pode facilmente gerar, por exemplo, a descrição em VHDL de um banco de 8 *flip-flops* tipo `D` sensível a borda, com *set* e *reset* (Fig 4.10).

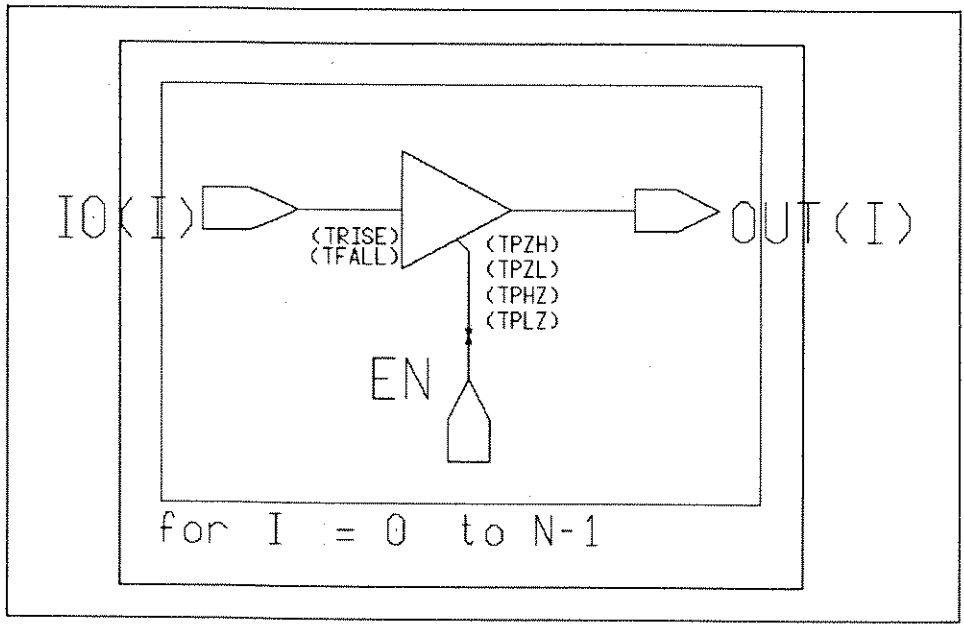


Figura 4.9: Macros em Esquemáticos

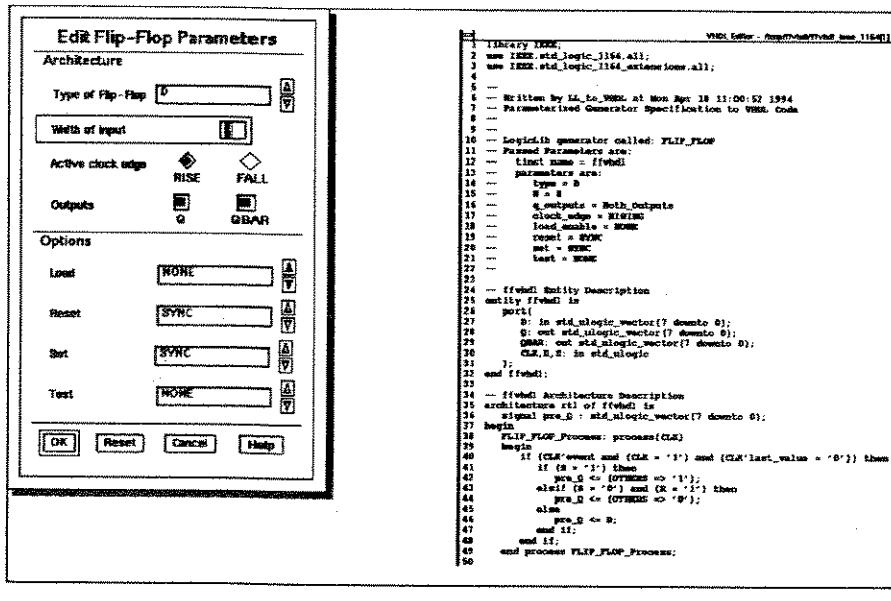


Figura 4.10: FF Tipo D Gerado pelo da

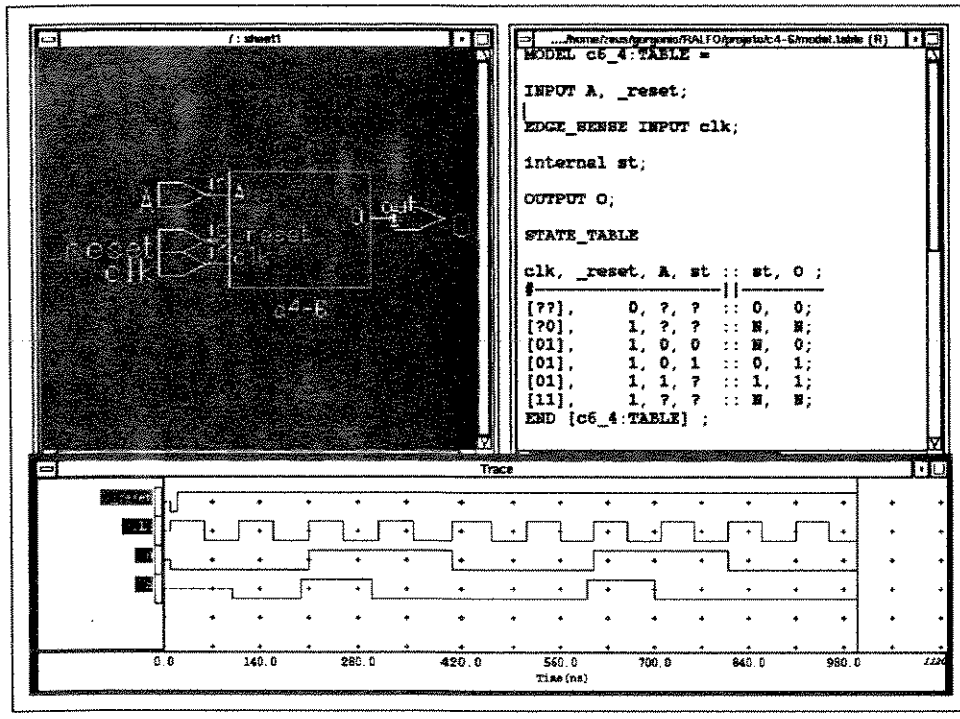


Figura 4.11: c4-6

4.6.3 qpt

A descrição via `qpt` permite a especificação de um componente a partir de uma tabela de estados. Esta facilidade permite a geração rápida de componentes deste tipo necessário num projeto, mas cuja descrição lógica não é importante numa primeira fase. A limitação deste processo é o número de estados que não pode ser superior a 10.

Para geração, por exemplo, do componente `c4-6` criamos a tabela de estado mostrada na janela `model.table` da Fig. 4.11. Então, com ajuda da geração automática de símbolo do `da`, obtivemos o símbolo mostrado na janela `sheet1` e na janela `Trace` temos o resultado da sua simulação.

4.7 A Metodologia Adotada

No nosso projeto trabalhamos até a fase de verificação e análise. O resultado da fase de concepção, iniciada por Guardieiro ([Gua91]) foi

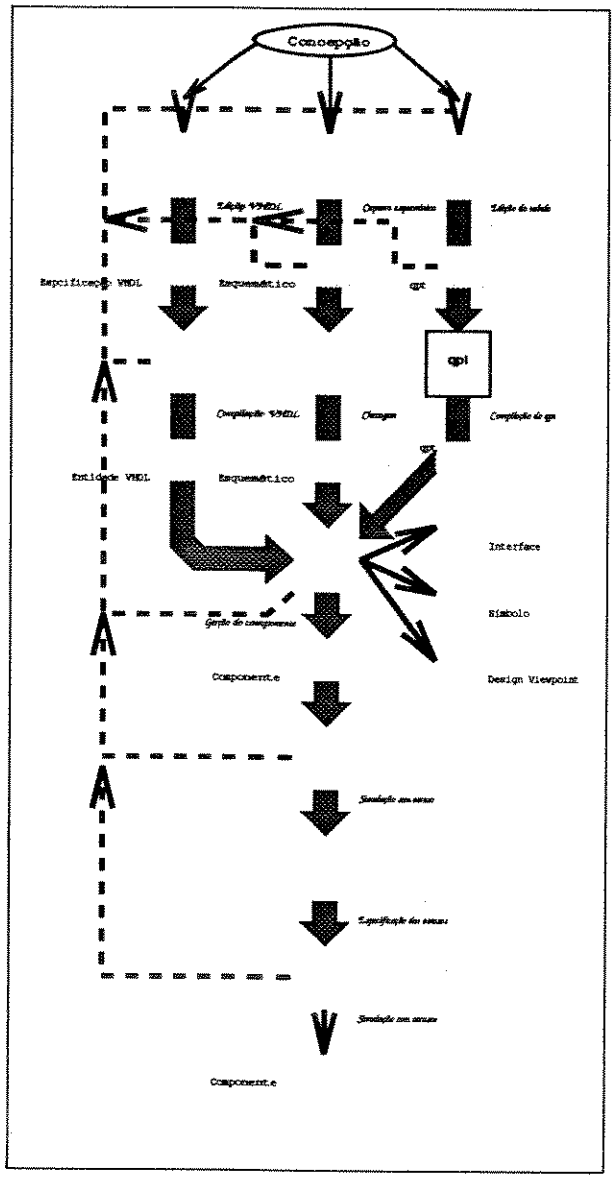


Figura 4.12: Ciclo do Projeto

detalhada no capítulo 3. A fase de criação foi feita, em parte, em conjunto com o estagiário André Goldemberg fazendo uso do suporte a projeto concorrente do ambiente de EDA da MGC. Finalmente utilizamos do simulador lógico `quicksim` na fase de análise do projeto ([MGC93k]). Detalharemos a seguir as duas últimas fases (Fig 4.12).

Na fase de criação do projeto adotamos duas formas de modelamento funcional: a captura esquemática e a linguagem VHDL. Seguindo a metodologia *top-down* e buscando fazer a descrição comportamental do projeto, fizemos a descrição de cada módulo diretamente em VHDL e ou em esquemáticos compostos por componentes VHDL.

A especificação em VHDL foi feita no próprio `da`, utilizando-se o editor VHDL, voltado para a linguagem. Terminada a edição, dentro do próprio `da`, o `sys_1076` é invocado criando, no caso da especificação estar sintaticamente correta, uma *interface* e uma entidade VHDL.

A especificação em esquemático foi feita também no `da`, utilizando-se de símbolos de alguma biblioteca do sistema ou do projetista, ou mesmo de símbolos *stanks* previamente definidos.

A especificação do `qpt` também pode ser feita no editor ASCII do `da`. Sua compilação é processada através do utilitário `qpt` da linha de comando. Os componentes feitos com o `qpt` não foram utilizados, em função da mudança da metodologia de projeto.

Após ser feita a checagem/compilação torna-se necessária a criação do símbolo, que é considerado um modelo meramente gráfico do componente. O símbolo poderá ser gerado automaticamente a partir de qualquer uma das três especificações. O `da` possui também um editor gráfico para geração manual ou aperfeiçoamento do símbolo gerado automaticamente. No caso de não ser uma especificação VHDL é preciso ainda gerar a interface do componente, passo efetuado automaticamente pelo `da`. Temos então o componente completamente especificado. Reforçamos que, conforme esquematizado na Fig. 4.5, um componente é formado pelos seus (1) modelos: a entidade VHDL e/ou esquemático e/ou `qptfile` e o símbolo; e (2) pela(s) sua(s) interface(s). Completando o projeto é preciso a geração do *Design Viewpoint*. O `da` gera automaticamente um *Design Viewpoint default*.

Duas outras ferramentas ainda não citadas foram bastante usadas neste projeto: o *Bold Browser* (`boldbrowser`) e o *Design Manager* (`dmgr`). A primeira é uma ferramenta para consulta e navegação da/na documentação do ambiente EDA da MGC, podendo ser invocada a partir de botões de `help/ref` presentes em quase todas as ferramentas. A segunda funciona como navegador sobre as ferramentas da MGC e sobre o seu projeto.

É importante enfatizar o caráter hierárquico do projeto. Podemos ver todo ele como um único componente (Fig. 5.1) composto por diversos

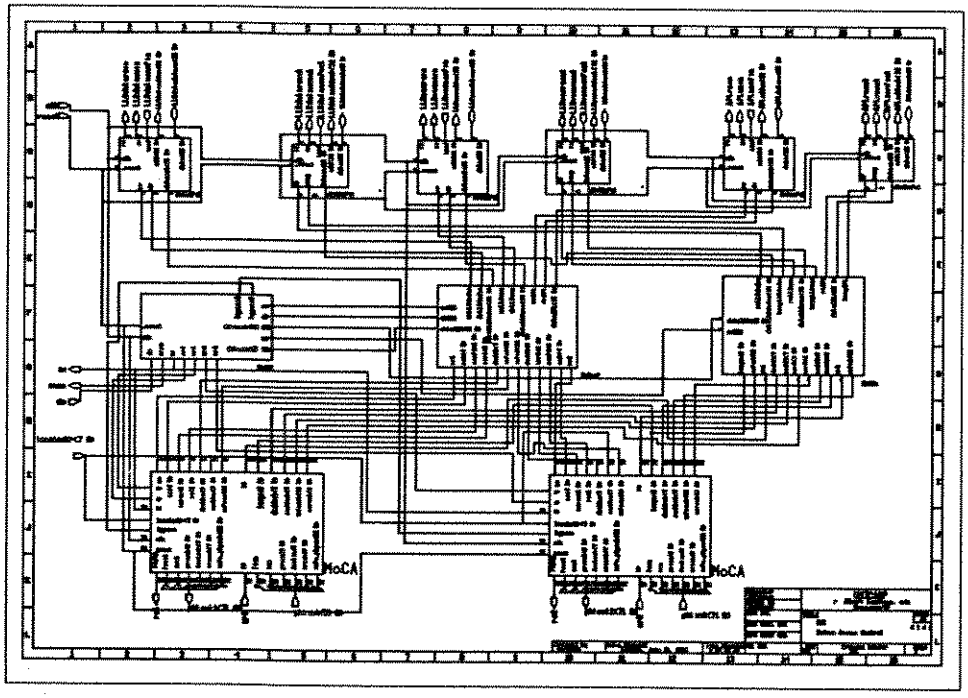


Figura 4.13: Esquemático da MAC

outros componentes (Fig. 4.13), que, por sua vez, são compostos por outros componentes e assim sucessivamente até chegar-se a componentes primitivos, as chamadas células básicas.

As células básicas utilizadas nas descrições via captura esquemática foram células VHDL.

Capítulo 5

A Implementação

Neste capítulo detalhamos o projeto da MAC da RALFO, mostrando o resultado da simulação em cada um dos módulos especificados. Boa parte do nosso tempo gasto nesse projeto foi dispendido buscando especificá-lo a nível de esquemático, numa metodologia *bottom-up*, mas com o aparecimento de ferramentas como o `autologic`, que possibilitam a conversão de descrições de mais alto nível, como pode ser uma especificação em VHDL, em uma especificação a nível de *gates*, i.e., células básicas, optamos por redirigir o trabalho buscando uma especificação totalmente feita em VHDL.

O *timing* do projeto está amarrado à taxa de transmissão da rede, que, para efeito das simulações efetuadas, será considerada de 10 `Mbits/s`.

Estamos descrevendo componentes que possuem duas descrições: (1) a estrutural e (2) a funcional. A descrição estrutural apenas especifica as interfaces do componente, detalhando o tipo de interface: se é um único pino ou um barramento, o que trafega neste barramento (no nosso caso sempre sinais lógicos: Z,%0,%1, e X), se é de entrada, saída ou bi-direcional. A descrição funcional contempla o comportamento do componente.

Todos os módulos foram especificado em VHDL, exceto o MoIB, que está mais relacionado com o barramento do equipamento do que com a MAC propriamente dita e o MoTI, cuja não existência não compromente o funcionamento da MAC em estado normal. Assim deixamos esta especificação para trabalhos futuros. A MAC sem o MoTI perde a capacidade de retirar pacotes perdidos nos anéis, assim como a capacidade de detectar falhas na rede e reconfigurá-la automaticamente.

É importante notar que toda a MAC, assim como cada um dos módulos, pode ser vista como um componente. Visto assim podemos encarar a descrição da MAC como uma descrição a nível de esquemático.

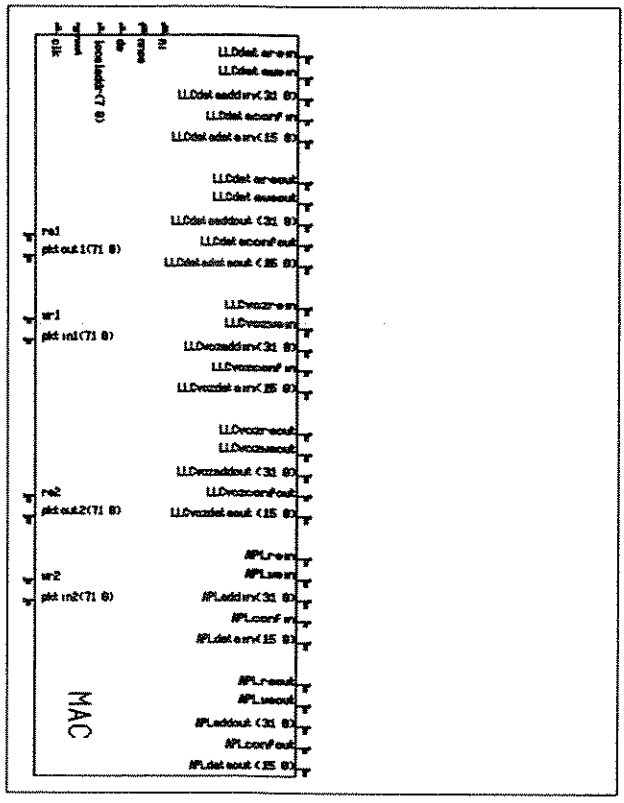


Figura 5.1: Símbolo da MAC

5.1 MAC

A MAC, cujo símbolo está na Fig. 5.1, possui interface com seis MolBs, um para cada usuário e com a camada PHY, além dos pinos *Reset* (`reset`), que evidentemente reinicia a MAC e o `localaddr` que indica qual o endereço da MAC.

A interface com o usuário pode ser de entrada, i.e., com os dados vindos do usuário para a MAC, ou de saída, i.e., com os dados indo para o usuário. A MAC estará sempre verificando se há algum envelope em alguma fila de entrada e, sempre que há algum envelope a ser encaminhado a uma fila de saída, ela tenta colocá-lo na fila. Para receber um envelope a MAC:

1. verifica se a fila de entrada não está vazia;

2. envia o envelope em quatro ciclos de relógio; e

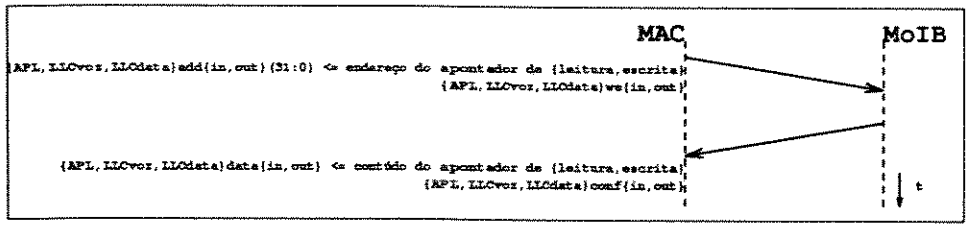


Figura 5.2: Requisição de Endereço

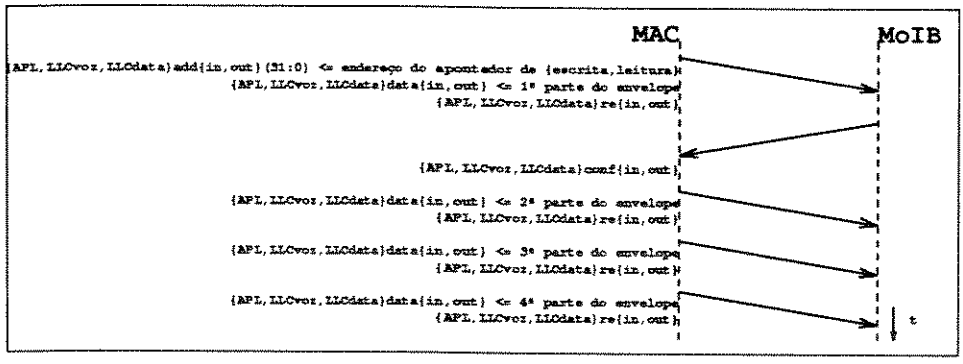


Figura 5.3: Envio de Mensagem

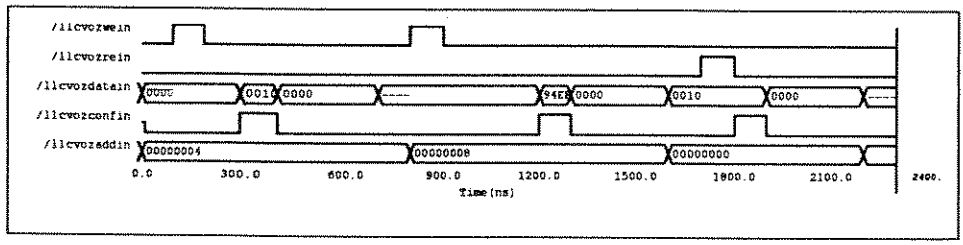


Figura 5.4: Diagrama de Tempo da Interface com um MoIB de Entrada

3. atualiza o endereço de leitura na fila de entrada.

Para verificar se a fila não está vazia é preciso solicitar ao usuário o conteúdo do apontador de escrita. Como esquematizado na Fig. 5.2, a MAC coloca o endereço do apontador de escrita, 0x0004, no {APL,LLC/voz,LLC/dados}`add{In,Out}`$_{out}$ e ativa o {APL,LLC/voz,LLC/dados}`we{In,Out}`$_{out}$, aguardando então a chegada do {APL,LLC/voz,LLC/dados}`conf{In,Out}`$_{in}$ para finalmente ler o endereço de escrita no {APL,LLC/voz,LLC/dados}`data{In,Out}`$_{in/out}$, comparando-o então com o endereço de leitura, guardado internamente. Se a fila não estiver vazia a MAC busca os dados, como esquematizado na Fig. 5.3. O endereço de leitura é colocado no {APL,LLC/voz,LLC/dados}`add{In,Out}`$_{out}$, quando então a leitura é solicitada ao MoIB ativando-se o {APL,LLC/voz,LLC/dados}`we{In,Out}`$_{out}$ (escrita do MoIB na MAC). O MoIB envia à MAC a confirmação de leitura concomitatemente com a palavra menos significativa da mensagem. Em seguida a MAC envia ao MoIB o seu atual endereço de leitura.

O diagrama de tempo desta interface pode ser visto na Fig. 5.4. O primeiro passo consiste na colocação do endereço do apontador de escrita, 0x0004, no LLC/voz *Add*ress *In*put (LLC/`vozaddin`$_{out}$) indicando o fato ao MoIB ativando o LLC/voz *W*rite *E*nable *In*put (LLC/`vozwein`$_{out}$). Após o MoIB ter conseguido ler o conteúdo do endereço relativo 0x0004 ele o encaminha para a MAC, através do LLC/voz *Data In*put (LLC/`vozdatain`$_{in/out}$), indicando o fato ativando o LLC/voz *Conf*irmation *In*put (LLC/`vozconfin`$_{in}$) e encaminhando os bytes mais significativos nos próximos 3 ciclos do relógio. A troca de dados entre o MAC e o MoIB dar-se-á sempre através de quatro ciclos do relógio, cada um transferindo uma palavra de 16 *bits*, a MAC ativa por um ciclo o `we`$_{out}$ quando quiser ler alguma coisa da fila e o `re`$_{out}$ quando quiser escrever alguma coisa na fila, aguardando a confirmação através do `conf`$_{in}$, para então enviar as outras três palavras. Comparando o conteúdo do apontador de escrita recebido com o conteúdo do apontador de leitura sob seu controle, a MAC solicitará

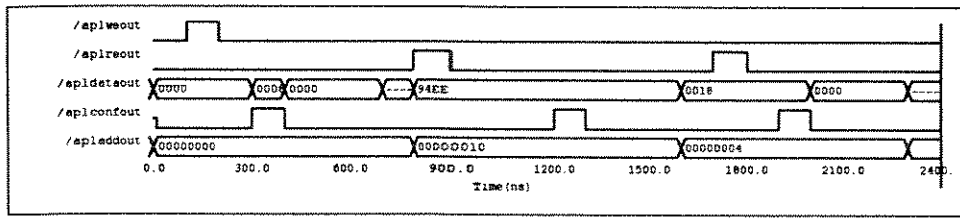


Figura 5.5: Diagrama de Tempo da Interface com um MoIB de Saída

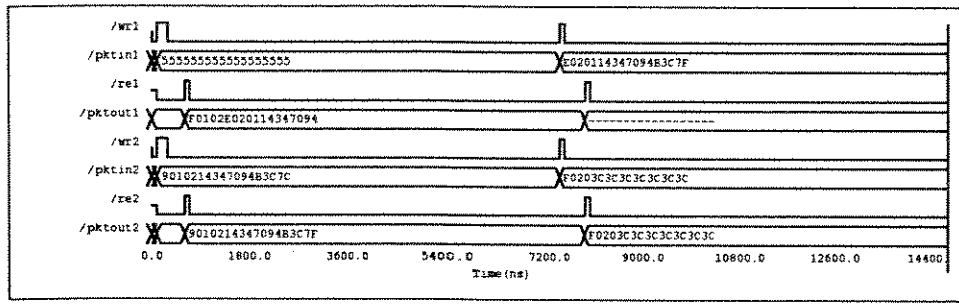


Figura 5.6: Diagrama de Tempo da Interface com a MAC/PHY

os dados da fila se esta não estiver vazia. No exemplo da Fig. 5.4 a fila não estava cheia (o endereço de escrita era 0x0010 e o de escrita 0x0008) então a MAC solicitou a mensagem colocando o endereço de leitura (0x0008) no LLC/$\texttt{vozaddin}_{out}$ e ativou o LLC/$\texttt{vozwein}_{out}$, mantendo o endereço disponível até receber a confirmação de recebimento dos dados do MoIB através do LLC/$\texttt{vozconfin}_{in}$, pegando-os finalmente no LLC/$\texttt{vozdatain}_{in/out}$. Fechando o ciclo a MAC passa para a fila o novo valor do apontador de leitura.

A interface da MAC com um MoIB de saída é análoga a com o MoIB de entrada. A Fig. 5.5 exemplifica esta interface.

A interface de entrada da MAC com a PHY é composta apenas por um barramento, o $\texttt{pktin}\{1,2\}_{in}$, e um sinal para solicitar a escrita na MAC, o $\texttt{wr}\{1,2\}_{in}$. A PHY coloca o pacote no $\texttt{pktin}\{1,2\}_{in}$ e indica o fato ativando o $\texttt{wr}\{1,2\}_{in}$, assumindo que os dados serão lidos pela MAC em um ciclo do relógio. A Fig. 5.6 mostra um exemplo do diagrama de tempo, detalhado na Fig. 5.7. Aqui pegamos quatro casos. Os dois primeiros no anel 1 e os dois últimos no anel 2. No primeiro caso a MAC local, cujo endereço é 0x1 recebe um *slot* vazio e encaminha um pacote tipo APL para a rede. No segundo caso um pacote LLC retorna à MAC e o *slot* é liberado como vazio. No terceiro caso a MAC recebe um pacote LLC/voz e indica o fato

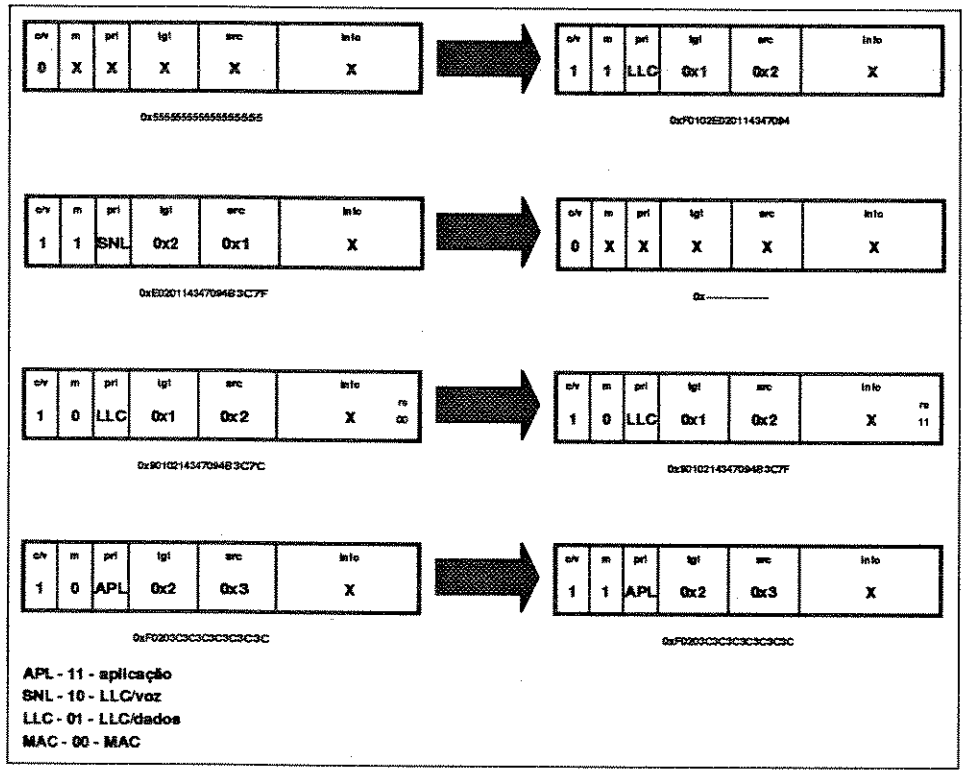


Figura 5.7: Esquema do Diagrama de Tempo da Interface com a MAC/PHY

no campo `re`, encaminhando o pacote à rede. No último caso o pacote não é local nem é para a MAC local, sendo passado adiante.

As descrições formais em VHDL das interfaces da MAC estão na seção A.1. Nesta especificação utilizamos o fato do VHDL permitir a exitência de vários modelos (`architectures`) para uma única entidade (`entity`).

5.2 MoCA

Analisaremos nesta seção o comportamento do MoCA (Fig. 3.18), simulando sete casos: (1) o envio de um pacote tipo 1, (2) o envio de um pacote tipo 2, (3) o recebimento de um pacote tipo 2, (4) o repasse de um pacote de e para outros, (5) a recusa de um pacote tipo 1, (6) o retorno de um pacote tipo 1, e (7) o retorno de um pacote tipo 2. Os 7 casos são simulados seqüencialmente, corespondendo a 7 pacotes consecutivos correndo no mesmo anel. Por simplicidade imaginamos que nossa rede é composta por apenas três nós, e analisamos um único anel.

Aqui o comportamento do MoCA é sincronizado pela PHY. A chegada de um *slot* é indicada ao MoCA ativando-se o $\texttt{wr}_{in}$ após a disponibilização do pacote nos pinos correspondentes, como especificado na seção 3.6.3. Se o *slot* estiver vazio, i.e., o $\texttt{fein}_{in} = \%0$ o MoCA solicitará um novo envelope ao MoTraP, encaminhando-o à rede neste *slot*. Não havendo nehum envelope novo a ser encaminhado, o *slot* será repassado como vazio. Se o *slot* recebido estiver cheio e o envelope nele contido for do tipo 1 e proveniente do nó local, será solicitado ao MoTraP o próximo envelope deste terminal, i.e., o próxmo envelope com o mesmo `tid`. Não havendo próximo envelop o *slot* será repassado como vazio. Se o envelope recebido for do tipo 2 e proveniente do nó local, o *slot* também será encaminhado como vazio. No caso do envelope ser destinado ao nó local, diretamente ou em *broadcast*, ele será repassado para o MoTraP, se o usuário em questão, LLC, LLC/voz ou APL, não estiver ocupado. O MoTraP indicará, no caso de um envelope do tipo 2, se o usuário aceitou ou não o envelope, através do campo `re`, repassando o envelope para a rede. Finalmente os envelopes que não forem destinados ao nó local e que não forem provenientes do nó local serão simplesmente repassados para o próximo nó. Estes procedimentos epecificam funcionalmente o MoCA. Sua descrição formal encontra-se na seção A.2.

Nas saídas que serão mostradas a seguir todos os sinais binários, i.e., aqueles que não são vetores, são representados binariamente. Os sinais que são vetores, i.e., barramentos, são representados em hexadecimal, exceto os $\texttt{priin}_{in}$, $\texttt{priout}_{out}$, $\texttt{si}_{out}$, $\texttt{ri}_{in}$, $\texttt{serin}_{in}$, $\texttt{busys}_{in}$, $\texttt{serout}_{out}$ que, embora vetoriais, são representados binariamente.

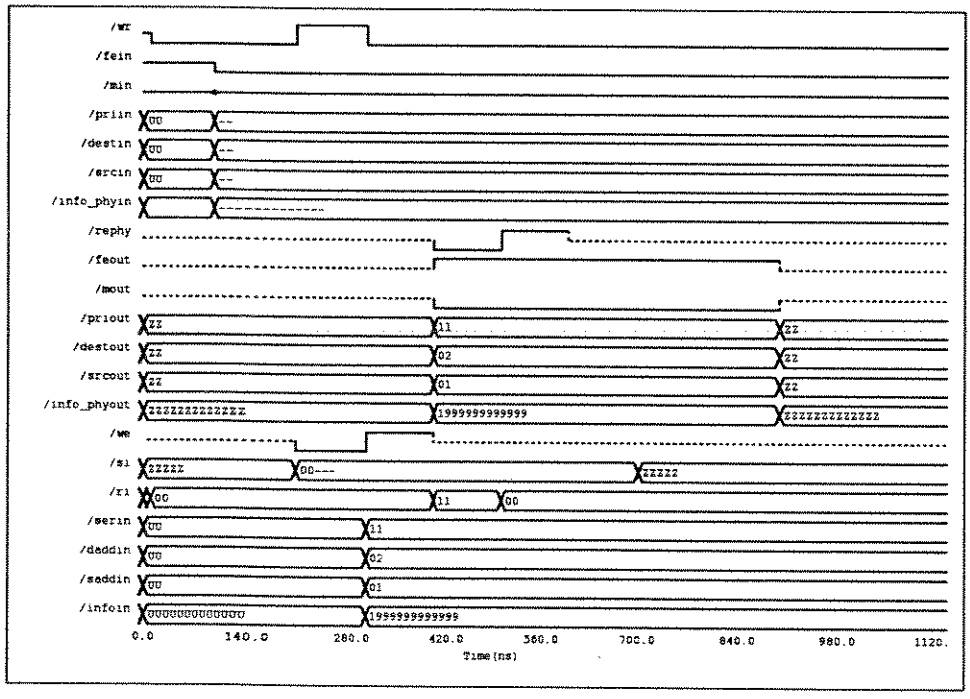


Figura 5.8: MoCA: Caso 1

Simplificamos a descrição do MoCA não especificando a função de cálculo e verificação do chk. Em nenhum caso o MoCA estará trabalhando no modo *by pass*.

No primeiro caso o MoCA encaminha num *slot* vazio um envelope da APL, i.e., um envelope tipo 1. O diagrama de tempo deste caso está na Fig. 5.8. Aqui a PHY indica ao MoCA a chegada de um *slot* vazio, fazendo $\texttt{fein}_{in}$ = %0 e, em seguinda, ativando o $\texttt{wr}_{in}$. Note que os valores dos pinos $\texttt{m}_{in}$, $\texttt{priin}_{in}$, $\texttt{destin}_{in}$, $\texttt{srcin}_{in}$ e $\texttt{info\_phyin}_{in}$ são indeterminados, pois são irrelevantes. O MoCA então indica ao MoTraP a chegada de um *slot* vazio, tornando $\texttt{si}_{out}$ = %00XXX e ativando $\texttt{we}_{out}$. O MoTraP responde positivamente ao MoCA, fazendo $\texttt{ri}_{in}$ = %11, após colocar o envelope nos pinos $\texttt{serin}_{in}$, $\texttt{dadd}_{in}$, $\texttt{sadd}_{in}$ e $\texttt{info}_{in}$. Neste caso trata-se de um envelope da APL, do terminal 1 do nó local (0x01) para o nó 0x02 com a hipotética mensagem 0x999999999999. Finalmente o MoCA encaminha a mensagem para a PHY pelos sinais $\texttt{feout}_{out}$, $\texttt{m}_{out}$, $\texttt{priout}_{out}$, $\texttt{destout}_{out}$, $\texttt{srcout}_{out}$ e $\texttt{info\_phyout}_{out}$, e ativa o sinal $\texttt{rePHY}_{out}$.

O segundo caso, cujo diagrama de tempo está na Fig. 5.9, é análogo ao

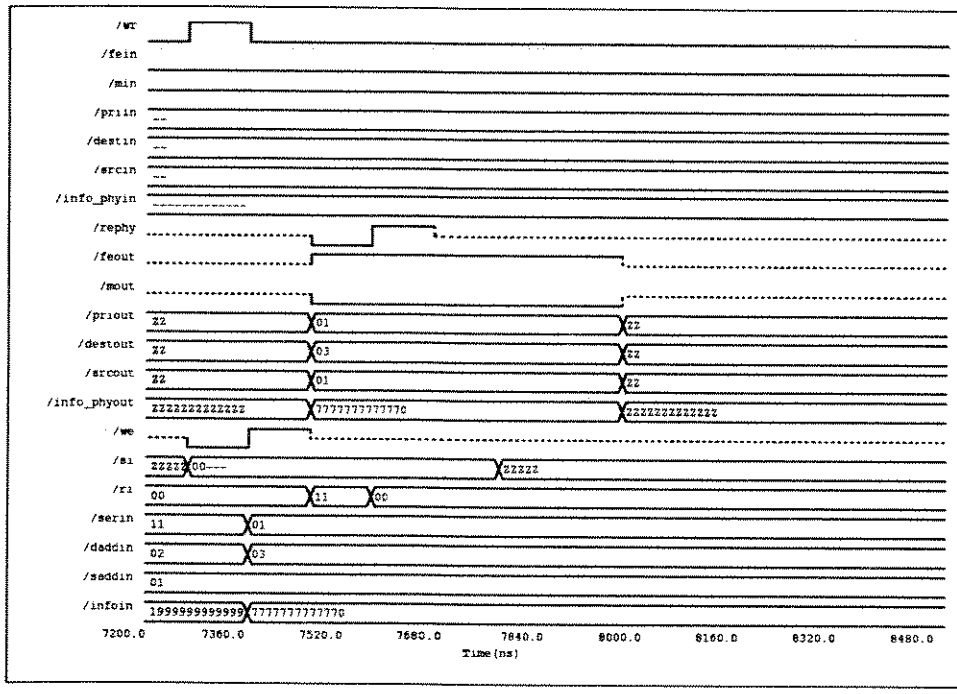


Figura 5.9: MoCA: Caso 2

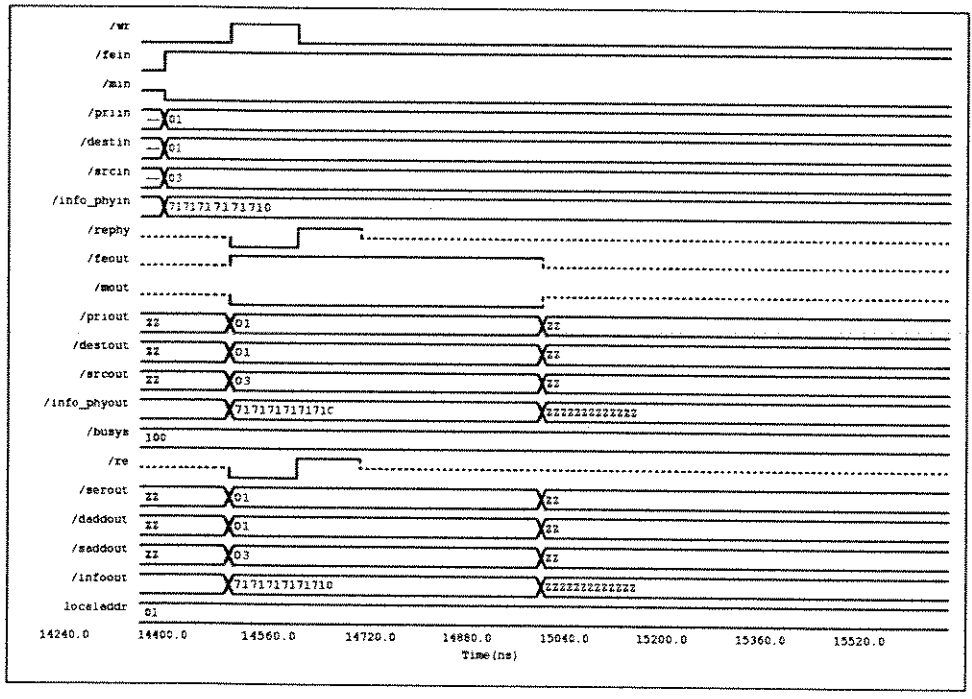


Figura 5.10: MoCA: Caso 3

primeiro, porém aqui trata-se da mensagem 0x777777777777 da LLC/dados destinada ao nó 0x03. Note que o campo **re** é iniciado com %00 e que o campo **chk** não está sendo calculado pela MoCA.

O MoCA, no terceiro caso (Fig. 5.10), recebe um pacote destinado à LLC/dados do seu nó, percebendo que o bit menos significativo do $\mathbf{busys}_{in}$ está %0, ele encaminha o envelope para o MoDEn, através dos pinos $\mathbf{serout}_{out}$, $\mathbf{dadd}_{out}$, $\mathbf{sadd}_{out}$ e $\mathbf{info}_{out}$, e indica o fato ativando o $\mathbf{re}_{out}$. O MoCA então encaminha o envelope para a rede, tomando o cuidado de indicar no campo **re** (sinal $\mathbf{info\_phyout}_{out}$) que o envelope foi recebido, i.e., fazendo **re** = %11.

No quarto caso, cujo diagrama de tempo pode ser visto na Fig. 5.11, o MoCA recebe um pacote da LLC/voz do nó 0x02 destinado ao nó 0x03. Aqui o trabalho é só encaminhar da entrada da PHY para a saída.

Na Fig. 5.12 temos o diagrama de tempo do caso 5, onde um pacote da LLC/voz proveniente do nó 0x02 é recusado, pois a fila de saída para a LLC/voz está cheia conforme a indicação do $\mathbf{busys}_{in}(1)$. Aqui, ao repassar o pacote para a PHY, o MoCA indicará no campo **re** (que equivale ao

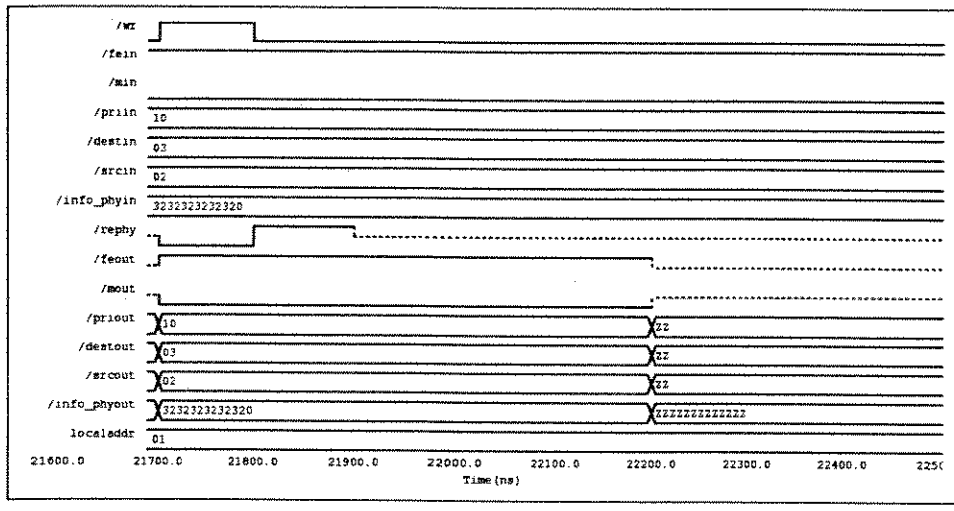


Figura 5.11: MoCA: Caso 4

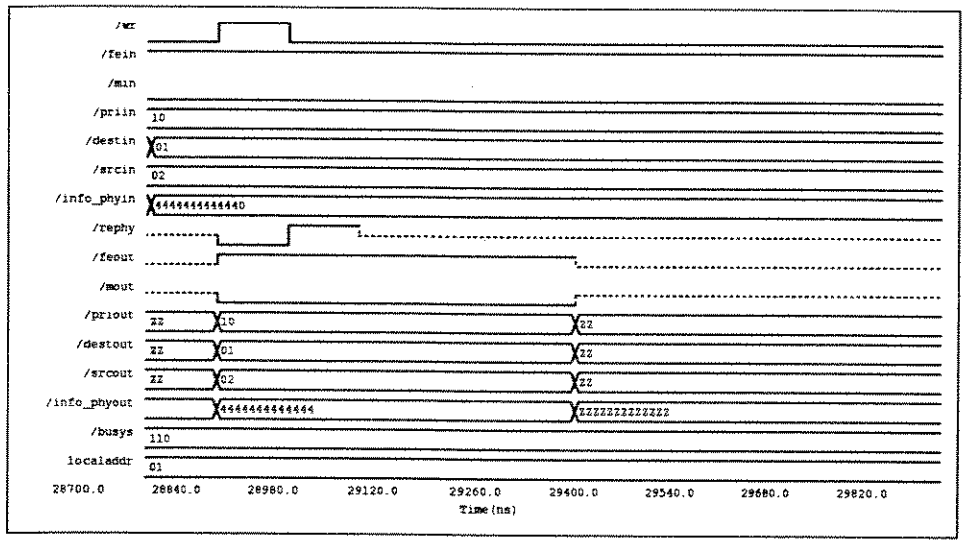


Figura 5.12: MoCA: Caso 5

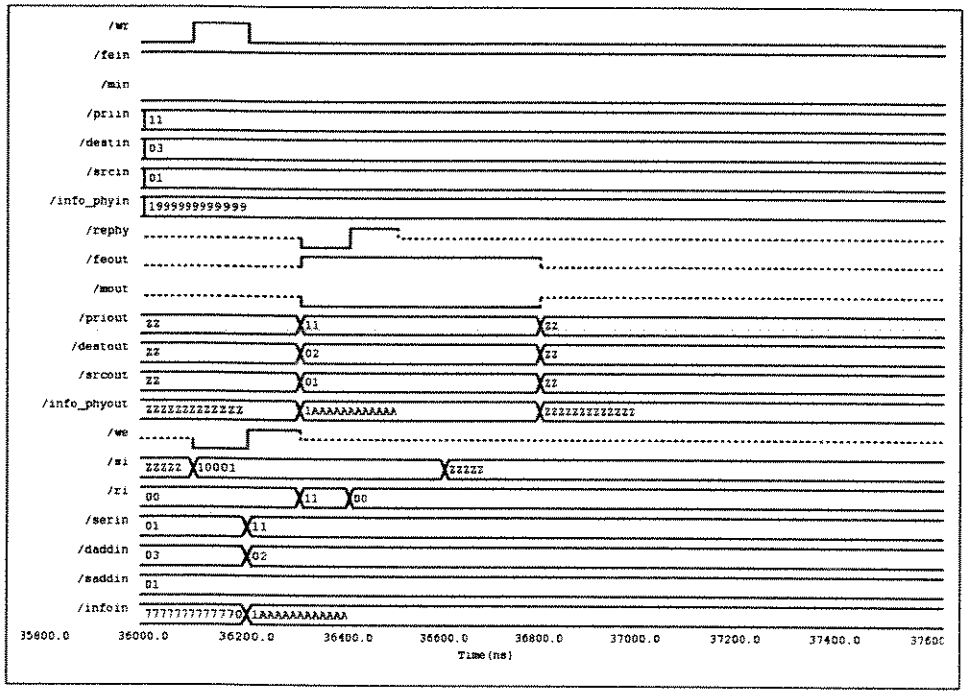


Figura 5.13: MoCA: Caso 6

$\texttt{info\_phyout}_{out}\texttt{(3 downto 2)}$) que o pacote foi recusado, fazendo `re` = %01.

Supomos que o pacote enviado no caso 1 retornou no caso 6, e que a APL do nó corrente, possui outra mensagem para enviar proveniente do mesmo terminal 1 (Fig. 5.13). Aqui o MoCA indicará ao MoTraP o retorno do *slot* com o `tid` 1, colocando o valor do $\texttt{info\_phyin}_{in}\texttt{(51 downto 48)}$ no $\texttt{si}_{out}\texttt{(3 downto 0)}$, quando então o MoTraP encaminhará a próxima mensagem (0xAAAAAAAAAAAA) para o MoCA, que a repassará para a PHY.

No sétimo e último caso, analogamente ao caso anterior, supomos que a mensagem do caso 2 tenha retornado, então o MoCA indicará o fato ao MoTraP, e repassará o *slot* para a PHY como vazio. No diagrama de tempo que pode ser visto na Fig. 5.14, notamos a chegada do pacote retornado do nó 0x03, notamos que ele foi recebido pela LLC/dados, pois o campo `re` da mensagem ($\texttt{info\_phyin}_{in}\texttt{(3 downto 2)}$) é %11. O MoCA então repassa a informação ao MoTraP. É importante notar aqui que, caso a fila de saída do usuário em questão esteja cheia, a confirmação será descartada.

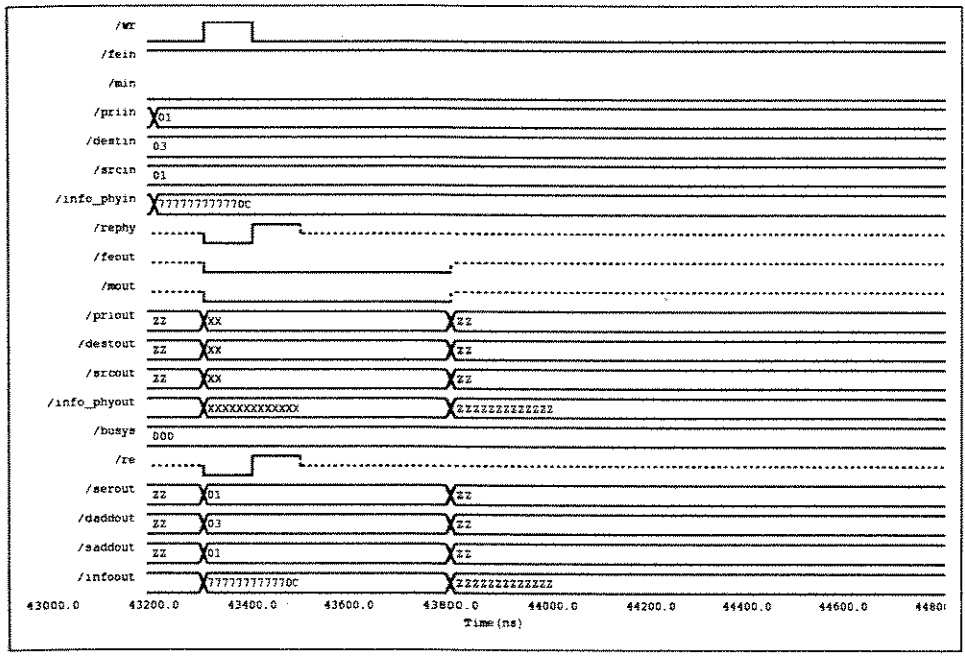


Figura 5.14: MoCA: Caso 7

5.3 MoTraP

Nesta seção detalharemos o funcionamento do MoTraP, exemplificando-o com a simulação de alguns casos. O papel do MoTraP é encaminhar ao MoCA novas mensagens da APL, LLC/voz, LLC/dados e MAC e encaminhar fragmentos de mensagens da APL sempre que for solicitado pelo MoCA. Desta forma o MoTraP pode receber dois tipos de envelopes da APL: envelopes de uma nova mensagem ou envelopes de uma mensagem em curso na rede. Cabe ao MoTraP, ao receber um envelope da APL, guardá-lo numa memória interna reservada ao terminal telefônico correspondente ao envelope recebido (identificado pelo `tid`). É também sua responsabilidade saber quais são os terminais com mensagens novas, quais são os terminais com mensagens em andamento, e, destes últimos, quais cujo envelope presente em sua memória já foram enviados à rede. Quando não há novas mensagens da APL para serem enviadas, o MoTraP verificará se há alguma de algum dos outros usuários. Já que as mensagens dos usuários LLC/voz, LLC/dados e MAC ficam aguardando nas filas de entrada ou, no caso da MAC, no MoTI, a interface entre o MoTraP e esses usuário é muito mais simplificada do que a interface com a APL. Nesse caso a fila tem que estar sendo atendida constantemente, independentemente da chegada ou não de *slot*s vazios no nó local. Quando o MoTraP recebe um novo fragmento de mensagem da APL e ainda não enviou o fragmento anterior à rede, o antigo segmento será descartado, sendo o novo fragmento guardado no lugar do antigo.

É preciso então que o MoTraP guarde: (1) os fragmentos de mensagens recebidos da APL, (2) os estados de cada terminal, e (3) a fila de novas mensagens da APL.

O MoTraP, cujo esquemático pode ser visto na Fig 5.15, é composto por 5 componentes: 2 `motrap_low_cores`, 1 `motrap_high_core`, 1 `motrap_data_base` e 1 `motrap_scheduler`.

O `motrap_low_core` (especificado formalmente em A.3.1) é o responsável pela interface com o MoCA, recebendo portanto os pedidos de pacotes novos, quando da chegada de um *slot* vazio, ou os pedidos de pacotes contendo um fragmento de mensagem da APL, quando da chegada de um *slot* reservado à APL. O `motrap_high_core` indica ao `motrap_low_core` quando há alguma mensagem nova da APL, i.e., quando existe algum envelope da APL que não possui um *slot* reservado na rede. Desta forma, quando há o pedido de uma nova mensagem pelo MoCA e há novas mensagens da APL aguardando *slot*s o `motrap_low_core` solicita que o `motrap_high_core` encaminhe o envelope contendo a nova mensagem para o MoCA.

O `motrap_high_core` (especificado formalmente em A.3.2) está sempre atendendo a chegada de novos fragmentos da APL, indicando ao

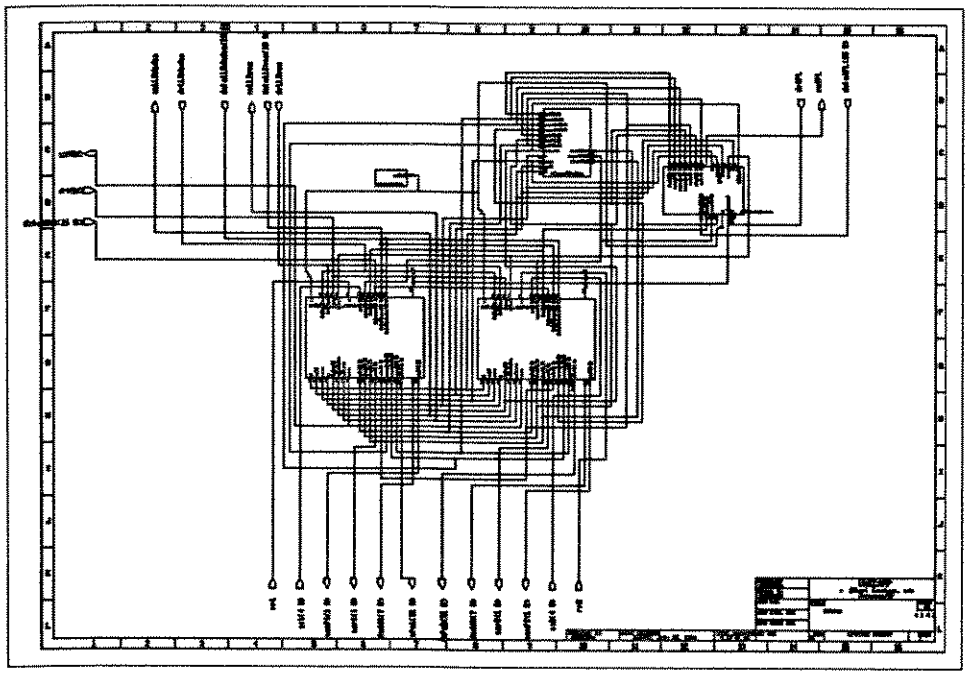


Figura 5.15: Esquemático do MoTraP

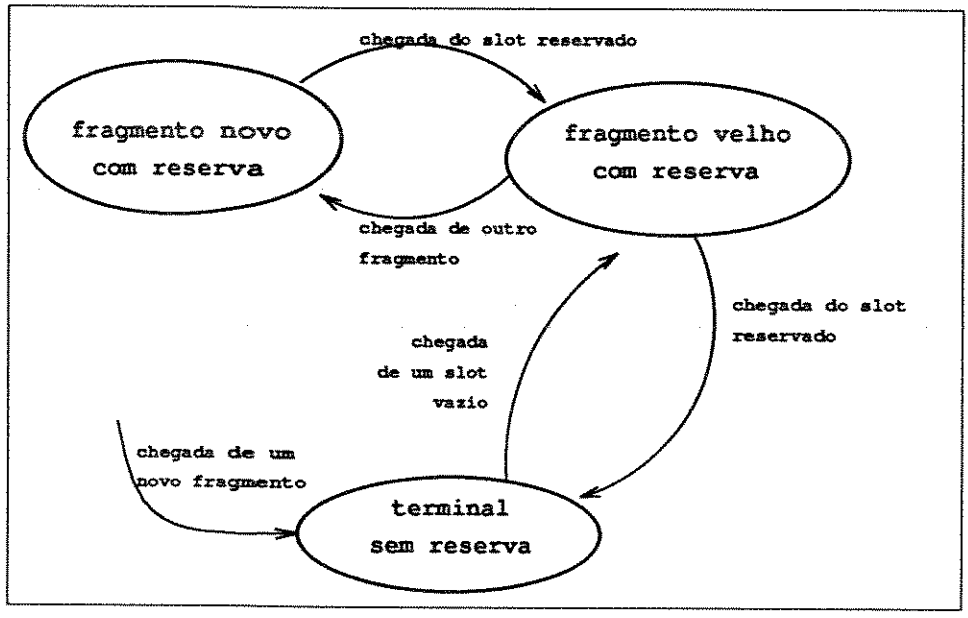


Figura 5.16: Estados dos Terminais

`motrap_low_core` a existência de mensagens novas da APL, gerenciando o estado de cada terminal da APL e encaminhando novas mensagens da APL quando solicitado pelo `motrap_low_core`. O `motrap_high_core` gerencia uma fila de mensagens novas da APL, enviando ao `motrap_low_core`, sempre que solicitado, a mensagem mais antiga da fila. A Fig. 5.16 ilustra o diagrama de estado utilizado por este componente para gerenciar os estados de cada terminal. Assim a chegada de um *slot* reservado para um determinado terminal fará com que o estado dele passe para *fragmento velho com reserva* ou para *terminal sem reserva*, conforme o estado anterior. A chegada de um *slot* vazio fará com que o estado do terminal cuja mensagem será enviada no *slot* passe para *fragmento velho com reserva*. Já a chegada de um outro fragmento fará com que o terminal fique no estado *fragmento novo com reserva*.

O `motrap_data_base` (especificado formalmente em A.3.3) é de fato uma memória onde ficam guardados os envelopes da APL, os estados de cada terminal e a fila de novas mensagens da APL. Como tal memória é acessada para escrita tanto pelo `motrap_high_core` quanto pelos `motrap_low_cores` é preciso gerenciar este acesso. A política adotada foi a de garantir o acesso de escrita um por vez a cada um dos componentes, ciclicamente (*time-*

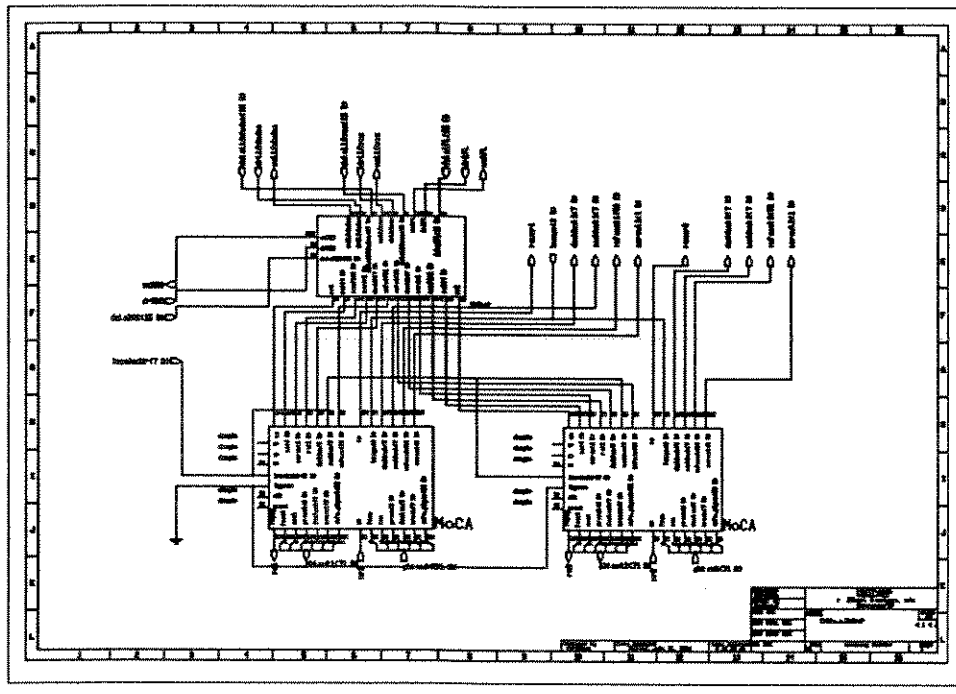


Figura 5.17: MoCAs e MoTraP

sharing). Quem controla este acesso é o componente `motrap_scheduler` (especificado formalmente em A.3.4).

Para ilustrar o funcionamento do MoTraP usaremos 3 dos 7 casos usados para exemplificar o MoCA e mais dois. Os 3 casos do MoCA usados são aqueles que utilizam o MoTraP: (1) o envio de um pacote tipo 1, (2) o envio de um pacote tipo 2, e o (6) o retorno de um pacote tipo 1. Além disso criamos o caso (8) e o (9), que, como o caso (1), correspondem ao envio de dois pacotes do tipo 1, porém que chegam à MAC consecutivamente. O objetivo aqui é testar o funcionamento da fila de mensagens novas da APL. Além disto os casos (2) e (8) usam o anel 2, testando assim a utilização simultânea de dois aneis pelo MoTraP. Outro modificação em relação a simulação apenas com o MoCA foi que nos casos (1) e (2) a mensagem do caso (2) (LLC/dados) chegou antes à MAC do que a mensagem do caso (1) (APL), apesar disso a mensagem da APL foi enviada primeiro, já que, no instante da chegada de um *slot* vazio, havia duas mensagens para serem transmitidas, sendo enviada a com maior prioridade.

O cenário utilizado está esquematizado na Fig. 5.17. As entradas

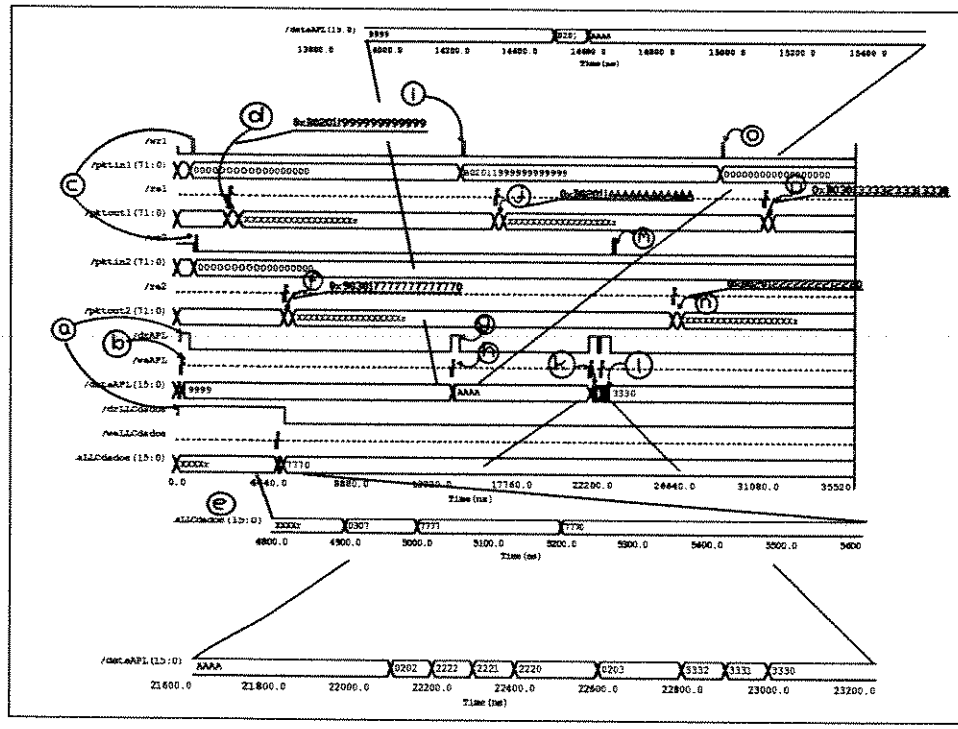


Figura 5.18: Simulação do MoTraP

dos MoCAs provenientes da PHY assim como as entradas do MoTraP provenientes da APL, da LLC/voz e LLC/dados foram geradas via AMPLE. As saídas analisadas foram dos MoCAs para a PHY e do MoTraP para os usuários. Foram ignoradas nessa seção as interfaces com o MoDEn e com o MoTI.

A Fig. 5.18 nos ajudará a entender os resultados da simulação do MoTraP. O primeiro evento relevante é a chegada de um envelope da LLC/dados e logo em seguida um da APL (indicado pelo ponto a). O Recebimento do envelope da APL é imediato (b), ficando guardado na fila de envelopes novos do MoTraP. Com a chegada de dois *slots* vazios (c) o MoTraP atenderá um de cada vez, conforme definido pelo componente `motrap_scheduler`. Neste caso o primeiro anel a ser atendido foi o 1, sendo a ele enviado o pacote da APL (d), que destina-se ao nó 2 e é proveniente do terminal 1 local. No tempo definido pelo `motrap_scheduler` o `motrap_low_core` solicita o envelope da LLC/dados (e), único disponível para ser enviado, encaminhando-o diretamente para o anel 2 (f). Em (g)

houve a chegada de outro envelope da APL, que novamente é atendido imediatamente[1] (h). O envelope recebido, corresponde a um envelope do terminal 1, sendo então guardado e o estado do terminal passa para *fragmento novo com reserva*. Em (i) houve o retorno do pacote enviado em (d), e como o terminal 1 já enviou um novo envelope, este é encaminhado ao anel 1 (j). Em (k) e (l) há a chegada de dois pacotes da APL consecutivamente. O primeiro proveniente do terminal 2 e o segundo do 3. Chega então um *slot* vazio no anel 2 (m), e o envelope do terminal 2 é então enviado (n). O mesmo ocorre no anel 1 e com o envelope do terminal 3 em (o) e (p).

5.4 MoDEn

Ao MoDEn cabe o repasse dos envelopes provenientes do MoCA ao respectivo usuário. O MoCA só encaminhará ao MoDEn os envelopes destinados aos usuários cujo sinal `busy` estiver %0. Como o MoDEn estará atendendo simultaneamente as duas MoCAs, e para evitar o conflito de uma eventual chegada de dois envelopes para um mesmo usuário, optamos por atender ora um MoCA, ora o outro, sempre atendendo aos dois no intervalo de tempo de um *slot*.

O MoDEn, cujo esquemático encontra-se na Fig. 5.19, é composto por 3 componentes: 2 `moden_core` e 1 `moden_scheduler`. O `moden_core`, recebe o envelope do MoCA, identifica o usuário e, quando o *C*hip *S*elector (`cs`) estiver ativo, envia o envelope para o usuário. O `moden_scheduler` controla devidamente os `css` dos `moden_cores`.

Para ilustrar o funcionamento do MoDEn utilizamos como gerador do sinais dos MoCAs os próprios MoCAs, simulando via AMPLE a chega dos pacotes na interface entre os MoCAs e a PHY. Analisando apenas a saída do MoDEn para os usuários APL, LLC/voz e LLC/dados. A Fig. 5.20 mostra nosso cenário.

A Fig. 5.21 nos ajudará a entender os resultados da simulação do MoDEn. No início temos a chegada quase simultânea de dois dados para a LLC/dados (a), que serão enviados um de cada vez, vide (b) e (c), para o usuário. Aqui, não há preocupação com relação a ordem de chegada. Em (d) temos a chegada de mais dois pacotes. No anel 1 para a APL no anel 2 para a LLC/voz, como a fila da LLC/voz está cheia, pois o `busyLLCvoz` está %1, apenas o envelope da APL foi encaminhado (f).

[1] De fato o tempo de atendimento da APL não é imediato, mas sim no tempo definido pelo `motrap_scheduler` para o `motrap_high_core` acessar o `motrap_data_base`

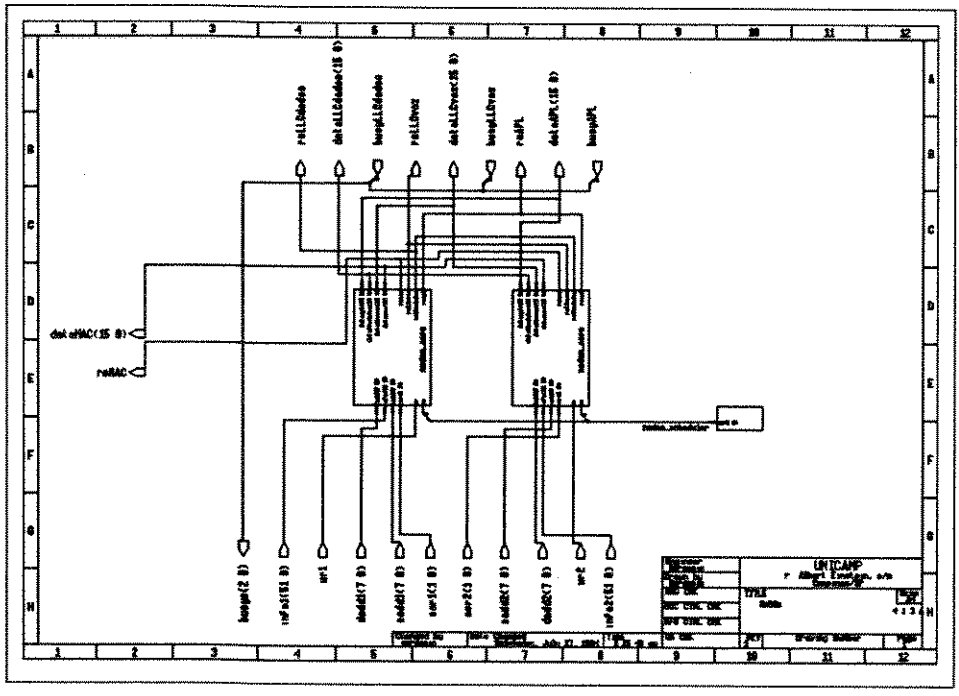


Figura 5.19: Esquemático do MoDEn

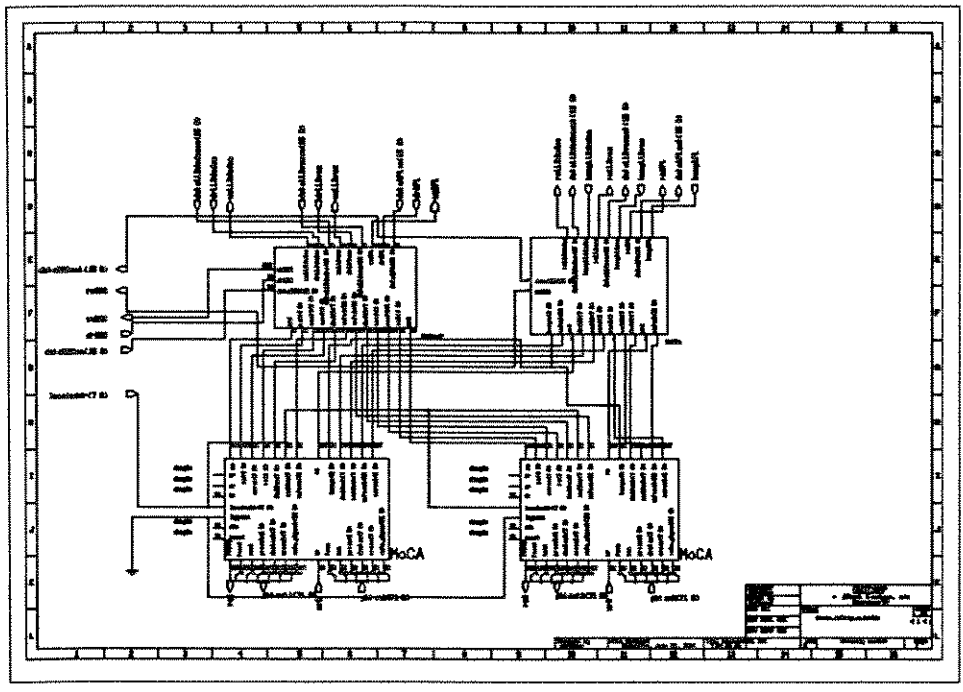


Figura 5.20: MoCAs, MoTraP e MoDEn

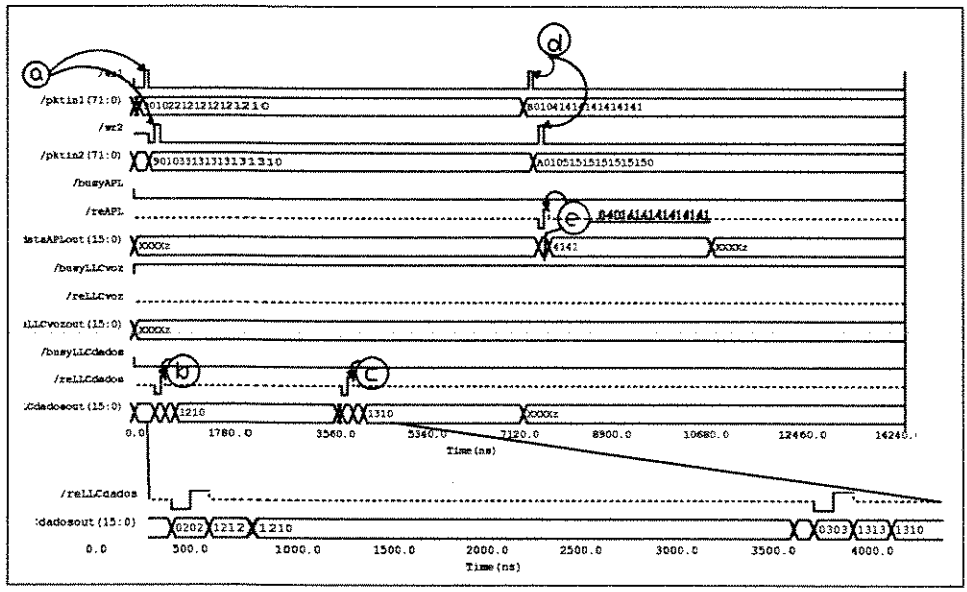


Figura 5.21: Simulação do MoDEn

5.5 MoConFis

O MoConFi controla a fila de saída/entrada. O MoConFiE e o MoConFiS são praticamente o mesmo componente, cuja descrição formal encontra-se em A.5, diferenciando-se apenas na propriedade **mode**, que pode ser **saída** ou **entrada**. Trata-se, na VHDL, de uma constante, cujo valor pode ser iniciado na captura esquemática (**da**). De fato os módulos MoConFiE e MoConFiS são encapsulamentos sobre o MoConFi, como pode ser visto nas Figs. 5.22 e 5.23. A diferença de comportamento do MoConFi quando no modo **entrada** para o modo **saída** é que o pino **st** indica no primeiro quando a fila não está vazia, e no segundo quando a fila está cheia.

Simulamos o recebimento de um envelope do usuário como exemplo do funcionamento do MoConFi. Aqui inicialmento o MoConFiE, ao solicitar o apontador de escrita percebe que há um envelope na fila, lê esse envelope, guarda-o a espera do pedido do usuário e finalmente atualiza o apontador de leitura na memória comum.

Na Fig. 5.24 vemos o resultado da simulação. O MoConFiE solicita ao MoIB o apontador de escrita (a), recebendo-o em (b). Em (c) solicita o dado que está na fila, recebendo em (d). Quando o MoTraP solicita o dado (e) ele é enviado. Fechando o ciclo o MoConFiE envia o novo apontador de

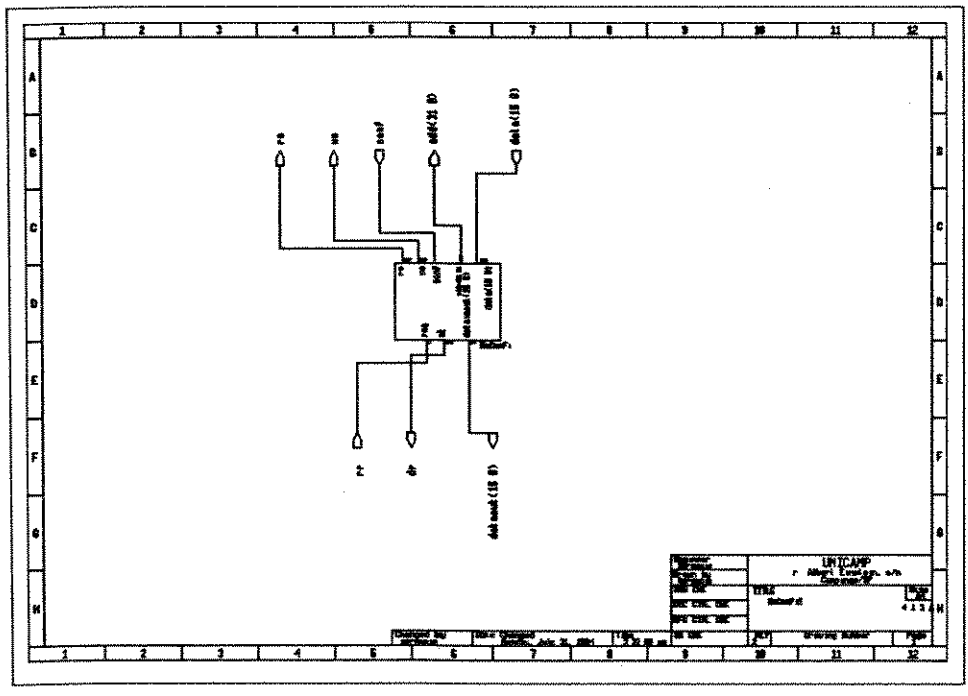


Figura 5.22: Esquemático do MoConFiE

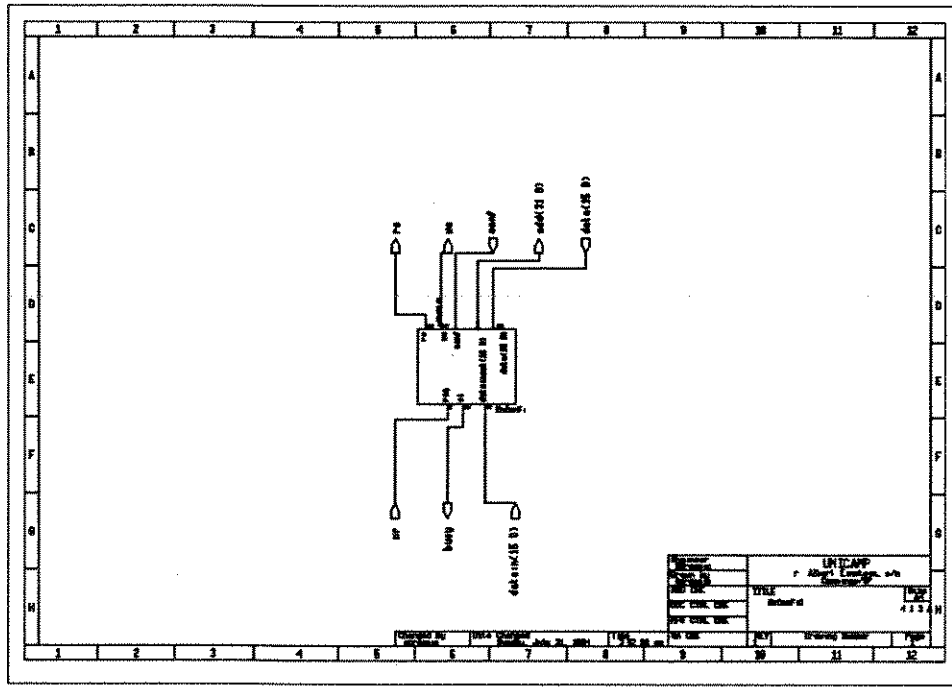


Figura 5.23: Esquemático do MoConFiS

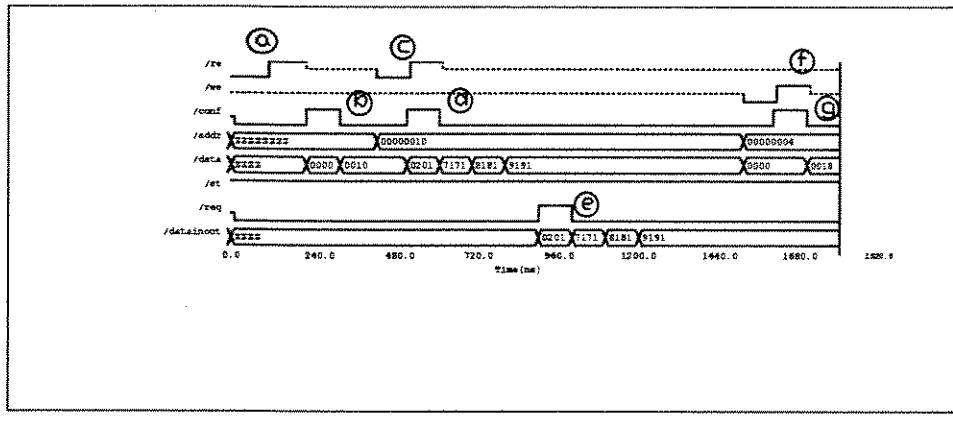


Figura 5.24: Simulação do MoConFi

leitura para o MoIB (f e g).

5.6 Conclusão

Neste capítulo mostramos a especificação formal da MAC de maneira análoga ao nosso desenvolvimento. Em primeiro lugar especificamos a interface da MAC e o seu funcionamento, em seguida especificamos a interface dos seus módulos e as ligações, formando o esquemático da MAC, para finalmente detalhar cada módulo, criando os componentes necessários.

Capítulo 6

As Conclusões

Os objetivos principais deste trabalho foram a especificação formal da camada de acesso ao meio — a camada MAC — da RALFO e a sua implementação, abrangendo as fases de criação, verificação e análise.

A RALFO foi proposta inicialmente por Paulo Guardieiro ([GM89]) como uma rede tipo anel de Cambridge modificada, visando ser uma rede de serviços integrados, que, na sua versão inicial, oferece ao usuário a possibilidade de transmitir dados e voz numa mesma rede.

A camada MAC foi totalmente especificada dentro da proposta do projeto no capítulo 3, através da definição e especificação funcional de sete módulos a saber: MoIB, MoConFiE, MoConFiS, MoTraP, MoDEn, MoCA e MoTI e a sua implementação foi apresentada no capítulo 5, utilizando o procedimento descrito no capítulo 4, qual seja, através de esquemáticos e na linguagem VHDL. Para facilitar a leitura a aqueles não familiarizados com o assunto, colocamos no capítulo 2, um breve resumo dos conceitos e questões ligados a redes de comunicações e no capítulo 4 descrevemos sucintamente metodologias de projeto de circuitos eletrônicos, principalmente aquelas baseadas em VHDL. Acreditamos que o nosso objetivo inicial apresentado na introdução foi alcançado, considerando que a proposta colocada na especificação da camada MAC, no capítulo 3, foi implementada e validada, através de diversos testes, no capítulo 5, dentro da metodologia de projeto *top-down* adotada.

Certamente o trabalho não se esgota aqui, importantes passos ainda devem ser dados até a concepção final do projeto: ele deve ser exaustivamente testado e analisado, para depois gerar a versão para fabricação, que por sua vez deve ser analisada e testada até a nível das placas, como por exemplo, mostrado na Fig. 4.2. Os trabalhos foram altamente facilitados pela existência das ferramentas da MGC, que também podem ser bastante

úteis para os passos seguintes do projeto.

Por outro lado, outros trabalhos podem ser desenvolvidos seguindo as metodologias aqui utilizadas ou sugeridas, como o uso de SDL, qpt, RTL, etc e sobre outros protocolos de comunicação como DQDB, ATM, SDH, etc. Finalmente, podemos arriscar a afirmação de que, seguindo os procedimentos semelhantes a aqueles adotados neste trabalho, é possível imaginar algumas das camadas hoje realizadas por meio de softwares, serem sintetizadas em hardwares, desde que a relação custo-benefício justifique.

Apêndice A

Especificações em VHDL

A.1 Especificação das Interfaces da MAC

```
--LIBRARY mgc_portable;
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_1164_extensions.all;
ENTITY MAC IS
GENERIC (
  CONSTANT Phydel : time := 500 ns ;
  CONSTANT Tclk : time := 100 ns);
  PORT (
     aplrein      : OUT    std_ulogic ;
     aplwein      : OUT    std_ulogic ;
     apladdin     : OUT    std_ulogic_vector(31 DOWNTO 0) ;
     rel          : OUT    std_ulogic ;
     pktout1      : OUT    std_ulogic_vector(71 DOWNTO 0) ;
     wr1          : IN     std_ulogic ;
     pktin1       : IN     std_ulogic_vector(71 DOWNTO 0) ;
     mi           : OUT    std_ulogic ;
     nnos         : OUT    std_ulogic ;
     de           : IN     std_ulogic ;
     localaddr    : IN     std_ulogic_vector(7 DOWNTO 0) ;
     llcdataconfin    : IN     std_ulogic ;
     llcdatadatain    : INOUT  std_ulogic_vector(15 DOWNTO 0) ;
     llcdatareout     : OUT    std_ulogic ;
     aplconfin    : IN     std_ulogic ;
     apldatain    : INOUT  std_ulogic_vector(15 DOWNTO 0) ;
     aplreout     : OUT    std_ulogic ;
     aplweout     : OUT    std_ulogic ;
     apladdout    : OUT    std_ulogic_vector(31 DOWNTO 0) ;
     aplconfout   : IN     std_ulogic ;
     apldataout   : INOUT  std_ulogic_vector(15 DOWNTO 0) ;
     pktin2       : IN     std_ulogic_vector(71 DOWNTO 0) ;
     wr2          : IN     std_ulogic ;
     pktout2      : OUT    std_ulogic_vector(71 DOWNTO 0) ;
     re2          : OUT    std_ulogic ;
     llcvozreout      : OUT    std_ulogic ;
     llcvozweout      : OUT    std_ulogic ;
     llcvozaddout     : OUT    std_ulogic_vector(31 DOWNTO 0) ;
     llcvozconfout    : IN     std_ulogic ;
     llcvozdataout    : INOUT  std_ulogic_vector(15 DOWNTO 0) ;
     llcdataweout     : OUT    std_ulogic ;
     llcdataaddout    : OUT    std_ulogic_vector(31 DOWNTO 0) ;
     llcdataconfout   : IN     std_ulogic ;
     llcdatadataout   : INOUT  std_ulogic_vector(15 DOWNTO 0) ;
     llcvozrein       : OUT    std_ulogic ;
     llcvozwein       : OUT    std_ulogic ;
     llcvozaddin      : OUT    std_ulogic_vector(31 DOWNTO 0) ;
     llcvozconfin     : IN     std_ulogic ;
     llcvozdatain     : INOUT  std_ulogic_vector(15 DOWNTO 0) ;
     clk          : IN     std_ulogic ;
```

```
    reset       : IN    std_ulogic ;
    llcdatarein   : OUT    std_ulogic ;
    llcdatawein   : OUT    std_ulogic ;
    llcdataaddin   : OUT    std_ulogic_vector(31 DOWNTO 0)
    ) ;
END MAC ;
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_1164_extensions.all;
ENTITY phyinter IS
GENERIC (
 CONSTANT Phydel : time ;
 CONSTANT  Tclk : time);
 PORT (
    re    : OUT   std_ulogic ;
    pktout: OUT    std_ulogic_vector(71 DOWNTO 0) ;
    wr    : IN    std_ulogic ;
    pktin  : IN    std_ulogic_vector(71 DOWNTO 0)
    ) ;
END phyinter;
ARCHITECTURE inter OF phyinter IS
-SIGNAL tmpre : std_ulogic := '0';
BEGIN
phy:
PROCESS ( wr )
   VARIABLE data : std_ulogic_vector(71 DOWNTO 0) ;
           - (f/e,m,pri,tgt,src,info) = (1,1,%11,0x01,0x02,E020114347094B3C80)
           - pktout(71) = f/e
           - pktout(70) = m
           - pktout(69 DOWNTO 68) = pri
           - pktout(67 DOWNTO 60) = tgt
           - pktout(59 DOWNTO 52) = src
           - pktout(51 DOWNTO 0)  = info
   VARIABLE phydel2 : time ;
   CONSTANT newdata  : std_ulogic_vector(71 DOWNTO 0)
    := "111100000001000000101110000000100000000100010100001101000111000010010100" ;
   CONSTANT newinfo  : std_ulogic_vector(51 DOWNTO 0)
    := "000100000001000000010000000100000000100000000100000001" ;
   CONSTANT emptypkt  : std_ulogic_vector(71 DOWNTO 0)
    := "0---------------------------------------------------------------" ;
   CONSTANT localaddress : std_ulogic_vector(7 DOWNTO 0) := "00000001" ;
   CONSTANT received : std_ulogic_vector(1 DOWNTO 0) := "11" ;
  BEGIN
   IF (wr'event AND wr = '1') THEN
    phydel2 := Phydel + Tclk ;
-    re <= '0';
    IF wr = '1' THEN - something arived
     data := pktin;
     IF data(71) = '0' THEN - empty
              pktout  <= newdata AFTER Phydel ;
              re <= '0',
                '1' AFTER Phydel ,
                '0' AFTER phydel2 ;
     ELSE
      IF data(59 DOWNTO 52) = localaddress THEN - from local
       IF data(69 DOWNTO 68) = "11" THEN - from APL
        pktout(71) <= '1' AFTER Phydel , - f/e = 1
         '-' AFTER phydel2 ;
        pktout(70) <= '0' AFTER Phydel , - m = 0
         '-' AFTER phydel2 ;
        pktout(69 DOWNTO 68) <= "00" AFTER Phydel , - pri = %11
         "-" AFTER phydel2 ;
        pktout(67 DOWNTO 60) <= "00000010" AFTER Phydel , - tgt = 0x02
         "--------" AFTER phydel2 ;
        pktout(59 DOWNTO 52) <= "00000001" AFTER Phydel , - src = 0x01
         "--------" AFTER phydel2 ;
        pktout(51 DOWNTO 0) <= newinfo ,
         "----------------------------------------------------" AFTER phydel2;
        re <= '0',
         '1' AFTER Phydel ,
         '0' AFTER phydel2 ;
       ELSE pktout <= emptypkt AFTER Phydel; - not from APL
        re <= '0',
         '1' AFTER Phydel ,
         '0' AFTER phydel2 ;
       END IF;
      ELSE
       IF data(67 DOWNTO 60) = localaddress THEN - not from local and to local
        pktout(71 DOWNTO 2) <= data(71 DOWNTO 2) AFTER Phydel;
        pktout(1 DOWNTO 0) <= received;
        re <= '0',
         '1' AFTER Phydel ,
         '0' AFTER phydel2 ;
```

```
          ELSE pktout(71 DOWNTO 0) <= data(71 DOWNTO 0) AFTER Phydel; -- not to local
           re <= '0',
             '1' AFTER Phydel ,
             '0' AFTER phydel2 ;
          END IF;
        END IF;
      END IF;
    END IF;
  END IF;
 END PROCESS phy;
END inter;
ARCHITECTURE interMACE OF MAC IS
 CONSTANT readPtrAdd: std_ulogic_vector(31 DOWNTO 0) := "00000000000000000000000000000000";
 CONSTANT writePtrAdd: std_ulogic_vector(31 DOWNTO 0) := "00000000000000000000000000000100";
BEGIN
 sending_data:
 PROCESS
  VARIABLE readPtr : std_ulogic_vector (31 DOWNTO 0) := "00000000000000000000000000001000";
BEGIN
  llcvozwein <= '0';
  llcvozrein <= '0';
-- Asking about writePtr
  llcvozaddin <= writePtrAdd;
  llcvozwein <= '0' AFTER 0 ns,
    '1' AFTER Tclk ,
    '0' AFTER 2*Tclk ;
  WAIT UNTIL llcvozconfin = '1';
  WAIT FOR Phydel; -- time to compare address
-- Sending data
--   llcvozdatain <= "10010100011111100";
  llcvozaddin <= readPtr;
  readPtr := readPtr + "10";
  llcvozwein <= '1' AFTER 0 ns,
    '0' AFTER Tclk;
  WAIT UNTIL llcvozconfin = '1';
  WAIT FOR Tclk;
--   llcvozdatain <= "00010111100111110";
  readPtr := readPtr + "10";
  WAIT FOR Tclk;
--   llcvozdatain <= "11001011101101110";
  readPtr := readPtr + "10";
  WAIT FOR Tclk;
--   llcvozdatain <= "01110101011000010";
  readPtr := readPtr + "10";
  WAIT FOR Tclk;
  llcvozdatain <= "----------";
  llcvozaddin <= "--------------------";
-- Sending readPtr
  llcvozdatain <= readPtr(15 DOWNTO 0);
  llcvozaddin <= readPtrAdd;
  llcvozrein <= '0' AFTER 0 ns,
    '1' AFTER Tclk,
    '0' AFTER 2*Tclk;
  WAIT UNTIL llcvozconfin = '1';
  WAIT FOR Tclk;
  llcvozdatain <= "0000000000000000";
  WAIT FOR Tclk;
  llcvozdatain <= "0000000000000000";
  WAIT FOR Tclk;
  llcvozdatain <= "0000000000000000";
  WAIT FOR Tclk;
  llcvozdatain <= "----------";
  llcvozaddin <= "--------------------";
  WAIT FOR 300 ns;
 END PROCESS sending_data;
END interMACE;
ARCHITECTURE interMACS OF MAC IS
 CONSTANT writePtrAdd: std_ulogic_vector(31 DOWNTO 0) := "00000000000000000000000000000100";
 CONSTANT readPtrAdd: std_ulogic_vector(31 DOWNTO 0) := "00000000000000000000000000000000";
BEGIN
 sending_data:
 PROCESS
  VARIABLE writePtr : std_ulogic_vector (31 DOWNTO 0) := "00000000000000000000000000010000";
BEGIN
  aplweout <= '0';
  aplreout <= '0';
-- Asking about readPtr
  apladdout <= readPtrAdd;
  aplweout <= '0' AFTER 0 ns,
    '1' AFTER Tclk,
    '0' AFTER 2*Tclk;
  WAIT UNTIL aplconfout = '1';
  WAIT FOR Phydel; -- time to compare address
```

```
- Sending data
   apldataout <= "1001010011101110";
   apladdout <= writePtr;
   writePtr := writePtr + "10";
   aplreout <= '1' AFTER 0 ns,
    '0' AFTER Tclk;
   WAIT UNTIL aplconfout = '1';
   WAIT FOR Tclk;
-   apldataout <= "1111011010011110";
   writePtr := writePtr + "10";
   WAIT FOR Tclk;
-   apldataout <= "0000000000000000";
   writePtr := writePtr + "10";
   WAIT FOR Tclk;
-   apldataout <= "0000000000000000";
   writePtr := writePtr + "10";
   WAIT FOR Tclk;
-   apldataout <= "----------";
   apladdout <= "----------";
- Sending writePtr
   apldataout <= writePtr(15 DOWNTO 0);
   apladdout <= writePtrAdd;
   aplreout <= '0' AFTER 0 ns,
    '1' AFTER Tclk,
    '0' AFTER 2*Tclk;
   WAIT UNTIL aplconfout = '1';
   WAIT FOR Tclk;
   apldataout <= "0000000000000000";
   WAIT FOR Tclk;
   apldataout <= "0000000000000000";
   WAIT FOR Tclk;
   apldataout <= "0000000000000000";
   WAIT FOR Tclk;
   apldataout <= "----------";
   apladdout <= "----------";
   WAIT FOR 300 ns;
  END PROCESS sending_data;
END interMACS;
ARCHITECTURE interMACPHY OF MAC IS
COMPONENT phyinter
GENERIC (
  CONSTANT Phydel : IN time ;
   Tclk : IN time );
  PORT (
   SIGNAL re : OUT std_ulogic ;
   SIGNAL pktout : OUT std_ulogic_vector(71 DOWNTO 0) ;
   SIGNAL wr : IN std_ulogic ;
   SIGNAL pktin : IN std_ulogic_vector(71 DOWNTO 0)
  );
END COMPONENT ;
FOR ALL : phyinter USE ENTITY WORK.phyinter
-   GENERIC MAP ( 500 ns ,
-    100 ns )
;
BEGIN
  phy1: phyinter
   GENERIC MAP (Phydel, Tclk)
   PORT MAP ( re1 ,
    pktout1 ,
    wr1 ,
    pktin1 );
  phy2: phyinter
   GENERIC MAP (Phydel, Tclk)
   PORT MAP ( re2 ,
   pktout2 ,
   wr2 ,
   pktin2 );
END interMACPHY ;
```

A.2 Especificação do MoCA

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_1164_extensions.all;
ENTITY MoCA IS
  GENERIC (
    CONSTANT Phydel: time := 500ns;
    CONSTANT  Tclk: time := 100ns;
    CONSTANT  Tclk2: time := 200ns;
    CONSTANT  Tclkn: time := 500ns --;
--    CONSTANT Broadcast: std_logic_vector(7 DOWNTO 0)
--      := "11111111"
  );
  PORT (
      we: OUT std_logic := 'Z';
      si: OUT std_logic_vector(4 DOWNTO 0)
        := "ZZZZZ";
      destout: OUT std_logic_vector(7 DOWNTO 0)
        := "ZZZZZZZZ";
      srcout: OUT std_logic_vector(7 DOWNTO 0)
        := "ZZZZZZZZ";
      info_phyout: OUT std_logic_vector(51 DOWNTO 0)
        := "ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ";
      infoout:  OUT std_logic_vector(51 DOWNTO 0)
        := "ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ";
      saddout: OUT std_logic_vector(7 DOWNTO 0)
        := "ZZZZZZZZ";
      mi: IN std_logic;
      sr: OUT std_logic := 'Z';
      mr: OUT std_logic := 'Z';
      min: IN std_logic;
      saddin: IN std_logic_vector(7 DOWNTO 0);
      infoin: IN std_logic_vector(51 DOWNTO 0);
      daddout: OUT std_logic_vector(7 DOWNTO 0)
        := "ZZZZZZZZ";
      errout: OUT std_logic_vector(1 DOWNTO 0)
        := "ZZ";
      destin: IN std_logic_vector(7 DOWNTO 0);
      srcin: IN std_logic_vector(7 DOWNTO 0);
      localaddr: IN std_logic_vector(7 DOWNTO 0);
      rephy: OUT std_logic
        := 'Z';
      ri: IN std_logic_vector(1 DOWNTO 0);
      priin: IN std_logic_vector(1 DOWNTO 0);
      re: OUT std_logic
        := 'Z';
      frout: OUT std_logic
        :='Z';
      wr: IN std_logic;
      frin: IN std_logic;
      moout: OUT std_logic
        := 'Z';
      priout: OUT std_logic_vector(1 DOWNTO 0)
        := "ZZ";
      errin: IN std_logic_vector(1 DOWNTO 0);
      busys: IN std_logic_vector(2 DOWNTO 0);
      reset: IN std_logic;
      clk: IN std_logic;
      info_phyin: IN std_logic_vector(51 DOWNTO 0);
      daddin: IN std_logic_vector(7 DOWNTO 0);
         bypass: IN std_logic
    );
 END MoCA;
 ARCHITECTURE functional OF MoCA IS
   CONSTANT Z1 : std_logic
     := 'Z';
   signal waiting_apl_data,
     waitingAPLdataOff,
     waitingAPLdataOn,
     waiting_new_data--,
--    waiting_new_data_clt,
--    waiting_new_data_srv
     : std_logic
       := 'Z';
   procedure pulse(
     signal sig: out std_logic
   ) is
   begin
     sig <= '0',
       'Z' after Tclk,
       'Z' after Tclk2;
```

```
end pulse;
procedure put1(
  signal sig: out std_logic;
  constant val: std_logic
) is
begin
  sig <= val, Z1 after Tclkn;
end put1;
procedure put(
  signal sig: out std_logic_vector;
  constant val: std_logic
) is
variable i: natural;
begin
  FOR i IN sig'range loop
--    put1(sig(i), val); -- Compiler doesn't permit this
    sig(i) <= val, Z1 after Tclkn;
  end loop;
end put;
procedure putn(
  signal sig: out std_logic_vector;
  constant val: std_logic_vector
) is
variable i, j: natural;
begin
  FOR i IN sig'range loop
--    put1(sig(i), val(i));-- Compiler doesn't permit this
    j := i; -- Is it possible to j <> i? If so I could use putn(a(3 downto 0), b(103 downto 1000))!
    sig(i) <= val(j), Z1 after Tclkn;
  end loop;
end putn;
FUNCTION and2 (
  SIGNAL a , b : IN std_logic
) RETURN std_logic IS
BEGIN
  IF a = '0' or b = '0' THEN
    RETURN '0';
  elsif a = '1' or b = '1' then
    return '1';
  ELSE
    return a and b;
  END IF;
END and2;
PROCEDURE send_empty(
-- to PHY
  signal rephy : out std_logic;
  signal feout : out std_logic;
  signal mout : out std_logic;
  signal priout : out std_logic_vector(1 downto 0);
  signal destout : out std_logic_vector(7 downto 0);
  signal srcout : out std_logic_vector(7 downto 0);
  signal info_phyout : out std_logic_vector(51 downto 0)
) IS
BEGIN
  put1(feout,'0');
  put1(mout, '0');
  put(priout, '-');
  put(destout, '-');
  put(srcout, '-');
  put(info_phyout,'-');
  pulse(rephy);
END send_empty;
PROCEDURE receive(
-- from PHY
  signal priin : in std_logic_vector(1 downto 0);
  signal destin : in std_logic_vector(7 downto 0);
  signal srcin : in std_logic_vector(7 downto 0);
  signal info_phyin : in  std_logic_vector(51 downto 0);
-- to PHY
  signal rephy : out std_logic;
  signal feout : out std_logic;
  signal mout : out std_logic;
  signal priout : out std_logic_vector(1 downto 0);
  signal destout : out std_logic_vector(7 downto 0);
  signal srcout : out std_logic_vector(7 downto 0);
-- from user
-- to user
  signal re : out std_logic;
  signal aerout : OUT std_logic_vector(1 downto 0);
  signal daddout : out std_logic_vector(7 downto 0);
  signal saddout : out std_logic_vector(7 downto 0);
  signal infoout : out std_logic_vector(51 downto 0)
) IS
```

```
  BEGIN
    putn(serout, priin);
    putn(daddout, destin);
    putn(saddout, srcin);
    putn(infoout, info'phyin);
    pulse(re);
    put1(feout, '1');
    put1(mout,'0');
    putn(priout,priin);
    putn(destout, destin);
    putn(srcout, srcin);
    pulse(rephy);
  END receive;
PROCEDURE receive1(
- from PHY
  signal priin : in std_logic_vector(1 downto 0);
  signal destin : in std_logic_vector(7 downto 0);
  signal srcin : in std_logic_vector(7 downto 0);
  signal info'phyin : in std_logic_vector(51 downto 0);
- to PHY
  signal rephy : out std_logic;
  signal feout : out std_logic;
  signal mout : out std_logic;
  signal priout : out std_logic_vector(1 downto 0);
  signal destout : out std_logic_vector(7 downto 0);
  signal srcout : out std_logic_vector(7 downto 0);
  signal info'phyout : out std_logic_vector(51 downto 0);
- from user
- to user
  signal re : out std_logic;
  signal serout : OUT std_logic_vector(1 downto 0);
  signal daddout : out std_logic_vector(7 downto 0);
  signal saddout : out std_logic_vector(7 downto 0);
  signal infoout : out std_logic_vector(51 downto 0)
) IS
BEGIN
  receive(priin, destin, srcin, info'phyin,
    rephy, feout, mout, priout, destout, srcout,
    re, serout, daddout, saddout, infoout);
  putn(info'phyout, info'phyin);
END receive1;
procedure receive2 (
- from PHY
  signal priin : in std_logic_vector(1 downto 0);
  signal destin : in std_logic_vector(7 downto 0);
  signal srcin : in std_logic_vector(7 downto 0);
  signal info'phyin : in std_logic_vector(51 downto 0);
- to PHY
  signal rephy : out std_logic;
  signal feout : out std_logic;
  signal mout : out std_logic;
  signal priout : out std_logic_vector(1 downto 0);
  signal destout : out std_logic_vector(7 downto 0);
  signal srcout : out std_logic_vector(7 downto 0);
  signal info'phyout : out std_logic_vector(51 downto 0);
- from user
- to user
  signal re : out std_logic;
  signal serout : OUT std_logic_vector(1 downto 0);
  signal daddout : out std_logic_vector(7 downto 0);
  signal saddout : out std_logic_vector(7 downto 0);
  signal infoout : out std_logic_vector(51 downto 0)
) is
begin
  receive(priin, destin, srcin, info'phyin,
    rephy, feout, mout, priout, destout, srcout,
    re, serout, daddout, saddout, infoout);
  putn(info'phyout(51 downto 4), info'phyin(51 downto 4));
  put(info'phyout(3 downto 2), '1');
  putn(info'phyout(1 downto 0), info'phyin(1 downto 0));
end receive2;
procedure receivemac (
- from PHY
  signal priin : in std_logic_vector(1 downto 0);
  signal destin : in std_logic_vector(7 downto 0);
  signal srcin : in std_logic_vector(7 downto 0);
  signal info'phyin : in std_logic_vector(51 downto 0);
- to PHY
- from user
  signal we : out std_logic;
  signal si : out std_logic_vector(4 downto 0);
- to user
  signal re : out std_logic;
```

```
    signal serout : OUT std_logic_vector(1 downto 0);
    signal daddout : out std_logic_vector(7 downto 0);
    signal saddout : out std_logic_vector(7 downto 0);
    signal infoout : out std_logic_vector(51 downto 0);
    signal waitingAPLdataOn : out std_logic
  ) is
    VARIABLE i : integer;
  begin
    putn(daddout, destin);
    putn(saddout, srcin);
    putn(infoout, info_phyin);
    putn(serout, priin);
    pulse(re);
    put1(si(4), '0');
    put1(si(3), '1');
    put(si(2 downto 0), "-");
    pulse(we);
    pulse(waitingAPLdataOn);
end receivemac;
  procedure reject(
  - from PHY
    signal fein : in std_logic;
    signal min : in std_logic;
    signal priin : in std_logic_vector(1 downto 0);
    signal destin : in std_logic_vector(7 downto 0);
    signal srcin : in std_logic_vector(7 downto 0);
  - to PHY
    signal rephy : out std_logic;
    signal feout : out std_logic;
    signal mout : out std_logic;
    signal priout : out std_logic_vector(1 downto 0);
    signal destout : out std_logic_vector(7 downto 0);
    signal srcout : out std_logic_vector(7 downto 0)
  - from user
  - to user
  ) is
  begin
    put1(feout, fein);
    put1(mout, min);
    putn(priout, priin);
    putn(destout, destin);
    putn(srcout, srcin);
    pulse(rephy);
  end reject;
  procedure reject1(
  - from PHY
    signal fein : in std_logic;
    signal min : in std_logic;
    signal priin : in std_logic_vector(1 downto 0);
    signal destin : in std_logic_vector(7 downto 0);
    signal srcin : in std_logic_vector(7 downto 0);
    signal info_phyin : in std_logic_vector(51 downto 0);
  - to PHY
    signal rephy : out std_logic;
    signal feout : out std_logic;
    signal mout : out std_logic;
    signal priout : out std_logic_vector(1 downto 0);
    signal destout : out std_logic_vector(7 downto 0);
    signal srcout : out std_logic_vector(7 downto 0);
    signal info_phyout : out std_logic_vector(51 downto 0)
  - from user
  - to user
  ) is
  begin
    reject(fein, min, priin, destin, srcin,
      rephy, feout, mout, priout, destout, srcout);
    putn(info_phyout, info_phyin);
  end reject1;
  procedure reject2(
  - from PHY
    signal fein : in std_logic;
    signal min : in std_logic;
    signal priin : in std_logic_vector(1 downto 0);
    signal destin : in std_logic_vector(7 downto 0);
    signal srcin : in std_logic_vector(7 downto 0);
    signal info_phyin : in std_logic_vector(51 downto 0);
  - to PHY
    signal rephy : out std_logic;
    signal feout : out std_logic;
    signal mout : out std_logic;
    signal priout : out std_logic_vector(1 downto 0);
    signal destout : out std_logic_vector(7 downto 0);
    signal srcout : out std_logic_vector(7 downto 0);
```

```
    signal info'phyout : out std'logic'vector(51 downto 0)
  - from user
  - to user
  ) is
    VARIABLE i : integer;
  begin
    reject(fein, min, priin, destin, srcin,
      rephy, feout, mout, priout, destout, srcout);
    putn(info'phyout(51 downto 4), info'phyin(51 downto 4));
    put1(info'phyout(3), '0');
    put1(info'phyout(2), '1');
    putn(info'phyout(1 downto 0), info'phyin(1 downto 0));
  end reject2;
BEGIN
-  waiting'new'data <= and2(waiting'new'data'srw, waiting'new'data'clt);
phy:
PROCESS ( wr )
    constant APL : std'logic'vector(1 downto 0)
      := "11";
    variable i, j : natural;
  BEGIN
    IF (wr'event AND wr = '1') THEN - somethimg arived
      -waiting'new'data'clt <= '0';
      waiting'new'data <= '0';
      IF fein = '0' THEN - empty
        putn(si, "--00");
        pulse(we);
        -waiting'new'data'clt <= '1';
        waiting'new'data <= '1';
      ELSE
        IF srcin = localaddr AND bypass /= '1' THEN - from local
          IF priin = APL THEN - from APL
            put1(si(4), '1');
            put1(si(0), info'phyin(48));
            put1(si(1), info'phyin(49));
            put1(si(2), info'phyin(50));
            put1(si(3), info'phyin(51));
            pulse(we);
            pulse(waitingAPLdataOn);
          ELSE
            putn(daddout, destin);
            putn(saddout, srcin);
            putn(infoout, info'phyin);
            putn(serout, priin);
            pulse(re);
            send'empty(rephy, feout, mout,
              priout, destout, srcout, info'phyout);
          END IF;
        ELSE
          IF ((destin = localaddr OR destin = "11111111") AND bypass /= '1' ) THEN
-Broadcast) AND bypass /= '1' ) THEN  - Sory bmt I don't know how to set Broadcast in QuickSim :-(
            case priin is
              when "11" => if busys(2) /= '1' then
                 receive1(priin, destin, srcin, info'phyin,
                   rephy, feout, mout,
                   priout, destout, srcout, info'phyout,
                    re, serout,daddout,saddout,infoout);
                else
                  reject1(fein, min, priin,
                    destin, srcin, info'phyin,
                    rephy, feout, mout, priout,
                    destout, srcout, info'phyout);
                end if;
              when "10" => if busys(1) /= '1' then
                 receive2(priin, destin, srcin, info'phyin,
                   rephy, feout, mout,
                   priout, destout, srcout, info'phyout,
                    re, serout,daddout,saddout,infoout);
                else
                  reject2(fein, min, priin,
                    destin, srcin, info'phyin,
                    rephy, feout, mout, priout,
                    destout, srcout, info'phyout);
                end if;
              when "01" => if busys(0) /= '1' then
                 receive2(priin, destin, srcin, info'phyin,
                   rephy, feout, mout,
                   priout, destout, srcout, info'phyout,
                    re, serout,daddout,saddout,infoout);
                else
                  reject2(fein, min, priin,
                    destin, srcin, info'phyin,
                    rephy, feout, mout, priout,
```

```
            destout, srcout, info'phyout);
          end if;
        when "00" => receivemac(priin, destin, srcin,
          info'phyin,
          we, si ,
          re, serout, daddout, saddout, infoout,
          waitingAPLdataOn);
        when others => NULL;
       end case;
      ELSE
       putl(feout, fein);
       putl(mout, min);
       putn(priout, priin);
       putn(destout, destin);
       putn(srcout, srcin);
       putn(info'phyout, info'phyin);
       pulse(rephy);
      END IF;
     END IF;
    END IF;
   END IF;
  END PROCESS phy;
  new'data :
  PROCESS (waiting'new'data,ri)
  BEGIN
   if ((waiting'new'data = '1') AND ( ri = "11" OR ri = "10" )) then
-    waiting'new'data'srv <= '0';
    IF ri(1) = '1' THEN
     putl(feout, '1');
     putl(mout, '0');
     putn(priout, serin);
     putn(destout, daddin);
     putn(srcout, saddin);
     putn(info'phyout, infoin);
     pulse(rephy);
    ELSE
     send'empty(rephy, feout, mout, priout, destout, srcout,
      info'phyout);
    END IF;
   else
-    waiting'new'data'srv <= '1';
   end if;
  END PROCESS new'data;
  new'apl'data :
  PROCESS (waiting'apl'data, ri)
  begin
   if(waiting'apl'data = '1' AND ( ri = "11" OR ri = "10" ) ) then
    pulse(waitingAPLdataOff);
    IF ri(1) = '1' THEN
     putl(feout, '1');
     putl(mout, '0');
     putn(priout, serin);
     putn(destout, daddin);
     putn(srcout, saddin);
     putn(info'phyout, infoin);
     pulse(rephy);
    ELSE
     send'empty(rephy, feout, mout, priout, destout, srcout,
      info'phyout);
    END IF;
   end if;
  END PROCESS new'apl'data;
  waitingAPLdataControl:
  process (waitingAPLdataOn, waitingAPLdataOff)
  begin
   if(waitingAPLdataOn'event and waitingAPLdataOn = '1') then
    waiting'apl'data <= '1';
   end if;
   if(waitingAPLdataOff'event and waitingAPLdataOff = '1') then
    waiting'apl'data <= '0';
   end if;
  end process waitingAPLdataControl;
```

A.3 Especificação do MoTraP

A.3.1 Especificação do motrap_low_core

```
library lib;
use lib.pac.all;
library ieee;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_1164_extensions.all;
use work.motrappac.all;
ENTITY motrap_low_core IS
  PORT (
      cs: in std_logic;
      drmac: IN std_logic;
      datallcvoz: IN std_logic_vector(15 DOWNTO 0);
      datallcdados: IN std_logic_vector(15 DOWNTO 0);
      rr: IN std_logic;
      si: IN std_logic_vector(4 DOWNTO 0);
      conf: OUT std_logic_vector(1 DOWNTO 0)
        := "ZZ";
      ser: OUT std_logic_vector(1 DOWNTO 0)
        := "ZZ";
      wemac: OUT std_logic
        := 'Z';
      wellcdados: OUT std_logic
        := 'Z';
      wellcvoz: OUT std_logic
        := 'Z';
      drllcdados: IN std_logic;
      drllcvoz: IN std_logic;
      dadd: OUT std_logic_vector(7 DOWNTO 0)
        := "ZZZZZZZZ";
      info: OUT std_logic_vector(51 DOWNTO 0)
        := "ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ";
      datamac: IN std_logic_vector(15 DOWNTO 0);
      wadr: out std_logic_vector(3 downto 0)
        := "ZZZZ";
      radr: out std_logic_vector(3 downto 0)
        := "ZZZZ";
      statusld, ld: out std_logic
        := 'Z';
      we: out std_logic
        := 'Z';
      din: in std_logic_vector(63 downto 0);
      stout: out std_logic_vector(1 downto 0)
        := "ZZ";
      stin: out std_logic_vector(1 downto 0)
        := "ZZ";
      newAPL: in std_ulogic;
      newout: out std_logic_vector(3 downto 0)
        := "ZZZZ";
      newin: in std_logic_vector(3 downto 0);
      newwadr: out std_logic_vector(3 downto 0)
        := "ZZZZ";
      newradr: out std_logic_vector(3 downto 0)
        := "ZZZZ";
      newld: out std_logic
        := 'Z';
      newwe: out std_logic
        := 'Z';
      getNewAPL: out std_logic
        := 'Z'
      );
END motrap_low_core;
ARCHITECTURE functional OF motrap_low_core IS
  procedure senddata(
    CONSTANT usr : std_logic_vector(1 downto 0); -- MAC, LLCdados, LLCvoz, APL
    SIGNAL we: out std_logic; -- we<usr>
    signal data: in std_logic_vector(15 downto 0); -- data<usr>
    signal dadd: out std_logic_vector(7 downto 0);
    signal ser: out std_logic_vector(1 downto 0);
    signal info: out std_logic_vector(51 downto 0);
    signal conf: out std_logic_vector(1 downto 0)
  ) is
  begin
    pulse(we);
    wait for Tclk;
    for i in 7 downto 0 loop
      dadd(i) <= data(8 + i), 'Z' after Tclkn;
    end loop;
```

```
    for i in 51 downto 48 loop
      info(i) <= data(i - 48), 'Z' after Tclkn;
    end loop;
    putn(ser, usr);
    WAIT FOR Tclk;
    for i in 47 downto 32 loop
      info(i) <= data(i - 32), 'Z' after Tclkn;
    end loop;
    WAIT FOR Tclk;
    for i in 31 downto 16 loop
      info(i) <= data(i - 16), 'Z' after Tclkn;
    end loop;
    WAIT FOR Tclk;
    for i in 15 downto 0 loop
      info(i) <= data(i), 'Z' after Tclkn;
    end loop;
    putn(conf, "11");
  end senddata;
  signal sendNewAPL: std_logic
    := 'Z';
  signal rrMem: std_logic := 'Z';
  signal siMem: std_logic_vector(4 downto 0)
    := "ZZZZZ";
  signal rrBuf: std_logic := 'Z';
  signal siBuf: std_logic_vector(4 downto 0)
    := "ZZZZZ";
BEGIN
  lets_work:
  PROCESS
    variable  i: natural;
  BEGIN
    WAIT ON ( rrBuf ) ;
    if rrBuf = '1' and cs = '1' then
      IF siBuf(4) = '1' THEN
        -- mande o pacote da APL
        putn(radr, siBuf (3 downto 0));
        wait for Tclk/10; putn(info, din(51 downto 0));
        putn(ser, APL);
        for i in 7 downto 0 loop
          dadd(i) <= din(56 + i), 'Z' after Tclkn;
        end loop;
        putn(conf, "11");
        -- newStatus(wadr, din, stout, ReservedOld, statusId);
        wadr <= siBuf(3 downto 0),
          "ZZZZ" after Tclkn;
        putn(stout, ReservedOld);
        pulse(statusId);
      ELSIF siBuf(4) = '0' THEN
        IF siBuf(3) = '1' THEN
          -- mande o pacote do MoTI
          ASSERT true
            REPORT "MoTI not implemented";
        ELSIF siBuf(3) = '0' THEN
          -- mande o pacote novo
          if(newAPL = '1') then
            -- mande o pacote novo da APL
            pulse(getNewAPL);
            wait for 2*Tclk/10; pulse(sendNewAPL);
          ELSIF drLLCvoz = '1'  THEN
            -- mande o pacote da LLC/voz
            senddata(LLCvoz, weLLCvoz, dataLLCvoz,
              dadd, ser, info, conf);
          ELSIF drLLCdados = '1' THEN
            -- mande o pacote da LLC/dados
            senddata(LLCdados, weLLCdados, dataLLCdados,
              dadd, ser, info, conf);
          ELSIF drMAC = '1' THEN
            -- mande o pacote da MAC
            senddata(MACusr, weMAC, dataMAC,
              dadd, ser, info, conf);
          ELSE
            -- n~ao h'a pacote para ser enviado
            putn(conf, "10");
          END IF ;
        END IF;
      END IF ;
    end if;
  END PROCESS lets_work ;
  PROCESS(sendNewAPL, din, cs)
    variable i: natural;
  BEGIN
    if(sendNewAPL = '1' and cs = '1') then
      -- mande o pacote novo da APL
```

```
      putn(info, din(51 downto 0));
      putn(conf, "11");
      putn(ser, APL);
      FOR i IN 7 DOWNTO 0  LOOP
         dadd(i) <= din(56 + i), 'Z' after Tclkn;
      END LOOP ;
    end if;
  END PROCESS;
chipSelector:
PROCESS ( cs , rr , si )
BEGIN
  if ( rr'event and rr = '1' ) then
    rrMem <= rr;
    siMem <= si;
  end if;
  if ( cs'event ) then
    if ( cs = '1' ) then
      rrBuf <= rrMem;
      rrMem <= '0';
      siBuf <= siMem;
      siMem <= "00000";
    elsif (cs = '0') then
      rrBuf <= 'Z';
      siBuf <= "ZZZZZ";
    end if;
  end if;
END PROCESS chipSelector ;
```

A.3.2 Especificação do motrap_high_core

```
library ieee;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_1164_extensions.all;
library lib;
use lib.pac.all;
use work.motrappac.all;
entity motrap_high_core is
  port(
    cs: in std_logic;
    data: IN std_logic_vector(15 DOWNTO 0);
    dr: IN std_logic;
    weAPL: out std_logic
      := 'Z';
    dout: out std_logic_vector(63 downto 0)
      := "ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ";
    wadr: out std_logic_vector(3 downto 0)
      := "ZZZZ";
    radr: out std_logic_vector(3 downto 0)
      := "ZZZZ";
    ld: out std_logic
      := 'Z';
    statusld: out std_logic
      := 'Z';
    we: out std_logic
      := 'Z';
    din: in std_logic_vector(63 downto 0);
    stout: out std_logic_vector(1 downto 0)
      := "ZZ";
    stin: in std_logic_vector(1 downto 0);
    newout: out std_logic_vector(3 downto 0)
      := "ZZZZ";
    newin: in std_logic_vector(3 downto 0);
    newwadr: out std_logic_vector(3 downto 0)
      := "0000";
    newradr: out std_logic_vector(3 downto 0)
      := "0000";
    newld: out std_logic
      := 'Z';
    newwe: out std_logic
      := 'Z';
    newAPL: out std_logic
      := '0';
    getNewAPL: in std_logic
);
end motrap_high_core;
ARCHITECTURE functional OF motrap_high_core IS
  signal getStatus, internalWeAPL,
    rd1, rd2, wr1, wr2: std_logic
    := 'Z';
  signal readPtr, writePtr : std_logic_vector(4 downto 0)
```

```
  := "00000";
 signal inputBuffer: std_logic_vector(63 downto 0);
 FUNCTION incPtr (
  ptr: integer RANGE -16 TO 16
 ) RETURN integer IS
  VARIABLE newptr: integer ;
 BEGIN
  newptr := (sig(ptr)*(abs(ptr) + 1)) mod 17;
  if ( newptr = 0 ) then
   newptr := - sig(ptr);
  end if;
  return newptr;
 END incPtr ;
 FUNCTION incPtr (
  ptr: std_logic_vector(4 downto 0)
 ) RETURN std_logic_vector IS
  variable newptr: std_logic_vector(4 downto 0);
 BEGIN
  newptr := std_logic_vector(std_ulogic_vector(ptr) + "00001");
  return newptr;
 END incPtr;
BEGIN
 -- newAPL <= and2(newAPL1, newAPL2);
 weAPL <= internalWeAPL;
 PROCESS
 BEGIN
  if (cs = '1' and dr = '1') then
   pulse(internalWeAPL);
  end if;
 WAIT ON dr, cs;
 END PROCESS;
 readStatus:
 process
 begin
  if(internalWeAPL'event and internalWeAPL = '1') then
   FOR i IN 63 DOWNTO 48 LOOP
    inputBuffer(i) <=  data(i - 48);
   END LOOP;
   WAIT FOR Tclk;
   FOR i IN 47 DOWNTO 32 LOOP
    inputBuffer(i) <= data(i - 32);
   END LOOP;
   WAIT FOR Tclk;
   FOR i IN 31 DOWNTO 16 LOOP
    inputBuffer(i) <= data(i - 16);
   END LOOP;
   WAIT FOR Tclk;
   FOR i IN 15 DOWNTO 0 LOOP
    inputBuffer(i) <= data(i);
   END LOOP;
   radr <= inputBuffer(51 downto 48),
    "ZZZZ" after Tclkn;
   pulse(getStatus);
  end if;
  wait on internalWeAPL;
 end process readStatus;
 newwe <= '1';
 we <= '1';
 process
 begin
  if(getStatus = '1') then
   CASE stin IS
    WHEN ReservedOld =>
     -- status novo: ReservedNew
     newStatus(wadr, inputBuffer, stout, ReservedNew, statusId);
    when ReservedNew =>
     -- status novo: NotReserved
     newStatus(wadr, inputBuffer, stout, NotReserved, statusId);
    when NotReserved =>
     -- coloque na fila de novas mensagens
    -- newStatus(wadr, inputBuffer, stout, NotReserved, statusId);
     newout <= inputBuffer(51 downto 48),
      "ZZZZ" after Tclkn;
     wait for Tclk/10; pulse(newId);
     wait for Tclk; pulse(wr1);
     IF writePtr(3 downto 0) = readPtr(3 downto 0) THEN
      if (writePtr(4) /= readPtr(4)) then
       pulse(rd1);
       pulse(wr2);
      end if;
     END IF ;
    when others => NULL;
   END CASE ;
```

```
      putn(dout, inputBuffer);
      wadr <= inputBuffer(51 downto 48),
        "ZZZZ" after Tclkn;
      wait for Tclk/10; pulse(ld);
    end if;
    wait on getStatus;
  end process;
  PROCESS
   BEGIN
    if(getNewAPL = '1') then
      putn(wadr, newin);
      putn(stout, ReservedNew);
      wait for Tclk/10; pulse(statusld);
      putn(radr, newin);
      wait for Tclk/10; pulse(rd2);
      newStatus(wadr, inputBuffer, stout, ReservedOld, statusld);
    end if;
    WAIT ON ( getNewAPL ) ;
  END PROCESS;
  process(readPtr, writePtr)
  begin
      if(readPtr = writePtr) then
        newAPL <= '0';
      else
        newAPL <= '1';
      end if;
  end process;
  increser:
  process(rd1,rd2,wr1,wr2)
  begin
    if(rd1'event and rd1 = '1') then
      readPtr <= incPtr(readPtr);
    end if;
    if(wr1'event and wr1 = '1') then
      writePtr <= incPtr(writePtr);
    end if;
    if(rd2'event and rd2 = '1') then
      readPtr <= incPtr(readPtr);
    end if;
    if(wr2'event and wr2 = '1') then
      writePtr <= incPtr(writePtr);
    end if;
    newradr <= readPtr(3 downto 0);
    newwadr <= writePtr(3 downto 0);
  end process increser;
```

A.3.3 Especificação do motrap_data_base

```
library ieee;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_1164_extensions.all;
library lib;
use lib.pac.all;
library auxs;
use auxs.all;
ENTITY motrap_data_base IS
  PORT (
    din: in std_logic_vector(63 downto 0);
    wadr: in std_logic_vector(3 downto 0);
    radr: in std_logic_vector(3 downto 0);
    ld: in std_logic;
    statusld: in std_logic;
    we: in std_logic;
    dout: out std_logic_vector(63 downto 0)
      := "ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ";
    stin: in std_logic_vector(1 downto 0);
    stout: out std_logic_vector(1 downto 0)
      := "ZZ";
    newin: in std_logic_vector(3 downto 0);
    newout: out std_logic_vector(3 downto 0)
      := "ZZZZ";
    newwadr: in std_logic_vector(3 downto 0);
    newradr: in std_logic_vector(3 downto 0);
    newld: in std_logic;
    newwe: in std_logic
  );
END motrap_data_base;
ARCHITECTURE functional OF motrap_data_base IS
COMPONENT mem16x2
   port(
```

```
    DIN1: in std_logic_vector(1 downto 0);
    RADR1: in std_logic_vector(3 downto 0);
    WADR1: in std_logic_vector(3 downto 0);
    DOUT1: out std_logic_vector(1 downto 0);
    LD1,WE1: in std_logic
  );
 END COMPONENT ;
 COMPONENT mem16x4
  port(
    DIN1: in std_logic_vector(3 downto 0);
    RADR1: in std_logic_vector(3 downto 0);
    WADR1: in std_logic_vector(3 downto 0);
    DOUT1: out std_logic_vector(3 downto 0);
    LD1,WE1: in std_logic
  );
 END COMPONENT ;
 COMPONENT mem16x64
  port(
    DIN1: in std_logic_vector(63 downto 0);
    RADR1: in std_logic_vector(3 downto 0);
    WADR1: in std_logic_vector(3 downto 0);
    DOUT1: out std_logic_vector(63 downto 0);
    LD1,WE1: in std_logic
  );
 END COMPONENT ;
 for status: mem16x2 use entity auxs.mem16x2;
 for newmen: mem16x4 use entity auxs.mem16x4;
 for mem: mem16x64 use entity auxs.mem16x64;
BEGIN
  -- status
 --   00 - ReservedOld   "
 --                      > with reserved slot
 --   01 - ReservedNew   /
 --   10 - NotReserved   > with out reserved slot
 status: mem16x2
  port map (stin,
    radr,
    wadr,
    stout,
    ld,
    we);
 newmen: mem16x4
  port map (newin,
    newradr,
    newwadr,
    newout,
    newld,
    newwe);
 mem: mem16x64
  port map (din,
    radr,
    wadr,
    dout,
    ld,
    we);
```

A.3.4 Especificação do motrap_scheduler

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_1164_extensions.all;
library lib;
use lib.pac.all;
ENTITY motrap_scheduler IS
  PORT (
    SIGNAL cs: OUT std_ulogic_vector(2 downto 0)
  );
END motrap_scheduler;
ARCHITECTURE functional OF motrap_scheduler IS
BEGIN
  PROCESS
  BEGIN
    cs <= "001";
    WAIT FOR Tframe/3;
    cs <= "010";
    wait for Tframe/3;
    cs <= "100";
    wait for Tframe/3;
  END PROCESS ;
```

A.4 Especificação do MoDEn

A.4.1 Especificação do moden_core

```
library lib;
use lib.pac.all;
library ieee;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_1164_extensions.all;
ENTITY moden_core IS
  PORT (
      cs: in std_logic;
      rellcdados: OUT std_logic
        := 'Z';
      dataapl: OUT std_logic_vector(15 DOWNTO 0)
        := "ZZZZZZZZZZZZZZZZ";
      datallcvoz: OUT std_logic_vector(15 DOWNTO 0)
        := "ZZZZZZZZZZZZZZZZ";
      datallcdados: OUT std_logic_vector(15 DOWNTO 0)
        := "ZZZZZZZZZZZZZZZZ";
      info: IN std_logic_vector(51 DOWNTO 0);
      dadd: IN std_logic_vector(7 DOWNTO 0);
      sadd: IN std_logic_vector(7 DOWNTO 0);
      wr: IN std_logic;
      ser: IN std_logic_vector(1 DOWNTO 0);
      remac: OUT std_logic
        := 'Z';
      datamac: OUT std_logic_vector(15 DOWNTO 0)
        := "ZZZZZZZZZZZZZZZZ";
      reapl: OUT std_logic
        := 'Z';
      rellcvoz: OUT std_logic
        := 'Z'
      ) ;
 END moden_core;
 ARCHITECTURE functional OF moden_core IS
  signal wrMem, wrBuf: std_logic
    := 'Z';
  signal serMem, serBuf: std_logic_vector(1 downto 0)
      := "ZZ";
  signal daddMem, daddBuf,
    saddMem, saddBuf: std_logic_vector(7 downto 0)
      := "ZZZZZZZZ";
  signal infoMem, infoBuf: std_logic_vector(51 downto 0)
    := "ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ";
  PROCEDURE receive (
    SIGNAL re : OUT std_logic;
    SIGNAL data : OUT std_logic_vector (15 DOWNTO 0);
    SIGNAL sadd : IN std_logic_vector (7 downto 0);
    SIGNAL info : IN std_logic_vector (51 downto 0)
  ) IS
  BEGIN
    data(15 downto 8) <= sadd;
    data(7 downto 4) <= "0000";
    data(3 downto 0) <= info(51 downto 48);
    pulse(re); wait for 2*Tclk;
    data <= info(47 downto 32),
      info(31 downto 16) after Tclk,
      info(15 downto 0) after 2*Tclk;
  END receive;
 BEGIN
  do_it:
  PROCESS
  BEGIN
    wait on wrBuf, cs;
    if( cs = '1' and wrBuf = '1' ) then
      CASE serBuf IS
          WHEN APL => receive(reAPL, dataAPL,
            saddBuf, infoBuf);
          WHEN LLCvoz => receive(reLLCvoz, dataLLCvoz,
            saddBuf, infoBuf);
          WHEN LLCdados => receive(reLLCdados, dataLLCdados,
            saddBuf, infoBuf);
          WHEN MACusr => receive(reMAC, dataMAC,
            saddBuf, infoBuf);
          WHEN OTHERS => NULL;
      END CASE ;
    elsif ( cs = '0' ) then
      dataAPL <= "ZZZZZZZZZZZZZZZZ";
      dataLLCvoz <= "ZZZZZZZZZZZZZZZZ";
```

```
    dataLLCdados <= "ZZZZZZZZZZZZZZZZ";
   end if;
 END PROCESS do'it;
 chipSelector:
 PROCESS
 BEGIN
   wait on cs, wr ;
   if ( wr'event and wr = '1' ) then
    if (cs = '0') then
      wrMem <= wr;
      infoMem <= info;
      daddMem <= dadd;
      saddMem <= sadd;
      serMem <= ser;
    elsif (cs = '1') then
      pulse(wrBuf);
      infoBuf <= info;
      daddBuf <= dadd;
      saddBuf <= sadd;
      serBuf <= ser;
    end if;
   end if;
   if (cs'event) then
    if (cs = '1') then
      if (wrMem = '1') then
        pulse(wrBuf);
        infoBuf <= infoMem;
        daddBuf <= daddMem;
        saddBuf <= saddMem;
        serBuf <= serMem;
        wait for Tclk/10;
          wrMem <= '0';
          infoMem <= "00000000000000000000000000000000000000000000000000000000";
          daddMem <= "00000000";
          saddMem <= "00000000";
          serMem <= "00";
      end if;
    elsif (cs = '0') then
      wrBuf <= 'Z';
      infoBuf <= "ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ";
      daddBuf <= "ZZZZZZZZ";
      saddBuf <= "ZZZZZZZZ";
      serBuf <= "ZZ";
    end if;
   end if;
  END PROCESS chipSelector;
END functional;
```

A.4.2 Especificação do moden_scheduler

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_1164_extensions.all;
library lib;
use lib.pac.all;
ENTITY moden_scheduler IS
  PORT (
     SIGNAL cs: OUT std_ulogic_vector(1 downto 0)
  );
END moden_scheduler;
ARCHITECTURE functional OF moden_scheduler IS
BEGIN
  PROCESS
  BEGIN
    cs <= "01";
    WAIT FOR Tframe/2;
    cs <= "10";
    wait for Tframe/2;
  END PROCESS ;
END functional;
```

A.5 Especificação do MoConFi

```
library ieee;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_1164_extensions.all;
library lib;
use lib.pac.all;
use work.moconfipac.all;
ENTITY moconfi IS
  GENERIC (
    CONSTANT ReadPtrAddr: std_logic_vector(31 downto 0)
      := "00000000000000000000000000001000";
    CONSTANT WritePtrAddr: std_logic_vector(31 downto 0)
      := "00000000000000000000000000000000";
    CONSTANT Size: natural := 16;
    CONSTANT mode: es := Entrada;
    CONSTANT Twait: time := 125us
  );
  PORT (
      st: OUT std_logic
        := 'Z';
      datainout: inout std_logic_vector(15 DOWNTO 0)
        := "ZZZZZZZZZZZZZZZZ";
      data: inout std_logic_vector(15 DOWNTO 0)
        := "ZZZZZZZZZZZZZZZZ";
      addr: OUT std_logic_vector(31 DOWNTO 0)
        := "ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ";
      re: OUT std_logic
        := 'Z';
      we: OUT std_logic
        := 'Z';
      conf: IN std_logic;
      req: IN std_logic
      );
END moconfi;
ARCHITECTURE functional OF moconfi IS
  signal start: std_logic
      := 'Z';
  signal inc,
    readAddrRequest,
    reqReadAddrRequest,
    resReadAddrRequest: std_logic
      := '0';
  signal stEnable: std_logic
      := '1';
  signal ptr, ptrAddr, tmpptr, tmpptr3: std_logic_vector(31 downto 0)
    := "ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ";
  signal addr1, addr2, addr3, readPtr, writePtr: std_logic_vector(31 downto 0)
    := "00000000000000000000000000000000";
  signal buf: std_logic_vector(63 downto 0)
    := "ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ";
  procedure buffering( signal output: out std_logic_vector;
    constant input: std_logic_vector) is
    variable i, j: natural;
  begin
    do_it : FOR i IN 3 downto 0 LOOP
      other_do_it : FOR j IN 15 downto 0 LOOP
        output(j+16*i) <= input(j);
      END LOOP other_do_it;
      wait for Tclk;
    END LOOP do_it ;
  end buffering;
  procedure unbuffering( signal output: out std_logic_vector;
    constant input: std_logic_vector) is
    variable i, j: natural;
  begin
    do_it : FOR i IN 3 downto 0 LOOP
      other_do_it : FOR j IN 15 downto 0 LOOP
        output(j) <= input(j+16*i);
      END LOOP other_do_it;
      wait for Tclk;
    END LOOP do_it ;
  end unbuffering;
  procedure sending(constant ptrAddr: std_logic_vector;
    signal ptr: std_logic_vector(31 downto 0);
    signal addr: out std_logic_vector(31 downto 0);
    signal data: inout std_logic_vector(15 downto 0);
    signal conf: in std_logic;
    signal re: out std_logic
  ) is
  begin
    wait for Tclk/10;
```

```
        addr <= ptrAddr;
        data <= ptr(31 downto 16);
        pulse(re);
        wait until conf = '1';
        wait for Tclk;
        data <= ptr(15 downto 0);
        wait for Tclk;
      end sending;
      FUNCTION incPtr (
        ptr: std_logic_vector(31 downto 0)
      ) RETURN std_logic_vector IS
        variable newptr, old: std_logic_vector(31 downto 0);
      BEGIN
        old := std_logic_vector(std_ulogic_vector(std_ulogic_vector(ptr) + "01000"));
       newptr := std_logic_vector(std_ulogic_vector(std_ulogic_vector(ptr) + "01000") rem To_Std_ulogic((size*8),
31));
        if(newptr(30 downto 0) < old(30 downto 0)) then
          newptr(31) := not(newptr(31));
        end if;
        return newptr;
      END incPtr;
      function conv2dataAddr(
        constant ptr: std_logic_vector
      ) return std_logic_vector is
      begin
        return std_logic_vector(std_ulogic_vector(ptr) + "01000");
      end conv2dataAddr;
      function convFromDataAddr(
        constant ptr: std_logic_vector
      ) return std_logic_vector is
      begin
        return std_logic_vector(std_ulogic_vector(ptr) - "01000");
      end convFromDataAddr;
    BEGIN
      initialize:
      process(ptr)
      begin
        if(mode = Entrada) then
          ptrAddr <= ReadPtrAddr;
          readPtr <= ptr;
        elsif(mode = Saida) then
          ptrAddr <= WritePtrAddr;
          writePtr <= ptr;
        end if;
      end process initialize;
      pooling:
      process
      begin
        addr1 <= ptrAddr;
        pulse(reqReadAddrRequest);
        pulse(re);
        WAIT UNTIL start'event and start = '1';
      end process pooling;
      read_addr:
      process
      begin
        wait until readAddrRequest = '1' and conf = '1';
        pulse(resReadAddrRequest);
        tmpptr(31 downto 16) <= data;
        wait for Tclk;
        tmpptr(15 downto 0) <= data;
        wait for Tclk/10;
        ptr <= convFromDataAddr(tmpptr);
        wait for Tclk;
        if (mode = Entrada) then
          if (writePtr = readPtr) then
            wait for Twait;
            pulse (start);
          else
            addr2 <= conv2dataAddr(readPtr);
            pulse(re);
            wait until conf'event and conf = '1';
            - buffering(buf, data); Sory, but buffering isn't a macro, since
            - there isn't any way to do macros under VHDL. If you know some,
            - please tell me.
            buf(63 downto 48) <= data;
            wait for Tclk;
            buf(47 downto 32) <= data;
            wait for Tclk;
            buf(31 downto 16) <= data;
            wait for Tclk;
            buf(15 downto 0) <= data;
            wait until req'event and req = '1';
```

```
        -- unbuffering(datainout, buf);
        datainout <= buf(63 downto 48);
        wait for Tclk;
        datainout <= buf(47 downto 32);
        wait for Tclk;
        datainout <= buf(31 downto 16);
        wait for Tclk;
        datainout <= buf(15 downto 0);
      --  pulse(inc);
        wait for Tclk;
        ptr <= incPtr(ptr);
        wait for Tclk;
        tmpptr3 <= conv2dataAddr(ptr);
        wait for Tclk;
        sending(ptrAddr, tmpptr3, addr3, data, conf, we);
--        pulse(start);
      end if;
    elsif (mode = Saida) then
      if(writePtr(30 downto 0) = readPtr(30 downto 0) and writePtr(31) = not(readPtr(31))) then
        wait for Twait;
        pulse (start);
      else
        addr2 <= conv2dataAddr(writePtr);
        pulse(we);
        wait until req'event and conf = '1';
        -- buffering(buf, datainout);
        buf(63 downto 48) <= data;
        wait for Tclk;
        buf(47 downto 32) <= data;
        wait for Tclk;
        buf(31 downto 16) <= data;
        wait for Tclk;
        buf(15 downto 0) <= data;
        wait until conf'event and req = '1';
        -- unbuffering(datainout, buf);
        datainout <= buf(63 downto 48);
        wait for Tclk;
        datainout <= buf(47 downto 32);
        wait for Tclk;
        datainout <= buf(31 downto 16);
        wait for Tclk;
        datainout <= buf(15 downto 0);
--        pulse(inc);
        wait for Tclk;
        ptr <= incPtr(ptr);
        wait for Tclk;
        tmpptr3 <= conv2dataAddr(ptr);
        wait for Tclk;
        sending(ptrAddr, tmpptr3, addr3, data, conf, we);
        pulse(start);
--        pulse(start);
      end if;
    end if;
  end process read'addr;
  readAddrRequest'control:
  process(reqReadAddrRequest, resReadAddrRequest)
  begin
    if(reqReadAddrRequest'event and reqReadAddrRequest = '1') then
      readAddrRequest <= '1';
    end if;
    if(resReadAddrRequest'event and resReadAddrRequest = '1') then
      readAddrRequest <= '0';
    end if;
  end process readAddrRequest'control;
--  increser:
--  process
--  begin
--    if(inc'event and inc = '1') then
--      ptr <= incPtr(ptr);
--      wait for Tclk;
--      tmpptr3 <= conv2dataAddr(ptr);
--      wait for Tclk;
--      sending(ptrAddr, tmpptr3, addr3, data, conf, we);
--    end if;
--    pulse(start);
--    wait on inc;
--  end process increser;
  addr'ctrl:
  process(addr1, addr2, addr3)
  begin
    if(addr3'event) then
      addr <= addr3;
    elsif(addr2'event) then
```

```
      addr <= addr2;
    elsif(addr1'event) then
      addr <= addr1;
    end if;
  end process addr_ctrl;
  st_ctrl:
  process(readPtr, writePtr, stEnable)
  begin
    if(stEnable = '1') then
      if (mode = Entrada) then
        if(writePtr /= readPtr) then
          st <= '1';
        else
          st <= '0';
        end if;
      elsif (mode = Saida) then
        if(writePtr(30 downto 0) = readPtr(30 downto 0) and writePtr(31) = not(readPtr(31))) then
          st <= '1';
        else
          st <= '0';
        end if;
      end if;
    end if;
  end process st_ctrl;
END functional;
```

Glossário de Siglas

AAL	*A*TM *A*daptaion *L*ayer
ACM	*A*ssociation for *C*omputing *M*achinery
ACSE	*A*ssociation *C*ontrol *S*ervice *E*lement
AE	*A*plication *E*ntity
AGV	*A*uto-*G*uide *V*ehicle
AM	*A*mplitude *M*odulada
AMI	*A*lternate-*m*ark-*i*nverted
AMPLE	*A*dvanced *M*ulti-*P*urpose *L*anguage
ANSI	*A*merican *N*ational *S*tandard *I*nstitute
APCC	*A*mbiente de *P*rogramação do *C*HILL *C*PqD
AP	*A*plication *P*rocess
APL	*Apl*icação
ARPA	*A*dvanced *R*esearch *P*rojects *A*gency
ARPANET	*ARPA Net*work
ARQ	*A*utomatic *R*epeat *Q*uest
ASCII	*A*merican *S*tandard *C*ode for *I*nterchange *I*nformation
ASE	*A*plication *S*ervice *E*lement
ASIC	*A*pplication-*s*pecific *I*ntegrated *C*ircuits
ASN.1	*A*bstract *S*intax *N*otation *1*
atm	Atraso de Tempo Médio
ATM	*A*ssyncronous *T*ransfer *M*ode
AT&T	*A*merican *T*elegraph *and T*elephone
AUI	*A*ttachment *U*nit *I*nterface
BER	*B*it *E*rro *R*ate
B	*B*yte Timing
BLM	*B*ehavioral *L*anguage *M*odel
BS	*B*ase *S*tation
CA	*Ca*lifornia
CASE	*C*ommon *A*plication *S*ervice *E*lement
CCIR	*C*omité *C*onsultative *I*nternational de *R*adio
CCITT	*C*omité *C*onsultatif *I*nternational de *T*élégraphique et *T*éléphonique

CCR	*C*ommitment, *C*oncurrency, and *R*ecovery
CD	*C*olision *D*etected
CEP	*C*ódigo de *E*ndereçamento *P*ostal
CEP	*C*onnection *E*nd *P*oint
CFMs	*C*élulas *F*lexíveis da *M*anufatura
CHILL	*C*CITT *H*igh *L*evel *L*anguage
C	*C*ontrol
CI	*C*ircuito *I*ntegrado
CIM	*C*omputer *I*ntegrated *M*anufacturing
CLNP	*C*onnection-*L*ess *N*etwork *P*rotocol
CLP	*C*ontrolador *L*ógico *P*rogramável
CMIP	*C*ommon *M*anagement *I*nformation *P*rotocol
CMISE	*C*ommon *M*anagement *I*nformation *S*ervice *E*lement
CMIS	*C*ommon *M*anagement *I*nformation *S*ervice
CMOT	*CM*IP *O*ver *T*CP/IP
CNC	*C*omando *N*umérico *C*omputadorizado
CNPQ	Centro de Desenvolvimento Científico e Tecnológico
COCF	*C*onnection *O*riented *C*onvergence *F*unction
CPqD	*C*entro de *Pesq*uisa e *D*esenvolvimento
CPU	*C*entral *P*rocess *U*nit
CRC	*C*yclic *R*edundancy *C*heck
CSMA	*C*arrier *S*ense *M*ultiple *A*ccess
CT	*C*ordless *T*elephone
CTS	*C*lear *t*o *S*end
DAD	*D*raft *Ad*dendum
DCA	*D*epartamento de Engenharia de *C*omputação e *A*utomação Industrial
DCD	*D*ata *C*arrier *D*etect
DCE	*D*ata *C*ircuit-terminating *E*quipment
DCS	*D*efined *C*ontext *S*et
DDP	*D*esign *D*ata*p*ort
DEC	*D*igital *E*quipament *C*oorporation
DECT	*D*igital *E*uropean *C*ordless *T*elephone
DENTEL	*De*partamento *N*acional de *Tel*ecomunições
DF	*D*on't *F*ragment
DFI	*D*esign *F*ile *I*nterface
DIB	*D*irectory *I*nformation *B*ase
DIS	*D*raft *I*nternational *S*tandard
DIT	*D*irectory *I*nformation *T*ree
DMD	*D*irectory *M*anagement *D*omain
DNS	*D*omain *N*ame *S*ervice
DOD	*D*epartment *o*f *D*efense
DP	*D*raft *P*roposal

DQDB	*D*istributed *Q*ueue *D*ual *B*us
DSA	*D*irectory *S*ervice *A*gent
DS	*D*irectory *S*ervice
DSR	*D*ata *S*et *R*eady
DTE	*D*ata *T*erminal *E*quipment
DT	*D*epartamento de *T*elemática
DTR	*D*ata *T*erminal *R*eady
DUA	*D*irectory *U*ser *A*gent
EBCDIC	*E*xtended *B*inary *C*oded *D*ecimal *I*nterchange *C*ode
EDA	*E*lectronic *D*esign *A*utomation
EDIF	*E*lectronic *D*esign *I*nterchange *F*ormat
EDI	*E*lectronic *D*ata *I*nterchange
EFT	*E*lectronic *F*unds *T*ransfer
EIA	*E*lectrical *I*ndustries *A*ssociation
EPROM	*E*rasable and *P*rogramable *ROM*
EUA	*E*stados *U*nidos da *A*mérica
EUTELSAT	*Eu*ropean *Tel*ecommunication *Sat*ellite Organization
FADU	*F*ile *A*ccess *D*ata *U*nits
FCC	*F*ederal *C*ommunications *C*ommission
FDDI	*F*iber *D*istributed *D*ata *I*nterface
FEE	*F*aculdade de *E*ngenharia *E*létrica
FFOL	*F*DDI *F*ollow *O*n
FIFO	*F*irst *i*n *f*irst *o*ut
FM	*F*reqüência *M*odulada
FPGA	*F*ield *P*rogramable *G*ate *A*rray
FPLMTS	*F*uture *P*ublic *L*and *M*obile *T*elecommunication *S*ystems
FSK	*F*requency *S*hift *K*ey
FTAM	*F*ile *T*ransfer, *A*ccess, and *M*anagement
FTP	*F*ile *T*ransfer *P*rotocol
Ga	Neutro do DTE
G	*G*round
GM	*G*eneral *M*otors
HDLC	*H*igh-level *D*ata *L*ink *C*ontrol
HML	*H*ardware *M*odel *L*ibrary
HRC	*H*ibrid *R*ing *C*ontrol
IA 5	*I*nternational *A*lphabet Number *5*
IAB	*I*nternet *A*ctivities *B*oard
IBM	*I*nternational *B*usiness *M*achine
ICF	*I*sochronous *C*onvergence *F*unction
ICI	*I*nterchange *C*ontrol *I*nformation
ICMP	*I*nternet *C*ontrol *M*essage *P*rotocol
IDU	*I*nterchange *D*ata *U*nit

IEC	*I*nternational *E*lectrotechnical *C*ommission
IEEE	*I*nstitute of *E*lectrical and *E*lectronics *E*ngineers
IETF	*I*nternet *E*ngineering *T*ask *F*orce
IHL	*I*nternet *H*eader *L*ength
I	*I*ndication
IME	*I*nstituto de *M*atemática e *E*statística
IMP	*I*nterface *M*essage *P*rocessor
INMARSAT	*In*ternational *Mar*itime *Sat*tellite Organization
INTELSAT	*Inte*rnational *Tele*communication *Sat*ellite Organization
IP	*I*nternet *P*rotocol
IPM	*I*nterpersonal *M*essage
IRAC	*I*nterdepartment *R*adio *A*divisory *C*ommittee
IS	*I*nternational *S*tandard
ISO	International Organization for Standardization
ITU	*I*nternational *T*elecomunication *U*nion
JTM	*J*ob *T*ransfer and *M*anagement
LAN	*L*ocal *A*rea *N*etwork
LAPB	*L*ink *A*ccess *P*rocedure *B*
LAPD	*L*ink *A*ccess *P*rocedure in *D* chanel
LCAEE	*L*aboratório de *C*omputação *A*plicada à *E*ngenharia *E*létrica
LCF-PMD	*L*ow-cost *F*iber *PMD*
L-connection.confirm	*L*ink *Connection Confirm*
L-connection.indication	*L*ink *Connection Indication*
L-connection.request	*L*ink *Connection Request*
L-data.indication	*L*ink *Data Indication*
L-data.request	*L*ink *Data Request*
L-data_connection.confirm	*L*ink *Data Connection Confirm*
L-data_connection.indication	*L*ink *Data Connection Indication*
L-data_connection.request	*L*ink *Data Connection Request*
L-disconnect.confirm	*L*ink *Disconnect Confirm*
L-disconnect.indication	*L*ink *Disconnect Indication*
L-disconnect.request	*L*ink *Disconnect Request*
LME	*L*ayer *M*anagement *E*ntity
L-PDU	*L*ink-*PDU*
LsimNet	*Lsim Net*list
MAC	*M*edium *A*ccess *C*ontrol
MAN	*M*etropolitan *A*rea *N*etwork
MAP	*M*anufacturing *A*utomation *P*rotocols
MAU	*M*edium *A*ccess *U*nit
MCF	*MAC C*onvergence *F*unction
MDI	*M*edium *D*ependent *I*nterface
MF	*M*ore *F*ragments

MGC	*M*entor *G*raphics *C*orporation
MHS	*M*essage *H*andling *S*ystem
MIB	*M*anagement *I*nformation *B*ase
MIC	*M*edium *I*nterface *C*onnector
MIT	*M*assachusetts *I*nstitute of *T*echnology
MMF	*M*ulti*m*ode *F*iber
MMS	*M*anufacturing *M*essage *S*tandard
MoCA	*Mó*dulo de *C*ontrole de *A*cesso ao Meio
MoConFiE	*Mó*dulo de *Con*trole de *Fi*la de *E*ntrada
MoConFi	*Mó*dulo de *Con*trole de *Fi*la
MoConFiS	*Mó*dulo de *Con*trole de *Fi*la de *S*aída
MoDEn	*Mó*dulo de *D*istribuição de *En*velopes
MoIB	*Mó*dulo de *I*nterface como o *B*arramento
MoTI	*Mó*dulo de *T*este e *I*niciação
MOTIS	*M*essage *O*riented *I*nterchange *S*ystems
MoTraP	*Mó*dulo de *Tra*tamento de *P*rioridades
MSC	*M*obile *S*witching *C*enter
M-SDU	*MAC SDU*
MS	*M*obile *S*tation
MTA	*M*essage *T*tranfer *A*gent
MTS	*M*obile *T*elephone *S*ystem
MTU	*M*ax *T*ransmission *U*nit
NFS	*N*etwork *F*ile *S*ystem
NJ	*N*ew *J*ersey
N-SAP	*N*etwork *SAP*
N-SDU	*N*etwork *SDU*
NT	*N*etwork *T*erminator
NTP	*N*etwork *T*ime *P*rotocol
ONU	*O*rganização das *N*ações *U*nidas
OOD	*O*bject *O*riente *D*esign
OSF	*O*pen *S*oftware *F*oundation
OSI	*O*pen *S*ystem *I*nterconnection
OSI-RM	OSI-*R*eference *M*odel
$\overline{\texttt{reset}}$	*Reset*
PABX	*P*rivate *A*utomatic *B*ranch E*x*change
PA	*P*re-*a*rbitrated
PBX	*P*rivate *B*ranch E*x*change
PCB	*P*rinted *C*ircuit *B*oard
PC	*P*ersonal *C*omputer
PCI	*P*resentation *C*ontex *I*dentifier
PCI	*P*rotocol *C*ontrol *I*nformation
PCM	*P*ulse *C*ode *M*odulation

PCN	*P*ersonal *C*ommunication *N*etwork
bps	*B*its *p*or *S*egundo
PDAU	*P*hysical *D*elivery *A*ccess *U*nit
PDU	*P*rotocol *D*ata *U*nit
PHY	*Phy*sical
PLA	*P*rogrammable *L*ogic *A*rray
PLCF	*P*hysical *L*ayer *C*onvergence *F*unction
PLD	*P*rogrammable *L*ogic *D*evice
PLP	*P*acket *L*ayer *P*rotocol
PLS	*P*hysica*l* *S*ignaling
PMA	*P*hysical *M*edia *A*ccess
PMD	*P*hysical *M*edia *D*ependent
PP	*P*rocessador *P*referencial
PR	*P*acket *R*adio
PROWAY	*Pro*cess Data High*way*
P-SAP	*P*resentation *SAP*
PSK	*P*hase *S*hift *K*ey
AC	*A*ccess *C*ontrol
add_{in}	*Add*ress *In*put
add_{out}	*Add*ress *Out*put
autologic	*AutoLogic*
boldbrowser	*Bold Browser*
$busy_{out}$	*Busy Out*put
$bypass_{in}$	*By Pass In*put
$bypass\{1,2\}_{out}$	*By Pass*{*1,2*} *Out*put
chk	*Check* Sum
cib	*C*omponent *I*nterface *B*rowser
$conf_{in}$	*Conf*irm *In*put
$conf_{out}$	*Conf*irm *Out*put
cs	*C*hip *S*elector
cvm	*C*ounter of *V*otes to *M*onitor
$dadd_{in}$	*D*estination *Add*ress *In*put
$dadd_{out}$	*D*estination *Add*ress *Out*put
da	*D*esign *A*rchitect
DA	*D*estination *A*ddress
$datain_{in}$	*Data In*put
$data_{in/out}$	*Data*
$dataout_{out}$	*Data Out*put
de_{in}	*D*enyed *E*nlace
$destin_{in}$	*Dest*ination *In*put
$destout_{out}$	*Dest*ination *Out*put
dmgr	*D*esign *M*anag*er*

$\texttt{dr}_{out}$	*D*ata *R*eady *Out*put
dve	*D*esign *V*iewpoint *E*ditor
ED	*E*nding *D*elimiter
E	*E*rror-detected Bit
enread	*E*DIF *N*etlister *Read*
enwrite	*E*DIF *N*etlister *Write*
FC	*F*rame *C*ontrol
FCS	*F*rame *C*heck *S*equence
FCS	*F*rame *C*heck *S*equence
f/e	*F*ull/*E*mpty
$\texttt{fein}_{in}$	*F*ull/*E*mpty *In*put
$\texttt{feout}_{out}$	*F*ull/*E*mpty *Out*put
FS	*F*rame *S*tatus
I	Binary One
Info	*Info*rmation
Info	*Info*rmation
info	*Info*rmation
$\texttt{info}_{in}$	*Info*rmation *In*put
$\texttt{info}_{out}$	*Info*rmation *Out*put
$\texttt{info\_phyin}_{in}$	*Info*rmation *In*put to *PHY*
$\texttt{info\_phyout}_{out}$	*Info*rmation *Out*put to *PHY*
localaddr	*Local Addr*ess
m	*M*onitor
$\texttt{mi}_{in}$	*M*onitor *I*ndicator *In*put
$\texttt{m}_{in}$	*M*onitor *In*put
$\texttt{mi}_{out}$	*M*onitor *I*ndicator *Out*put
$\texttt{m}_{out}$	*M*onitor *Out*put
$\texttt{mr}_{out}$	*M*onitor *R*equest *Out*put
nn	*N*umber of *N*odes
$\texttt{nnos}_{out}$	*N*úmero de *Nós Out*put
pri	*Pri*ority
$\texttt{priin}_{in}$	*Pri*ority *In*put
$\texttt{priout}_{out}$	*Pri*ority *Out*put
$\texttt{LLC/vozaddin}_{out}$	LLC/voz *Add*ress *In*put
$\texttt{LLC/vozconfin}_{in}$	LLC/voz *Conf*irmation *In*put
$\texttt{LLC/vozdatain}_{in/out}$	LLC/voz *Data In*put
$\texttt{LLC/vozwein}_{out}$	LLC/voz *W*rite *E*nable *In*put
qpt	*Q*uick*P*art *T*able
quickgrade	*QuickGrade* II
quickpath	*QuickPath*
quicksim	*QuickSim* II
re	*Re*sponse

$\texttt{re}_{out}$	*R*ead *E*nable *Out*put
$\texttt{rePHY}_{out}$	*R*ead *E*nable *Out*put to *Phy*sical
$\texttt{ri}_{in}$	*R*ead *I*ndicator *In*put
$\texttt{rr}_{in}$	*R*ead *R*equest *In*put
$\texttt{sadd}_{in}$	*S*ource *Add*ress *In*put
$\texttt{sadd}_{out}$	*S*ource *Add*ress *Out*put
SA	*S*ource *A*ddress
sb	*S*tart *B*it
SD	*S*tarting *D*elimiter
$\texttt{serin}_{in}$	*Ser*vice *In*put
$\texttt{serout}_{out}$	*Ser*vice *Out*put
$\texttt{si}_{out}$	*S*lot *I*ndicator *Out*put
src	*S*ource Address
$\texttt{srcin}_{in}$	*S*ource *In*put
$\texttt{srcout}_{out}$	*S*ource *Out*put
$\texttt{sr}_{out}$	*S*lot *R*eceived *Out*put
std_lib	*St*and*ard* *Lib*rary
tgt	*Tar*get Address
tid	*T*erminal *Id*entifier
vm	*V*otes to *M*onitor
$\texttt{we}_{out}$	*W*rite *E*nable *Out*put
$\texttt{wr}_{in}$	*W*rite *R*equest *In*put
PTT	*P*ost, *T*elegraph & *T*elephone
QA	*Q*ueue *A*rbitrated
QOS	*Q*uality *o*f *S*ervice
QuickCheck	*QuickCheck*
QuickFault	*QuickFault*
RALFO	*R*ede de *Á*rea *L*ocal com *F*ibra *Ó*tica
RAM	*R*andom *A*ccess *M*emory
RDN	*R*elative *D*istinguished *N*ame
RDSI-FE	*RDSI-F*aixa *E*streita
RDSI-FL	*RDSI-F*aixa *L*arga
RDSI	*R*ede *D*igital de *S*erviços *I*ntegrados
RFC	*R*equest *f*or *C*omments
R	*R*eceive
ROSE	*R*emote *O*perations *S*ervice *E*lement
RPC	*R*emote *P*rocedure *C*all
RTL	*R*egister *T*ransfer *L*evel
RTSE	*R*eliable *T*ransfer *S*ervice *E*lement
RTS	*R*equest *t*o *S*end
RXD	Receive Data
SAP	*S*ervice *A*ccess *P*oint

SASE	*S*pecific *A*plication *S*ervice *E*lement
SC	*S*ub*C*ommittee
SDCDs	*S*istemas *D*igitais de *C*ontrole *D*istribuído
SDH	*S*ynchronous *D*igital *H*ierarchy
SDL	*S*pecification and *D*escription *L*anguage
SDU	*S*ervice *D*ata *U*nit
SFD	*S*tart *F*rame *D*elimiter
SGMP	*S*imple *G*ateway *M*anager *P*rotocol
S	*S*ignal
SMF	*S*ingle-*m*ode *F*iber
SMF-PMD	*S*ingle-*m*ode *F*iber
SMT	*S*tation *M*anagement
SMTP	*S*imple *M*ail *T*ransfer *P*rotocol
SNA	*S*ystem *N*etwork *A*rchitecture
SNMP	*S*imple *N*etwork *M*anager *P*rotocol
SOP	*S*istema *O*peracional do *PP*
SP	*S*ão *P*aulo
SpiceNet	*Spice Net*list
S-SDU	*S*ession *SDU*
STM	*S*yncronous *T*ransfer *M*ode
TC	*T*echnical *C*ommittee
TCP	*T*ransmission *C*ontrol *P*rotocol
TCU	*T*runk *C*oupling *U*nit
TE	*T*erminal *E*quipament
TFTP	*T*rivial *F*ile *T*ransfer *P*rotocol
T	*T*ransport
T-PDU	*T*ransport *PDU*
TP	*T*ransport *P*rotocol
TP-PMD	*T*wisted-*p*air *PMD*
TR	*T*erminal *R*esistor
T-SAP	*T*ransport *SAP*
T-SDU	*T*ransport *SDU*
TTL	*T*ime *t*o *L*ive
TV	*T*ele*v*isão
TXD	Transmit Data
UA	*U*ser *A*gent
UNICAMP	*Uni*versidade Estadual de *Camp*inas
UDP	*U*ser *D*atagram *P*rotocol
UPN	*U*nidade de *P*rocessamento *N*umérico
USA	*U*nited *S*tates of *A*merica
USP	*U*niversidade de *S*ão *P*aulo
UTP	*U*nshielded *T*wisted *P*air

VC	*V*irtual *C*hanel
VCI	*V*irtual *C*hanel *I*dentifier
VHDL	*V*HSIC *H*ardware *D*escription *L*anguage
VHSIC	*V*ery *H*igh *S*peed *I*ntegrated *C*ircuit
VLSI	*V*ery *L*arge *S*cale *I*ntegration
V-Net	*V*erilog *Net*list
VP	*V*irtual *P*ath
VT	*V*irtual *T*erminal
WAN	*W*ide *A*rea *N*etwork
WBC	*W*ide*b*and *C*hannel
WG	*W*ork*G*roup
XDR	E*x*ternal *D*ata *R*epresentation

Referências

[AAG+90] Gorgonio Barreto Araújo, Humberto Adamatti, Iran Gonçalves, Patrícia Lobão, and Walter Furloni. Ia726 — redes de computadores em automação industrial i — ISDN — integrated service digital network — camadas de enlace e física. Technical report, UNICAMP, Cidade Univesitária Zeferino Vaz, Campinas/*São Paulo* (SP), Brasil, 1o Semestre 1990. Trabalho apresentado no curso Redes de Computadores em Automação Industrial I (IA726) da Pós-graduação da FEE, UNICAMP.

[Alf89] Roger C. Alford. *Programmable Logic Designer's Guide*. Howard W. Sams & Company, 4300 West 62nd Street — Indianapolis, Indiana 46268 *United States of America* (USA), 1st edition, 1989.

[ANS92a] ANSI. Enhanced media access control (MAC-2). ANSI X3.239-199x, ANSI, September 1992. Rev. 4.1.

[ANS92b] ANSI. Information processing systems — fiber distributed data interface (FDDI) part 5: Hybrid ring control (HRC). ANSI dpANS X3.186-199x, ANSI, ASC X3T9/87-10, May 1992. Rev. 4.2. ISO/IEC DIS 9314-5.

[ANS92c] ANSI. Information processing systems — fiber distributed data interface (FDDI) part 7: Token ring physical layer protocol (PHY-2). ANSI X3.231-199x, ANSI, June 1992. Rev. 5. ISO/IEC WD 9314-7.

[Arc86] Architecture Technology Corporation. The ethernet-type local networks report. Technical report, Architecture Technology Corporation, P.O. BOX 24344, Minneapolis, Minnesota 55424, July 1986. 2nd Edition.

[AYM93] Gorgonio B. Araújo, Akebo Yamakami, and Shusaburo Motoyama. Implementação da sub-camada MAC em uma rede com

integração de serviços utilizando circuitos integrados dedicados. In *Anais do 11o SBRC*, Campinas, May 1993. UNICAMP.

[Bap90] Luiz F. Baptistella. Rede digital de serviços integrados — RDSI, 1990.

[Bel82] John Bellamy. *Digital Telephony*. John Wiley & Sons, Chichester, 1982. 526pp.

[BR84] W. Burr and F. Ross. The fiber distributed data interface: A proposal for a standard 100 mbit/s, fiber optic token ring network. In *Proc. FOC/LAN '84*, Las Vegas/NV, Information Gatekeepers, Boston/MA, 1984.

[Bra68] P. T. Brady. A statistical analysis of on-off patterns in 16 conversations. *Bell System Techinical Journal*, January 1968.

[BSF+93] Armando Eduardo Barbieri, Antonio Nadir Dei Santi, Carlos Alberto Froes, Herculano Antonio Duarte, José dos Santos, Carlos Gunter Klemz, Nelson Takeo Sakomura, Pedro Grael Jr., and Sérgio Luiz Benjovengo. Panorama de telecomunicações faixa larga no mundo. PA SCAD, CPqD/Telebrás, Rodovia SP340 km 118,5, Campinas/SP, Brasil, April 1993. Projeto Aplicado: Sistema de Comutação de Alto Desempenho.

[Car91] M. C. C. Carneiro. Rede local de computadores multi-serviços: proposição de uma arquitetura e implementação da camada de sinalização. Master's thesis, Universidade Federal de São Carlos, 1991.

[CCI89a] CCITT. CCITT high level language (CHILL). Recommendations Z.200, CCITT, Geneva, 1989. Volume X. Fascicle X.6. Also International Standard ISO/IEC 9496.

[CCI89b] CCITT. Data communication networks: Message handling systems. Recommendations X.400–X.420, CCITT, Geneva, 1989. Volume VIII. Fascicle VIII.7.

[CCI89c] CCITT. Data communication networks: Open systems interconnection (OSI) — protocol specifications, conformance testing. Recommendations X.220–X.290, CCITT, Geneva, 1989. Volume VIII. Fascicle VIII.5.

[CCI89d] CCITT. Data communication networks: Open systems interconnection (OSI) model and notation, service definition. Recommendations X.200–X.219, CCITT, Geneva, 1989. Volume VIII. Fascicle VIII.4.

[CCI89e] CCITT. Data communication networks: Services and facilities, interfaces. Recommendation X.1–X.32, CCITT, Geneva, 1989. Volume VIII. Fascicle VIII.2.

[CCI89f] CCITT. Data communication over the telephone network. Recommendations Series V, CCITT, Geneva, 1989. Volume VIII. Fascicle VIII.1.

[CCI89g] CCITT. Integrated services digital network (ISDN) — general structure and service capabilities. Recommendations I.110–I.257, CCITT, Geneva, 1989. Volume VIII. Fascicle III.7.

[CCI89h] CCITT. Integrated services digital network (ISDN) — overall network aspects and functions, ISDN user-network interfaces. Recommendations I.310–I.470, CCITT, Geneva, 1989. Volume VIII. Fascicle III.8.

[CCI89i] CCITT. Telephone network and ISDN — quality of service, network management and trafic engineering. Recommendations E.401–E.880, CCITT, Geneva, 1989. Volume II. Fascicle II.3.

[Cer88] Vinton G. Cerf. IAB recommendations for the development of internet network management standards. RFC 1052, DDN Network Information Center, SRI International, April 1988.

[CF86] K. Caves and A. Flatman. FDDI-II: A new standard for integrated services high speed LANs. In *Proc. International Conference on Wideband Communications*, Pinner, Middlesex, England, 1986. Online Publications.

[CFSD88] J. Case, M. Fedor, M. Schoffstall, and J. Davin. A simple network management protocol (SNMP). RFC 1067, University of Tennessee at Knoxville, via ftp anonymous from ftp.nisc.sri.com, directory rfc, file rfc1067.txt, August 1988.

[Cha89] A. L. Chapin. Status of OSI standards. *Computer Communication Review*, 19(3), July 1989.

[Cla91] Martin P. Clark. *Networks and Telecommunications — Design and Operation*. John Wiley & Sons, Chichester, 1991.

[Col87] Vincente J. Coli. Introduction to programmable array logic. *Byte*, January 1987.

[Com91] Douglas E. Comer. *Internetworking with TCP/IP - Principles, Protocols, and Architecture*, volume 1. Prentice Hall, 07632 Englewood Cliffs/New Jersey (NJ) USA, 1991.

[Der91] Michel L. Dertouzoz. Communications, computers and networks. *Scientific American - Special Issue: Communications, Computers and Networks*, 265(3), September 1991.

[EIA86] EIA. Mms: Service and protocol definition. Draft 5, EIA, 1986.

[Fre87a] Robert A. Freedman. Getting started with pals. *Byte*, January 1987.

[Fre87b] Robert A. Freedman. A PAL programmer. *Byte*, January 1987.

[GM89] Paulo Roberto Guardieiro and S. Motoyama. Integrated voice and data services on a plastic optical fiber local area network. In *IEEE global telecommunication conference - GLOBECOM '89*, pages 1017–1021, Dallas/Texas - USA, 1989.

[GN86] Fernando Antonio Campos Gomide and Márcio Luiz Andrade Netto. Introdução à automação industrial informatizada, November 1986.

[Gon83] T. A. Gonsalves. Packet-voice communication on an ethernet local comp network. *Computer Communications Reviews*, 13(2), 1983. Also from Proceedings Association for Computing Machinery (ACM) SIGCOMM'83 — Symposium on Communication Architectures and Protocols.

[Gua91] Paulo Roberto Guardieiro. *Um método de controle de acesso para rede local com fibras ópticas e integração de voz e dados.* PhD thesis, UNICAMP, FEE - UNICAMP - Campinas/SP - Brasil, October 1991.

[Hal88] Fred Halsall. *Data communications, computer networks and OSI.* Electronic systems engineering series. Addison-Wesley Publishing Company, Wokingham England, 2nd edition, 1988.

[Hed88] Charles L. Hedrick. Introducion to the internet protocols. Technical report, Rutgers — The State University of New Jersey, via ftp anonymous from topaz.rutgers.edu, directory tcp-ip-docs, file tcp-ip-intro.doc, October 1988.

[HGH91] W. Hawe, R. Graham, and P. Hayden. Fiber distributed data interface overview. *Digital Technical Journal*, 3(2), 1991.

[HW93] A. Hopper and R. C. Williamson. Design and use of an integrated Cambridge ring. IEEE *Journal on Selected Area in Communication*, SAC–1, November 1993.

[IEE85a] IEEE. Local area networks — carrier sense multiple access with collision detection. ANSI/IEEE Std 802.3-1985, IEEE, 1985.

[IEE85b] IEEE. Local area networks — logical link control. ANSI/IEEE Std 802.2-1985, IEEE, 1985.

[IEE85c] IEEE. Local area networks — token-passing bus access method. ANSI/IEEE Std 802.4-1985, IEEE, 1985.

[IEE85d] IEEE. Local area networks — token ring access method. ANSI/IEEE Std 802.5-1985, IEEE, 1985.

[IEE87] IEEE. VHDL. Technical Report 1076 — 1987, IEEE, 1987.

[IEE90] IEEE. Project 802 — local & metropolitan area networks. Proposed Standard P802.6/D15, IEEE, October 1990.

[II89] ISO and IEC. Information processing systems — open systems interconnection — application layer structure. DIS 9545, ISO/IEC, September 1989. 34pp.

[Ina90] Paulo Cesar Minoru Inazumi. Aspectos de especificação e implementaçã da camada de apresentação do padrão MAP utilizando o ambiente EPOS. Master's thesis, UNICAMP, Cidade Univesitária Zeferino Vaz, Campinas/SP, Brasil, 1990.

[ISO82] ISO. Information processing systems — open systems interconection — basic reference model. ISO/DIS 7498, ISO, April 1982. Voting terminates on 22.10.1982.

[ISO85] ISO. OSI: JTM service/protocol specification. Technical Report 8831/2, ISO, 1985.

[ISO86a] ISO. Information processing systems — data communications — protocol for providing the connectionless-mode network service and provision of underlying service. Consolidated Final Text of DIS 8473, ISO, May 1986. IEC.

[ISO86b] ISO. Information processing systems — open systems interconnetcion — transport protocol specification. IS 8073, ISO, 1986. IEC.

[ISO86c] ISO. Osi: VT service/protocol specifications. Technical Report 9040/1, ISO, 1986.

[ISO87a] ISO. Information processing systems — open systems interconnection — abstract syntax notation one (ASN.1). IS 8324, ISO/IEC, 1987.

[ISO87b] ISO. Information processing systems — open systems interconnection — abstract syntax notation one (ASN.1) — draft addendum 1: Extensions to ASN.1. *D*raft *Ad*dendum (DAD) 8324/DAD 1, ISO/IEC, 1987.

[ISO87c] ISO. Information processing systems — open systems interconnection — basic connection oriented session service definition. IS 8326, ISO/IEC, August 1987.

[ISO87d] ISO. Information processing systems — open systems interconnection — basic connection oriented session service definition — addendum 1: Session symmetric synchronization for the session service. Draft Addendum 8326/DAD 1, ISO/IEC, July 1987.

[ISO87e] ISO. Information processing systems — open systems interconnection — basic connection oriented session service definition — addendum 2: Incorporation of unlimited user data. Draft Addendum 8326/DAD 2, ISO/IEC, August 1987.

[ISO87f] ISO. Information processing systems — open systems interconnection — basic connection oriented session protocol specification. IS 8327, ISO/IEC, August 1987.

[ISO87g] ISO. Information processing systems — open systems interconnection — basic connection oriented session protocol specification — addendum 1: Session symmetric synchronization for the session protocol. Draft Addendum 8327/DAD 1, ISO/IEC, July 1987.

[ISO87h] ISO. Information processing systems — open systems interconnection — basic connection oriented session protocol specification — addendum 2: Incorporation of unlimited user data. Draft Addendum 8327/DAD 2, ISO/IEC, August 1987.

[ISO87i] ISO. Information processing systems — open systems interconnection — specification of basic encoding rules for abstract syntax notation one ASN.1. IS 8325, ISO/IEC, 1987.

[ISO87j] ISO. Information processing systems — open systems interconnection — specification of basic encoding rules for abstract syntax notation one ASN.1 — draft addendum 1: Extensions to ASN.1. DAD 8325/DAD 1, ISO/IEC, 1987.

[ISO88a] ISO. Information processing systems — file transfer, access, and management. Final Text of Draft Internjational Standard 8571, ISO/IEC, April 1988.

[ISO88b] ISO. Information processing systems — open systems interconnection — connection oriented presentation service definition. Final text of Draft Internation Standard 8822, ISO/IEC, April 1988.

[ISO88c] ISO. Information processing systems — open systems interconnection — connection oriented presentation protocol. Final text of Draft Internation Standard 8823, ISO/IEC, April 1988.

[ISO88d] ISO. Information processing systems — open systems interconnection — protocol specification for the association control service element. Revised final Text of Draft Internation Standard 8650, ISO/IEC, April 1988.

[ISO88e] ISO. Information processing systems — open systems interconnection — service definition for the association control service element. Revised final Text of Draft Internation Standard 8649, ISO/IEC, April 1988.

[ISO88f] ISO. Information processing systems — open systems interconnections, management information service definition — part 2: Common management information service. DIS9595-2, ISO/IEC, December 1988.

[ISO88g] ISO. Information processing systems — open systems interconnections, management information protocol specification — part 2: Common management information protocol. DIS9596-2, ISO/IEC, December 1988.

[ISO88h] ISO. Information processing systems — text communication — motis — message handling: System and service overview. International Standard 10021-1, ISO/IEC, December 1988.

[ISO88i] ISO. Information processing systems — text communication — reliable transfer part 1: Model and service definition. Working Document for International Standard 9066-1, ISO/IEC, March 1988.

[ISO88j] ISO. Information processing systems — text communication — reliable transfer part 2: Protocol specification. Working Document for International Standard 9066-2, ISO/IEC, March 1988.

[ISO88k] ISO. Information processing systems — text communication — remote operations part 1: Model, notation and service definition. Working Document for International Standard 9072-1, ISO/IEC, March 1988.

[ISO88l] ISO. Information processing systems — text communication — remote operations part 2: Protocol specification. Working Document for International Standard 9072-2, ISO/IEC, March 1988.

[ISO88m] ISO. Information processing systems — text communication — the directory — overview of concepts. International Standard 9594-1, ISO/IEC, December 1988.

[ISO89a] ISO. Information processing systems — fiber distributed data interface (FDDI) part 2: Token ring media access control (MAC). ISO 9314-2, ISO, 1989. 67pp.

[ISO89b] ISO. Information processing systems — fiber distributed data interface (FDDI) part 1: Token ring physical layer protocol (PHY). ISO 9314-1, ISO, 1989. 31pp.

[Jai93] Raj Jain. FDDI: Current issues and future plans. *IEEE Communication Magazine*, 31(9), September 1993.

[Kro89] E. Krol. The hitchhikers guide to the internet. Technical Report RFC 1118, University of Illinois Urbana, via ftp anonymous from ftp.nisc.sri.com, directory rfc, file rfc1118.txt, September 1989.

[Kru93] Carlos Geraldo Kruger. ASIC e VHDL: Um estudo de metodologias de projeto visando reusabilidade de hardware. Master's thesis, UNICAMP, Cidade Univesitária Zeferino Vaz, Campinas/SP, Brasil, May 1993.

[Lat79] B. P. Lathi. *Sistemas de Comunicação.* Editora Guanabara Dois, Rio de Janeiro/RJ, Brasil, 1979. 401p.

[Lit93] Jonathan Littman. Commerce on the internet — the digital gold rush. *Unix World*, X(12), December 1993.

[LMT93] Ian M. Leslie, Derek R. McAuley, and David L. Tennenhouse. ATM everywhere? *IEEE network*, 7(2), March 1993.

[Lot87] M. Lottor. Domain administrators operations guide. RFC 1033, SRI International, via ftp anonymous from ccsun.unicamp.br, directory pub/rfc, file rfc1033.txt.Z, November 1987.

[Mar87] Trevor G. Marshall. PALs simplify complex circuits. *Byte*, January 1987.

[MB76] R. M. Metcalfe and D. R. Boggs. ETHERNET: Distributed packet switching for local computer networks. *Commun. Ass. Comput. Mach.*, 19, 1976.

[Men] Manuel de Jesus Mendes. Redes de computadores em automação industrial. *D*epartamento de Engenharia de *C*omputação e *A*utomação Industrial (DCA), FEE, UNICAMP, Campinas/SP, Brasil.

[Men89] Rafael S. Mendes. Controle em tempo real por computadores, 1989.

[Men90] Manuel de Jesus Mendes. Redes locais industriais, August 1990.

[Men93] Mentor Graphics Corporation, 8005 S.W. Boeckman Road, Wilsonville, Oregon 97070. *Fault Analysis User's Manual*, 1993. Software Version 8.2.

[MGC91] V8.0 an introduction to modeling in VHDL. 8005 S. W. Boeckman Road, Wilsonville, Oregon 97070, November 1991.

[MGC92] Mentor Graphics Corporation, 8005 S.W. Boeckman Road, Wilsonville, Oregon 97070. *Properties Reference Manual*, 1992.

[MGC93a] Mentor Graphics Corporation, 8005 S.W. Boeckman Road, Wilsonville, Oregon 97070. *A Guide to Design Process and Database Concepts*, 1993. Software Version 8.2.

[MGC93b] Mentor Graphics Corporation, 8005 S.W. Boeckman Road, Wilsonville, Oregon 97070. *AutoLogic User Interface Reference Manual*, 1993. Software Version 8.2_5.

[MGC93c] Mentor Graphics Corporation, 8005 S.W. Boeckman Road, Wilsonville, Oregon 97070. *Component Interface Browser User's and Reference Manual*, 1993. Software Version 8.2.

[MGC93d] Mentor Graphics Corporation, 8005 S.W. Boeckman Road, Wilsonville, Oregon 97070. *Design Architect Reference Manual*, 1993. Software Version 8.2.

[MGC93e] Mentor Graphics Corporation, 8005 S.W. Boeckman Road, Wilsonville, Oregon 97070. *Design Archtect User's Manual*, 1993. Software Version 8.2.

[MGC93f] Mentor Graphics Corporation, 8005 S.W. Boeckman Road, Wilsonville, Oregon 97070. *Design Manager User's Manual*, 1993.

[MGC93g] Mentor Graphics Corporation, 8005 S.W. Boeckman Road, Wilsonville, Oregon 97070. *Design Viewpoint User's and Reference Manual*, 1993. Software Version 8.2_5.

[MGC93h] Mentor Graphics Corporation, 8005 S.W. Boeckman Road, Wilsonville, Oregon 97070. *EDIF Netlist User's and Reference Manual*, 1993. Software Version 8.2_5.

[MGC93i] Mentor Graphics Corporation, 8005 S.W. Boeckman Road, Wilsonville, Oregon 97070. *QuickCheck User's and Reference Manual*, 1993. Software Version 8.2.

[MGC93j] Mentor Graphics Corporation, 8005 S.W. Boeckman Road, Wilsonville, Oregon 97070. *QuickPath User's and Reference Manual*, 1993. Software Version 8.2_5.

[MGC93k] Mentor Graphics Corporation, 8005 S.W. Boeckman Road, Wilsonville, Oregon 97070. *QuickSim II User's Manual*, 1993. Software Version 8.2.

[MGC93l] Mentor Graphics Corporation, 8005 S.W. Boeckman Road, Wilsonville, Oregon 97070. *Standard Component Library Data Book GEN_LIB Parts*, 1993. Software Version 8.2.

[Mic87] Sun Microsystem. XDR: External data representation standard. RFC 1014, Sun Microsystem, via ftp anonymous from ccsun.unicamp.br, directory pub/rfc, file rfc1014.txt.Z, June 1987.

[Mic88] Sun Microsystem. RPC: Remote procedure call – protocol specification – version 2. RFC 1057, Sun Microsystem, via ftp anonymous from ccsun.unicamp.br, directory pub/rfc, file rfc1057.txt.Z, June 1988.

[Moc87a] P. Mockapetris. Domain names – concepts and facilities. RFC 1034, Information Sciences Institute, via ftp anonymous from ccsun.unicamp.br, directory pub/rfc, file rfc1034.txt.Z, November 1987.

[Moc87b] P. Mockapetris. Domain names – implementation and specification. RFC 1035, Information Sciences Institute, via ftp anonymous from ccsun.unicamp.br, directory pub/rfc, file rfc1035.txt.Z, November 1987.

[Mot83] Motoyama. Relatório de progresso. Technical report, UNICAMP, 1983.

[MR88] K. McCloghrie and M. Rose. Management information base for network management of TCP/IP-based internets. RFC 1066, TWG, via ftp anonymous from ftp.nisc.sri.com, directory rfc, file rfc1066.txt, August 1988.

[Nar89] T. Narten. Internet routing. *Computer Communication Review*, 19(4), September 1989. Proceedings of the ACM SIGCOMM '89 Workshop.

[NH82] R. M. Needham and A. J. Herbert. *The Cambridge Distributed System*. Addison-Wesley Publishing Company, 1982.

[OH90] K. Ocheltree and S. Horvath. Requirements and design considerations ofr the FDDI follow-on LAN (FDDI-FO), version 2.0. Presentation to ASC X3T9.5 ad hoc working meeting on FFOL LAN. FFOL-007, X3T9.5/90-068. 5pp, May 1990.

[OMI86] D. J. Olander, G. J. McGrath, and R. K. Israel. A framework for network in System V. In *Proceedings of the 1986 Summer USENIX Conference*, pages 38–45, Atlanta, GA, USA, 1986.

[Per91] Douglas L. Perry. VHDL. McGraw-Hill, Inc., São Paulo, 1991.

[Pes91] Paulo Maurício Costa Pessoa. Implementação de uma interface de voz para rede local com fibras Óticas e integração de voz e dados. Master's thesis, UNICAMP, DT — FEE — UNICAMP — Campinas/SP, December 1991.

[Pos80] Jonathan B. Postel. User datagram protocol. RFC 768, Information Sciences Institute, via ftp anonymous from ccsun.unicamp.br, directory pub/rfc, file rfc0768.txt.Z, August 1980.

[Pos81a] Jonathan B. Postel. Assigned numbers. RFC 790, Information Sciences Institute, via ftp anonymous from ccsun.unicamp.br, directory pub/rfc, file rfc0790.txt.z, September 1981.

[Pos81b] Jonathan B. Postel. Internet control message protocol DARPA internet program protocol specification. RFC 792, Information Sciences Institute, via ftp anonymous from ccsun.unicamp.br, directory pub/rfc, file rfc0792.txt.Z, September 1981.

[Pos81c] Jonathan B. Postel. Internet protocol — darpa internet program protocol specification. RFC 791, Information Sciences Institute, via ftp anonymous from ccsun.unicamp.br, directory pub/rfc, file rfc0791.txt.Z, September 1981.

[Pos81d] Jonathan B. Postel. Transmission control protocol - DARPA internet program - protocol specification. RFC 793, Information Sciences Institute, via ftp anonymous from ccsun.unicamp.br, directory pub/rfc, file rfc0793.txt.Z, September 1981.

[Pry91] Martin de Pryckor. *Asynchronous Transfer Mode — Solution for Broadcand ISDN.* Ellis Horwood, 1991.

[RF92] F. Ross and R. Fink. Overview of FFOL — FDDI follow-on LAN. *Computer Communication*, 15(1), 1992.

[RM84] F. Ross and R. Moulton. FDDI overview — a 100 megabit per second solution. In *Proc. Wescon '84, Electron. Conventions Manage*, Las Angeles/CA, 1984. 2/1/1-2/1/9.

[RM88] Marshall T. Rose and K. McCloghrie. Structure and identification of management information for TCP/IP-based internets. RFC 1065, TWG, via ftp anonymous from ftp.nisc.sri.com, directory rfc, file rfc1065.txt, August 1988.

[Rob87] Phillip Robinson. Overview of programmable hardware. *Byte*, January 1987.

[Ros86] F. Ross. FDDI — a tutorial (fibre distributed data interface). *IEEE Communication Magazine*, 24(5), 1986.

[Ros89] F. Ross. Overview of FDDI: the fiber distributed data interface. *IEEE Journal Select Areas Communication*, 7(7), 1989.

[Ros90] Marshall T. Rose. *The Open Book: A Practical Perspective on OSI.* Prentice Hall, 07632 Englewood Cliffs/NJ USA, 1990.

[Ros91a] F. Ross. Fiber distributed data interface: An overview and update. *Fiber Optics Magazine*, July–August 1991.

[Ros91b] F. Ross. Get ready for FDDI-II. *Networking Management*, 9(8), 1991.

[RP89] Rogério Botteon Romano and José Enéas Ferreira Pó. Comparação dos conceitos da RDSI-faixa larga com os conceitos da RDSI=faixa estreita. Trabalho de ie-301 — tópicos em comunicações ii, UNICAMP, Cidade Univesitária Zeferino Vaz, Campinas/SP, Brasil, November 1989.

[RSS81] I. Richer, M. Steiner, and M. Seigoku. Office communications and the digital *P*rivate *B*ranch E*x*change (PBX). *Computer Networks*, pages 411–422, December 1981.

[Sch87] Robert W. Scheifler. X window system protocol, version 11. RFC 1013, Laboratory for Computer Science, via ftp anonymous from ccsun.unicamp.br, directory pub/rfc, file rfc1033.txt.Z, June 1987.

[SR89] Alicia Noemi Di Sarno and Nádia Lago Ribeiro. Comparação entre os protocolos X.25 usado na rede de pacotes e o protocolo da RDSI usado no canal B. Technical report, UNICAMP, Cidade Univesitária Zeferino Vaz, Campinas/SP, Brasil, November 1989. Trabalho apresentado no curso de Pós-graduação IE301 (Tópicos em Comunicações II) da FEE, UNICAMP.

[SS87] Nelso Orlando Berton Sanatori and Francisco Sukys. *Introdução à TV a Cores e ao Sistema PAL-M.* Editora Guanabara, Travessia do Ouvidor, 11, Rio de Janeiro/RJ, Brasil, *C*ódigo de *E*ndereçamento *P*ostal (CEP) 20040, 1987.

[Sta87] M. Stahl. Domain administrators guide. RFC 1032, SRI International, via ftp anonymous from ccsun.unicamp.br, directory pub/rfc, file rfc1032.txt.Z, November 1987.

[Ste90] W. Richard Stevens. Unix *Network Programming.* Prentice Hall, 07632 Englewood Cliffs/NJ USA, 1990.

[Tam89] Tamura. ??? Technical report, UNICAMP, 1989.

[Tan88] Andrew S. Tanenbaum. *Computer Networks.* Prentice Hall, 07632 Englewood Cliffs/NJ USA, 2nd edition, 1988.

[Tau84] Herbert Taub. *Circuito Digitais e Microprocessadores.* McGraw-Hill, Inc., São Paulo, 1984.

[Tel] CPqD/Telebrás, Rodovia SP340 km 118,5, Campinas/SP, Brasil. *Especificação do Barramento Global.* código PD.16.SP.EEA.0001A/CA-01-AA. Projeto Processador Preferencial.

[Tel86] CPqD/Telebrás, Rodovia SP340 km 118,5, Campinas/SP, Brasil. *Descrição Funcional de Hardware da Unidade de Rastreamento PP-RAS,* June 1986. código PD.25.PP.H3A.0002A/DF-01-AA. Projeto Processador Preferencial.

[Tel87a] CPqD/Telebrás, Rodovia SP340 km 118,5, Campinas/SP, Brasil. *Diagrama de Sinais no Tempo da Unidade de Rastreamento PP-RAS,* February 1987. código PD.25.PP.H3A.0002A/ST-01-AA. Projeto Processador Preferencial.

[Tel87b] CPqD/Telebrás, Rodovia SP340 km 118,5, Campinas/SP, Brasil. *Manual de Figuras PP-CON,* August 1987. código PD.25.PP.H3A.0008A/DF-01-AB. Projeto Processador Preferencial.

[Tel88a] CPqD/Telebrás, Rodovia SP340 km 118,5, Campinas/SP, Brasil. *Descrição de Sinais da Placa de Rastreamento — PP-RAS,* July 1988. código PD.25.PP.H3A.0002A/DS-01-AB. Projeto Processador Preferencial.

[Tel88b] CPqD/Telebrás, Rodovia SP340 km 118,5, Campinas/SP, Brasil. *Manual de Figuras — PP-UCP,* May 1988. código PD.25.PP.H3P.0001A/DF-01-AG. Projeto Processador Preferencial.

[Tel88c] CPqD/Telebrás, Rodovia SP340 km 118,5, Campinas/SP, Brasil. *Unidade Central de Processamento — Especificações e Características da UCP,* May 1988. código PD.25.PP.H3P.0001A/CA-01-AT. Projeto Processador Preferencial.

[Tel89a] CPqD/Telebrás, Rodovia SP340 km 118,5, Campinas/SP, Brasil. *Especificação e Características do Firmware da Unidade de Rastreamento — PP-RAS,* April 1989. código PD.25.PP.H3A.0002A/CA-02-AD. Projeto Processador Preferencial.

[Tel89b] CPqD/Telebrás, Rodovia SP340 km 118,5, Campinas/SP, Brasil. *Unidade Controladora de Interfaces Compatíveis — Especificações e Características do Hardware — PP-CON,* March 1989. código

PD.25.PP.H3A.0008A/CA-01-AB. Projeto Processador Preferencial.

[Tel90a] CPqD/Telebrás, Rodovia SP340 km 118,5, Campinas/SP, Brasil. *Especificação e Características Gerais do Processador Preferencial — PP-CAR*, May 1990. código PD.25.PP.EEA.0001A/CA-01-AD. Projeto Processador Preferencial.

[Tel90b] CPqD/Telebrás, Rodovia SP340 km 118,5, Campinas/SP, Brasil. *A Linguagem de Programação CHILL*, June 1990. APCC.

[Tel90c] CPqD/Telebrás, Rodovia SP340 km 118,5, Campinas/SP, Brasil. *Especificação e Características do Firmware da Unidade Controladora de Discos Rígidos e Flexíveis — PP-DIS*, February 1990. código PD.25.PP.H3D.0001A/CA-02-AD. Projeto Processador Preferencial.

[Tel90d] CPqD/Telebrás, Rodovia SP340 km 118,5, Campinas/SP, Brasil. *Especificação e Características do Hardware da Unidade Controladora de Discos Rígidos e Flexíveis — PP-DIS*, February 1990. código PD.25.PP.H3D.0001A/CA-01-AE. Projeto Processador Preferencial.

[Tel90e] CPqD/Telebrás, Rodovia SP340 km 118,5, Campinas/SP, Brasil. *Especificação e Características do Hardware da Unidade Controladora de Interfaces Seriais — PP-SER*, April 1990. código PD.25.PP.H3D.0004A/CA-01-AE. Projeto Processador Preferencial.

[Tel90f] CPqD/Telebrás, Rodovia SP340 km 118,5, Campinas/SP, Brasil. *Unidade Controladora de Interfaces Compatíveis — Especificação e Características de Firmware — PP-CON*, May 1990. código PD.25.PP.H3A.0008A/CA-02-AB. Projeto Processador Preferencial.

[TLX90] Tadao Takahashi, Hans K. E. Liesenberg, and Daniel Tabares Xavier. *Programação Orientada a Objetos.* Cursos Intermediários. VII Escola de Computação, *I*nstituto de *M*atemática e *E*statística (IME), *U*niversidade de *S*ão *P*aulo (USP), São Paulo/SP, 1990.

[TPP87] Jesús García Tomás, Juan Pa'on, and Orlando Pereda. OSI service specification: SAP and CEP modelling. *Computer Commun. Rev.*, 17(1), January 1987.

[UIPF] Maria Cristina Emiko Ussami, Luiz A. Iaderoza, Paulo Mauricio C. Pessoa, and Walter Furloni. Mhs — message handling system. Seminário preparado para o curso de Redes de Computação e Automação Industrial II da FEE, UNICAMP.

[Uss93] Maria Cristina Emiko Ussami. Implementação e análise de desempenho de um protocolo de comunicação na rede de serviços integrados RALFO. Master's thesis, UNICAMP, UNICAMP, Campinas/SP, Brasil, May 1993.

[WB89] U. Warrier and L. Besaw. The common management information services and protocol over TCP/IP (CMOT). RFC 1095, DDN Network Information Center, SRI International, April 1989.

[Wu93] Tsong-Ho Wu. Cost-effective network evolution. *IEEE Communication Magazine*, 31(9), September 1993.

[X/O89] X/Open, 07632 Englewood Cliffs/NJ USA. *X/Open Portability Guide*, 1989. Prentice Hall.

[Yac93] Michel Daoud Yacoub. *Foundations of Mobile Radio Engineering.* CRC, 2000 Corporate Blvd., n.W., Boca Raton, Florida 33431, 1993.