Arquitetura e Implementação de Interface de Linha Física para um Comutador ATM

por

Mamadou Touré

Este trabalho foi apresentado à Pós-graduação em Informática do Centro das Ciências e Tecnologia da Universidade Federal de Paraíba como requisito parcial para obtenção do grau de Mestre em Informática.

Orientador: Prof Dr. William Ferreira Giozza

Co-orientador: Prof. Hamilton Soares da Silva

Campina Grande

30 de Outubro de 1998

T727a Touré, Mamadou.
Arquitetura e implementação de interface de linha física para um comutador ATM / Mamadou Touré. - Campina Grande, 1998.
97 f.

Dissertação (Mestrado em Informática) - Universidade Federal da Paraíba, Centro de Ciências e Tecnologia, 1998.
"Orientação : Prof. Dr. William Ferreira Giozza, Prof. M.Sc. Hamilton Soares da Silva".
Referências.

1. Interface Homem-Máquina. 2. Arquitetura Flexível. 3. Comutador ATM - COMATM 4. Dissertação - Informática. I. Giozza, William Ferreira. II. Silva, Hamilton Soares da. III. Universidade Federal da Paraíba - Campina Grande (PB). IV. Título

CDU 004.5(043)

ARQUITETURA E IMPLEMENTAÇÃO DE UMA INTERFACE DE LINHA FÍSICA PARA UM COMUTADOR ATM

MAMADOU TOURÉ

DISSERTAÇÃO APROVADA EM 30.10.1998

PROF. WILLIAM FERREIRA GIOZZA , Dr.
Orientador

PROF. HAMILTON SOARES DA SILVA, M.Sc
Co-Orientador

PROF. JOÃO MARQUES DE CARVALHO, Ph.D
Examinador

PROF. JOSÉ ROBERTO DE ALMEIDA AMAZONAS, Dr.
Examinador

CAMPINA GRANDE - PB

Para minha mãe, meu pai, meu irmão Pape, minha família, Angela E.S. Kross, Kathleen M. Bond, professores William Ferreira Giozza, Hamilton Soares da Silva, e Gustavo H.B. Motta.

Agradecimentos

ALHAMDOU LILLÁHI RABIL ALAMINA.

Gostaria através deste expressar toda a minha gratidão ao meu orientador Prof William F. Giozza, pela compreensão, ajuda e principalmente por tornar este trabalho uma experiência válida;

Ao Professor Hamilton, por ser um parceiro neste trabalho e ter confiado em mim;

A Angela E. S. Kross pela compreensão e por ser uma parceira dinámica deste trabalho;

Ao meu irmão Pape pelo apoio e incentivo;

Ao Professor Gustavo H.B Motta, pelo apoio e a confiança de sempre;

A amiga de sempre Angela E.S Kross;

A Solange Bandeira;

Aos funcionários da Assessoria Internacional e ao Professor Sales da PRG

A Vera Lucia e Aninha da COPIN;

Ao Departamento de Informática da UFPB-Campus I;

Ao LCS/Escola Politécnica da USP;

À CAPES, pelo apoio financeiro, através de Bolsa de Estudos;

E finalmente a todas as pessoas que de uma maneira ou de uma outra me apoiaram nas várias etapas desta tese.

Resumo

O presente trabalho apresenta uma arquitetura flexível para a Interface de Linha Física (ILF) do Comutador ATM COMATM. A ILF é uma arquitetura flexível e modular podendo trabalhar com os padrões **SONET** (155 ou 622 Mbps) e **TAXI** (100 Mbps). A sua interface com o módulo de comutação ATM (MCOA) do COMATM é feita usando sinais do padrão UTOPIA. Apresentaremos também detalhes da organização do *hardware* e do *software* e da interligação dos blocos constituindo a ILF, concluindo com uma proposta de uma arquitetura de interface linha física genérica para comutadores ATM. Este trabalho teve início em Março de 1996 e demorou 2 anos e meio.

Abstract

This Master Degree dissertation describes the physical interface (ILF) architecture of a ATM LAN switch (COMATM). The ILF is a flexible and modular architecture which can be used with physical layer standards ***155Mbps SDH/SONET*** *or* ***TAXI*** *(100 Mbps). Its interface with the ATM Switching Module (MCOA), implementing the ATM layer functions, is based on UTOPIA standard signals. We'll point out in this trhesis the details of the hardware/software organization and the junction of the ILF block, and a proposition of a generic physical interface for ATM switches. This work started in March 1996, and lasted about two years and half.*

SUMÁRIO

1 INTRODUÇÃO 8
1.1 Motivação *11*
1.2 Objetivo *14*
1.3 Organização do Trabalho *14*
2 MODO DE TRANSFERÊNCIA ASSÍNCRONO (ASYNCHRONOUS TRANSFER MODE - ATM) 16
2.1 Introdução *16*
2.2 Estrutura de uma Rede ATM *17*
2.3 Modelo de Referência *20*
2.4 As Camadas de protocolos do ATM *21*
2.4.1 A Camada Física 22
2.4.2 A Camada ATM 22
2.4.3 Camada de Adaptação ATM (AAL) 25
2.4.4 Sinalização e controle de tráfego 26
2.5 Operação e Manutenção *27*
2.6 Resumo *28*
3 CAMADA FÍSICA DO MODELO DE REFERÊNCIA DA RDSI-FL 29
3.1 Introdução *29*
3.2 Tipos de Células *30*
3.3 Sub-Camada de Meio Físico (PM) *31*
3.4 Sub-Camada de Convergência de Transmissão (TC) *32*
3.5 Sistemas de Transmissão *32*
3.5.1 Estrutura baseada em SDH/SONET 32
3.5.2 Camada Física para Interface de 100 Mbps 40
3.6 Resumo *45*
4 INTERFACE DE LINHA FÍSICA PARA O COMATM 46
4.1 O COMATM *46*
4.2 A Interface de Linha Física (ILF) do Comutador ATM COMATM *50*
4.2.1 Módulo de Transmissão 51
4.2.2 Módulo de Recepção 58
4.2.3 Memória ROM 66
4.2.4 Memória RAM 67
4.2.5 Bloco de Teste 67
4.2.6 ISA Bus 68
4.3 Arquitetura Genérica de Interface de Linha Física para Comutadores ATM *69*
4.3.1 Arquitetura 69
4.3.2 Interface Genérica de Transmissão 71
4.3.3 Interface Genérica de Recepção 72
4.4 Resumo *72*
5 ESTRATÉGIA DE IMPLEMENTAÇÃO , TESTES E SIMULAÇÃO 74
5.1 Introdução *74*
5.2 Implementação do Driver da Placa *75*
5.3 Estratégia de implementação em Hardware *75*
5.3.1 Justificativa para Escolha da Linguagem VHDL 76
5.3.2 Justificativa para Escolha de FPGA (*Field-Programmable Gate Arrays*) 77
5.4 Simulação *79*
5.4.1 Simulação da Entrada do Bloco de Controle de Transmissão 79
5.4.2 Simulação da Saída do Módulo de Recepção 81
5.4.3 Simulação da Saída do Bloco de Controle de Recepção 85
5.5 Resultados *87*
6 CONCLUSÕES 90

Lista de Figuras

FIGURA 1.1 REDE DEDICADA PARA CADA TIPO DE SERVIÇO 9
FIGURA 1.2: PCS COM PLACAS ATM CONECTADOS A UM COMUTADOR ATM 13
FIGURA 2.1: ESTRUTURA DE UMA REDE ATM 17
FIGURA 2.2: ROTEAMENTO DE CONEXÕES VIRTUAIS DE UM COMUTADOR ATM BASEADO NA INFORMAÇÃO DE UMA TABELA DE ROTEAMENTO 19
FIGURA 2.3: MODELO DE REFERÊNCIA DOS PROTOCOLOS DA RDSI-FL 20
FIGURA 2.4: MODELO DE REFERÊNCIA DO PROTOCOLO ATM. 21
FIGURA 2.5: AS CAMADAS DO PROTOCOLO DE UMA REDE ATM. 22
FIGURA 2.6: ESTRUTURA DE UMA CÉLULA ATM NA UNI E NA NNI. 23
FIGURA 3.1: PONTOS DE REFERÊNCIA DA INTERFACE USUÁRIO-REDE (UNI). 31
FIGURA 3.2: ESTRUTURA DE UM QUADRO SONET STS-1. 33
FIGURA 3.3: HIERARQUIA DIGITAL SÍNCRONA (SDH). 34
FIGURA 3.4 TRANSMISSÃO DE CÉLULAS ATM NO STM-1 35
FIGURA 3.5: FORMATO DO QUADRO STS-3C. 39
FIGURA 3.6: CODIFICAÇÃO MLT-3. 43
FIGURA 4.1: ARQUITETURA DO COMATM. 47
FIGURA 4.2: INTERFACE UTOPIA ENTRE A ILF E O MCOA (ICF/ABSE) 50
FIGURA 4.3: DIAGRAMA EM MÓDULOS DA ILF 51
FIGURA 4.4: ARQUITETURA DA INTERFACE DA LINHA FÍSICA ILF 52
FIGURA 4.5: BLOCO CONTROLE DE TRANSMISSÃO. 53
FIGURA 4.6: BLOCO GERADOR DE CABEÇALHO. 54
FIGURA 4.7: *BUFFER* DE TRANSMISSÃO. 55
FIGURA 4.8: MUX PARA SISTEMA DE TRANSMISSÃO. 59
FIGURA 4.9: EMBARALHADOR. 57
FIGURA 4.10: MAPEADOR DE CÉLULA. 58
FIGURA 4.11: BLOCO CONTROLE DE RECEPÇÃO. 59
FIGURA 4.12: CONTROLADOR DE FLUXO DE RECEPÇÃO E *BUFFER* E RECEPÇÃO. 60
FIGURA 4.13: BLOCO VERIFICAR CABEÇALHO 62
FIGURA 4.14: DETECTAR CÉLULAS DE GERENCIAMENTO. 63
FIGURA 4.15: DESEMBARALHADOR 64
FIGURA 4.16: *SDH RECEIVER* 65
FIGURA 4.17: BLOCO DIAGNÓSTICO DE ERROS. 69
FIGURA 4.18: MEMÓRIA ROM. 66
FIGURA 4.19: MEMÓRIA RAM DA ILF. 67
FIGURA 4.20: BLOCO TESTE. 68
FIGURA 4.21: ARQUITETURA DE UMA ILF GENÉRICA 70
FIGURA 4.22: INTERFACE GENÉRICA DE TRANSMISSÃO 71
FIGURA 4.23: INTERFACE GENÉRICA DE RECEPÇÃO. 72
FIGURA 5.1: ALTERNATIVAS DE PROJETOS DE CIRCUITOS. 79
FIGURA 5.2: FLUXOGRAMA DE UM PROJETO DE CIRCUITO USANDO UMA FERRAMENTA CAD PARA FPGAS. 83
FIGURA 5.3: DIAGRAMA DE SIMULAÇÃO DO MÓDULO DE TRANSMISSÃO EM MODO DE TESTE. 84
FIGURA 5.4: DIAGRAMA DE SIMULAÇÃO DO MÓDULO DE TRANSMISSÃO EM MODO NORMAL. 85
FIGURA 5.5: DIAGRAMA DE SAÍDA DO MÓDULO DE RECEPÇÃO DE CÉLULAS VINDAS DA INTERFACE TAXI 86
FIGURA 5.6: DIAGRAMA DE SAÍDA DO MODULO DE RECEPÇÃO DE CÉLULAS VINDO DA INTERFACE TAXI (CBO = 1). 87
FIGURA 5.7: DIAGRAMA E TRANSMISSÃO DE CÉLULAS VINDO DA INTERFACE SDH. 88
FIGURA 5.8: DIAGRAMA E TRANSMISSÃO DE CÉLULAS VINDO DA INTERFACE SDH (CBO =1) 89
FIGURA 5.9: DIAGRAMA DE FLUXO DO BLOCO DE RECEPÇÃO 91
FIGURA 5.10: DIAGRAMA DE FLUXO DO BLOCO DE RECEPÇÃO.EM MODO TESTE 92
FIGURA 5.11: A INTERFACE DE LINHA FÍSICA GERADO PELO MAXPLUS II DA ALTERA. 94
FIGURA 5.12: *LAYOUT* DA PLACA DA ILF. 94

Lista de Tabelas

TABELA 2.1: VALORES DO CAMPO PT DA CÉLULA ATM. 91
TABELA 2.2: CLASSES DE SERVIÇOS DAS AALS 94
TABELA 3.1: ESTRUTURA DE CABEÇALHO DE UMA CÉLULA OCIOSA. 32
TABELA 3.2: PADRÃO DE CABEÇALHO PARA CÉLULAS DE GERENCIAMENTO DE NÍVEL FÍSICO. 33
TABELA 3.3: OCTETOS DE *OVERHEAD* DE UM QUADRO STS-3C UTILIZADO NA RDSI-FL. 40
TABELA 3.4: FUNÇÕES DE OPERAÇÃO E MANUTENÇÃO (OAM). 41
TABELA 3.5: FUNÇÕES DA CAMADA FÍSICA. 43
TABELA 3.6: CÓDIGO DE CONTROLE DE UMA ENLACE DE ACESSO ATM. 45
TABELA 5.1: ALGUNS VETORES DE SIMULAÇÃO E OS RESPECTIVOS RESULTADOS ESPERADOS. 91
TABELA 5.2: CONFIGURAÇÃO DO *CHIP* DA ILF 94

Capítulo

1

Introdução

Desde o ENIAC até hoje, os computadores e as tecnologias em volta deles conheceram uma constante e crescente evolução, tendo feito um progresso sem precedente em um tempo muito curto. Com a necessidade de compartilhar recursos, dados e informações, surgiram as primeiras redes de computadores. O modelo inicial de um computador central, servindo a todas as necessidades computacionais de uma instituição, ainda existe em uma escala menor. Este modelo, está sendo substituído por outra tecnologia onde um grande número de computadores separados, mas interconectados, executam tarefas bem distintas, compartilhando ou não recursos, dados, informações e aplicações. Esses sistemas são chamados de redes de computadores[1].

Mas, bem antes das redes de computadores, estavam presentes e operacionais as redes de telecomunicações que sofreram uma grande evolução desde os tempo de Alexander Graham Bell até os nossos dias[2]. Apesar de usar o mesmo meio físico de transmissão, tínhamos, ou melhor, ainda temos para cada tipo de serviço (telefonia, telex, comunicação de dados, Internet, etc.) uma rede dedicada conforme está ilustrado na Figura 1.1.

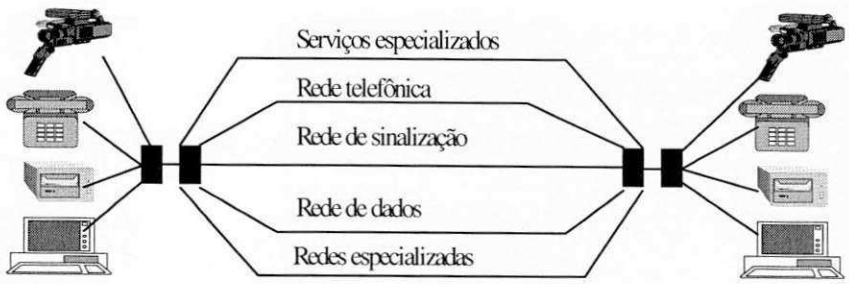


Figura 1.1: Rede Dedicada para Cada Tipo de Serviço.

Tradicionalmente, esses sistemas de comunicação foram desenvolvidos para o transporte de tipos específicos de informação. Com o grande desenvolvimento da tecnologia digital nas telecomunicações, principalmente com a digitalização da rede telefônica associada ao desejo de sinergia entre as diversas redes, surgiram as Redes Digitais de Serviços Integrados (RDSI) em junho de 1971, numa reunião do grupo de trabalho 2 do grupo de estudo do XI do CCITT[1][2]. Assim, começou a desaparecer o conceito de rede por tipo de serviço; e os diferentes tipos de informação passaram a ser processados de forma integrada.

A RDSI objetiva integrar todos os serviços de telecomunicação em uma só rede. E será ainda possível oferecer novos serviços aos usuários. Ela é dividida em 2 categorias: a RDSI Faixa Estreita (RDSI –FE) e a RDSI Faixa Larga (RDSI-FL).

A RDSI-FE consiste na integração de serviços, porém ainda é dependente de redes dedicadas para o atendimento dos mesmos. A taxa de transferência da informação, neste tipo de redes é de baixa velocidade, com acesso básico à taxa de 144 Kbps (dois canais B de 64 Kbps e um canal D de 16 Kbps), e um acesso primário com taxas correspondentes às dos canais T-1 ou E-1 (1,5 ou 2 Mbps, respectivamente), de acordo com o padrão de transmissão adotado em cada país.

A RDSI-FL fornece a infra-estrutura de transporte para uma variedade de fontes de tráfego tais como vídeo, voz, dados, etc., num ambiente integrado. As taxas de transferência, neste tipo de rede, vão, inicialmente de 155 a 622 Mbps, permitindo a

[1] Comitê Consultivo Internacional de Telegrafia e Telefonia, atualmente denominado ITU-T (*International Telecommunication Union Telecomunication Standardization Sector*)

utilização de aplicações, tais como teleconferência, videoconferência, telepresênca etc. Ela não terá apenas o acesso integrado; (como também) haverá uma única rede de transporte. A RDSI-FL suporta conexões comutadas, permanentes e semi-permanentes; podendo ser ponto-a-ponto, ponto-a-multiponto e multiponto-a-multiponto. Os serviços podem ser fornecidos sob demanda, reservados ou permanentes. Esta rede suporta serviços orientados a circuitos, ou pacotes, do tipo mono ou multimídia, e configurações unidirecionais ou bidirecionais[2].

Para poder suportar os diferentes serviços definidos pela RDSI-FL, e, ao mesmo tempo, preencher os requisitos exigidos pelos diferentes tipos de tráfego, a altas velocidades de transmissão, foi escolhida a tecnologia ATM (*Asynchronous transfer Mode*) como sendo a mais adequada [3].O modo ATM é conhecido como a tecnologia de base para a próxima geração de comunicação global [4] e é considerado um componente chave que vai tornar a visão da RDSI-FL uma realidade . Ele provê uma habilidade única para manusear, em tempo real os requisitos de tráfegos das aplicações multimídia emergentes, provendo, ao mesmo tempo, compatibilidade com as crescentes necessidades de maior largura de banda. O modo ATM está fazendo grandes avanços na área de comunicação de dados; conectando e substituindo redes já existentes [5].

Nos últimos anos constatou-se uma crescente demanda da tecnologia ATM nas redes locais e/ou corporativas, isto para prover interconexão de servidores de redes a diversas redes locais ou simplesmente a interconexão das próprias redes de uma forma quase que transparente. Esta demanda vai continuar a crescer pois está tendo ultimamente uma grande procura de serviços multimídia, principalmente os que envolvem vídeo, necessitando portanto de acessos a alta velocidade. A utilização do ATM para redes locais e públicas permitirá compatibilidade total na interconexão das redes, evitando a necessidade de se prover modos de interfuncionamento, como os utilizados para a interconexão de redes de alta velocidade como DQDB (150 Mbps) e FDDI (100 Mbps).

O modo ATM é uma tecnologia de telecomunicação definida pelas instituições de padronização ANSI[2] e ITU (ex - CCITT), para transportar uma grande variedade de tráfegos de usuários, incluindo voz, vídeo e dados em qualquer Interface de Usuário - Rede (*User-to-Network Interface UNI*)[4]. O ATM é baseado na transmissão de pequenas unidades de informações, de tamanho fixo e formato padronizado, denominadas células. Conceitualmente, as células não são muito diferentes dos pacotes utilizados nas redes de transmissão de dados, como X.25. Cada célula tem 53 octetos referentes a um cabeçalho de 5 octetos e a um campo de informação de 48 octetos. As células são transmitidas através de conexões com circuitos virtuais; seu encaminhamento se baseia na informação contida no cabeçalho de cada uma delas.

1.1 Motivação

Como qualquer tecnologia bem definida e elaborada, o modo ATM também precisa de uma ferramenta para poder implementar as funcionalidades nele contidas. Assim, o comutador ATM é o elemento físico capaz de implementar todas as funcionalidades que o modo ATM se propõe a oferecer. As funções de um comutador ATM podem ser divididas em três categorias principais: comutação básica, funções relacionadas com a comutação e funções não relacionadas com a comutação[1]. O Modo de Transferência Assíncrono (ATM) permite a concepção de redes locais com banda passante e vazão superiores às das redes locais com meio compartilhado. Além disso, o modo ATM é uma tecnologia de transporte flexível que pode integrar todos os tipos de mídia, tais como voz, dados e vídeo [2,6].

Um comutador ATM implementa todas as funcionalidades das camadas definidas no seu modelo de referência (Física, ATM). Trataremos desse assunto no Capítulo 2. Uma decisão fundamental no projeto de um comutador ATM é o sistema adotado para a interface de linha física (módulo que implementa a camada Física do modelo de referência). A primeira providência a ser tomada quanto a esse assunto é saber se a interface da camada física é voltada para a interface UNI pública ou privada;

---

[2] *American National Standardization Institute*, órgão de padronização nos Estados Unidos

só depois se pode escolher o sistema de transmissão adequado para a aplicação desse comutador. O Forum[3] ATM especificou vários tipos de sistemas de transmissão que podem ser utilizados na implementação de um comutador ATM, dentre os quais podemos citar o SDH/SONET, o DS3, o sistema tipo FDDI, o sistema a 155 Mbps sobre par trançado não blindado (UTP), etc.

O sistema de transmissão mais utilizado na implementação de interfaces de linha física para comutadores ATM tem sido o SDH (*Synchronous Digital Hierarchy*), também conhecido nos E.U.A como SONET (*Synchronous Optical NETwork*), um padrão internacional definido pelo CCITT, em 1989 (juntamente com um conjunto de recomendações: G707, G708, G709), e que permite a interoperabilidade entre UNI pública e UNI privada. O grande interesse por este padrão vem do fato dele, aproveitar a rede totalmente sincronizada, unificar os padrões europeu e americano, além de ser utilizado tanto em fibras óticas quanto em rádio, de colocar a inteligência nos multiplexadores de modo a resolver problemas de operação e de gerenciamento, e de ser compatível com canais PDH[4] existentes[2]. Com todas essas características anteriormente não encontradas nos outros sistemas de transmissão, o padrão SDH/SONET tem sido o sistema mais utilizado para implementação de interfaces de linha física para comutadores ATM.

Uma das maneiras de se conectar um comutador ATM com micrcomputadores tipo PCs é usar uma placa ATM no PC hospedeiro e conectá-lo diretamente numa das portas do comutador, conforme ilustrado na Figura 1.2.

---

[3] Organismo privado sem fins lucrativos, carregado de definir especificar e divulgar padrões no que diz respeito o modo ATM

[4] Hierarquia Digital Plesiócrona, Sistema de transmissão geralmente utilizado na rede telefônica

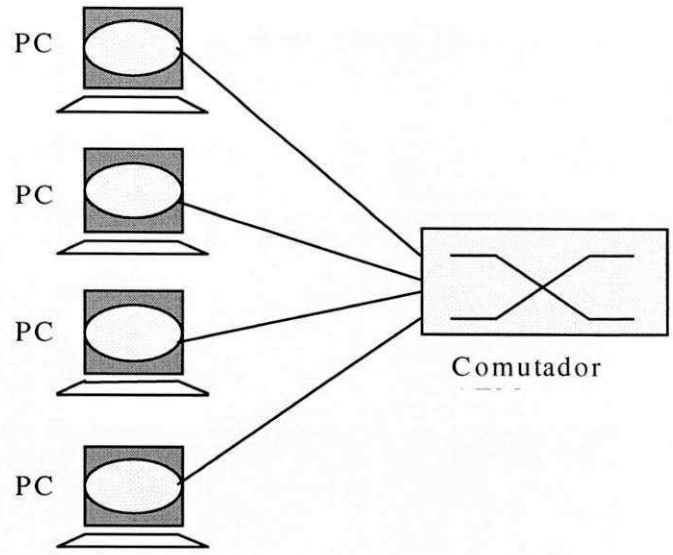


Figura 1.2:PCs com Placas ATM conectados a um comutador ATM.

Para poder utilizar tal esquema de conexão com um comutador ATM, cujo sistema de transmissão é o SDH/SONET, a estrutura física de cabeamento da rede local deverá ser feita com fibra ótica, que é o meio de transmissão utilizado pelo SDH/SONET. Mas a observação que se faz na estrutura física das redes locais é que 80% delas utilizam a tecnologia Ethernet, com par trançado[7]. Portanto, para substituir toda a infra-estrutura já existente por um cabeamento em fibra ótica e migrar para um ambiente ATM, são necessários grandes investimentos na ordem de milhões de dólares. Isso poderá afetar fortemente o êxito da adoção do modo ATM nas redes locais. Assim, utilizar uma interface de linha física baseada exclusivamente no SDH/SONET nos comutadores ATM constitui um risco que pode comprometer todo o investimento e as expectativas feitos sobre a tecnologia ATM. Cientes disso, vários fabricantes de componentes de redes, como a PMC Sierra (S3005/S3006), a AMD (TAXI Chip 7968/7969) e a TEXAS Instruments têm colocado no mercado placas, interfaces ou componentes de interface físicas (como Tranceptor) que podem ser utilizados em estrutura com par trançado, como meio físico de comunicação. Esses foram os principais motivos que nos levaram a projetar uma interface de linha física flexível, que permitirá suprir essas insuficiências. A Interface de Linha Física (ILF) que propomos é capaz de trabalhar, inicialmente, com par trançado, mas também será

possível, caso seja preciso, adaptá-la para ser usada em meios físicos que utilizam outros sistemas de transmissão, como o SDH, o FDDI, etc.

1.2 Objetivo

Neste trabalho, além de apresentarmos os conceitos, a terminologia e a organização da RDSI-FL e do modelo de referência ATM, mostraremos como foi projetado a ILF para o COMATM, detalhando a sua arquitetura completa e explicando todas as fases da sua implementação. A partir das experiências adquiridas ao longo deste trabalho, propomos um roteiro de implementação, e uma arquitetura genérica de interface de linha física que poderá ser melhorada posteriormente para, em um intervalo de tempo menor, se obter uma ILF para qualquer tipo de sistema de transmissão.

A relevância deste trabalho se encontra na flexibilidade e modularidade da arquitetura da ILF para COMATM, características inexistentes na maioria das interfaces de linha física encontradas comercialmente.

Este trabalho foi feito dentro do projeto COMATM PROTEM-II/CNPq/RHAE, envolvendo as universidades: UFPb, UFPe e USP[8]. O projeto COMATM é uma proposta de um comutador ATM utilizando como sistema hospedeiro um PC rodando o sistema operacional Linux.

1.3 Organização do Trabalho

Este trabalho dividide-se em três partes.

Na primeira parte, apresentamos o modelo de referência ATM. Trata-se de dar uma visão geral do modo ATM, da sua composição arquitetural e de como ele se integra dentro do conceito de RDSI-FL (Capítulo 2). No Capítulo 3, trataremos da Camada Física do modelo de referência da RDSI-FL.

Na segunda parte (Capítulo 4), mostramos de forma detalhada a estrutura e a arquitetura da Interface de Linha Física (ILF) para o comutador ATM (COMATM[5]) dentro do modelo de referência da RDSI/FL, e a proposta da ILF flexível para comutadores ATM. No Capítulo 5 tratamos das diferentes fases da implementação da ILF para o COMATM, as ferramentas utilizadas e os resultados obtidos.

Na terceira parte (Capítulo 6) apresentamos algumas considerações gerais sobre a dissertação, uma visão da nossa contribuição e também sugerimos alguns trabalhos futuros.

E por fim, trataremos receptivamente nos anexos A, B, C, D e E do procedimento para implementação de um *device drivers* para uma plataforma Linux, da Interface UTOPIA, das Características Funcionais do CRC, dos circuitos integrados, AM7968/AM7969 e S3005/S3006.

[5] COMATM (Comutador ATM) um projeto de um comutador ATM concebido por Giozza e Zerrouk[8]

Capítulo

2

Modo de Transferência Assíncrono (*Asynchronous Transfer Mode - ATM*)

2.1 Introdução

A RDSI, como mostramos no Capítulo 1 revolucionou de maneira substancial o conceito de redes de telecomunicações. A indústria deste setor tem investido milhões de dólares para tornar esta visão uma realidade. O modo ATM, além de ser a tecnologia básica na concretização da RDSI-FL, constitui uma excelente solução de interconexão de redes de alta velocidades em ambientes locais. O modo ATM oferece mais flexibilidade e desempenho que as tecnologias existentes, podendo integrar a transmissão de qualquer tipo de dados numa única rede.

O termo *Asynchronous* do acrônimo ATM é usado como oposição ao termo *Synchronous* da técnica STM (*Synchronous Transfer Mode*), que constitui a técnica utilizada na comutação de circuitos com a alocação de largura de banda fixa para cada um desses circuitos nas linhas físicas de comutação.

O modo ATM em oposição ao modo STM é baseado na técnica de comutação de pacotes e de multiplexacão estatística de circuitos (alocação de largura de banda conforme demanda); as informações são agrupadas em pequenas unidades de tamanho fixo com identificadores de conexão (células) cujo fluxo constitui um "circuito virtual".

2.2 Estrutura de uma Rede ATM

Uma rede ATM é constituída de comutadores ATM que são conectados entre si usando uma interface chamada de *Network Node Interface* (NNI). Os equipamentos terminais são conectados à rede usando a interface *User-Network Interface* (UNI). Se estiver dentro de uma rede de comutadores privados, a interface entre a rede pública e a privada é chamada de UNI Pública. Por outro lado, a interface de conexão entre dois comutadores privados é chamada de NNI privada (P-NNI).

Uma conexão ATM é chamada de conexão ou canal virtual (Virtual Channel VC). Têm dois tipos principais de conexão virtual: Conexão Virtual Permanente (PVC - *Permanent Virtual Connection*) e Conexão Virtual Comutada (SVC - *Switched Virtual Connection*). A Figura 2.1 mostra um exemplo de uma rede ATM onde os terminais A e C tem uma conexão virtual comutada ativa.

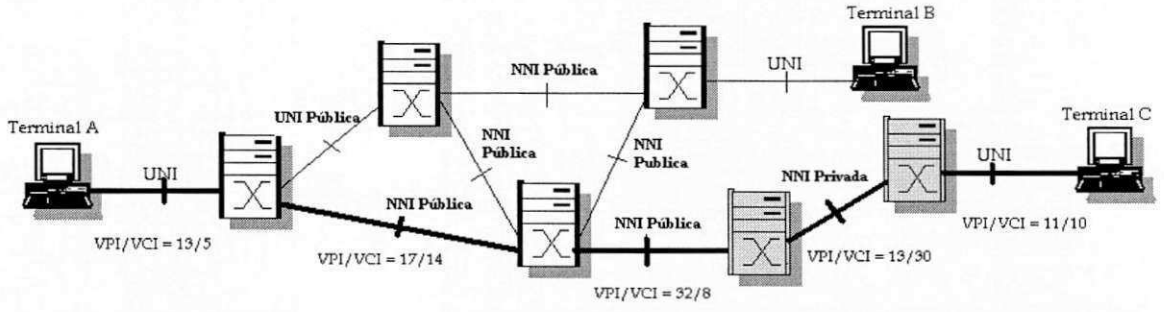


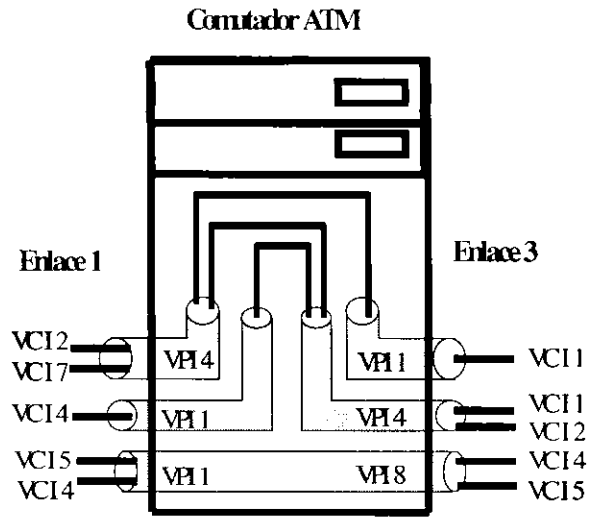
Figura 2.1: Estrutura de uma rede ATM

Em um comutador ATM, o tráfego numa conexão virtual é roteado de acordo com dois identificadores: o Identificador de Caminho Virtual (VPI - *Virtual Path Identifier*) e o Identificador de Canal Virtual (VCI - *Virtual Channel Identifier*). Além do VPI/VCI a identidade do enlace (que identifica o conjunto de VPIs em uma porta do comutador ATM) é também utilizada para identificar o VC[5]. Um VPI/VCI, é alocado para uma conexão virtual pelo comutador na sua inicialização e fica inalterado durante todo o tempo de vida da conexão. Os valores de VPI/VCI de uma única conexão são geralmente diferentes em enlaces físicos diferentes[9]. Como já colocamos na seção anterior, no ATM os dados são transportados em pacotes pequenos

chamados de células. As células carregam os identificadores de caminho e canal virtual (respectivamente VPI, VCI) no seu cabeçalho e esses identificadores são usados nos comutadores para rotear a célula. Cada comutador tem uma tabela de roteamento com as informações de todas conexões ativas trafegando no comutador. A informação de roteamento inclui novo VPI/VCI e o novo enlace físico de saída (*outgoing link*) para cada canal virtual (VC). Um caminho virtual (VP) é identificado por um VPI, e é composto de vários canais virtuais com mesma terminação. O roteamento de células pode também ser feito considerando-se somente o VPI. Neste caso, o roteamento é chamado de roteamento de caminho virtual e o valor do VCI fica inalterado. O relacionamento entre a tabela de roteamento, VPI e VCI, num comutador, é exemplificado na Figura 2.2.

Protocolos de sinalização são necessários para estabelecer e administrar conexões numa rede ATM. Os dois principais protocolos utilizados são o Q.2931[10] padronizado pelo ITU-T e a sinalização UNI do Fórum ATM[4,11]. A sinalização UNI oferece ao equipamento terminal uma linguagem que lhe permite conversar com o comutador ATM para poder estabelecer e gerenciar as conexões virtuais. A sinalização utilizada entre os comutadores de uma rede ATM é chamada de sinalização NNI e é utilizada para controle de chamadas e roteamento de conexões virtuais em uma rede ATM.

Tabela de Roteamento					
Entrada			Saída		
Enlace	VPI	VCI	Enlace	VPI	VCI
1	VPI 4	VCI 2	3	VPI 1	VCI 1
1	VPI 4	VCI 7	3	VPI 4	VCI 1
1	VPI 5	VCI	3	VPI 4	VCI 2
1	VPI 1	*	3	VPI 8	*



* Qualquer valor do VCI

Figura 2.2: Roteamento de conexões virtuais de um comutador ATM baseado na informação de uma tabela de roteamento

Quando um comutador recebe uma solicitação de inicialização de conexão, ele executa uma função de Controle de Admissão de Conexão (CAC) para definir se ele pode aceitar a conexão ou não. Se for aceita, ele encaminha a solicitação para o equipamento terminal chamado. Se a conexão não puder ser suportada considerando-se os parâmetros de tráfego e de qualidade de serviço solicitados , o comutador vai rejeitar o pedido de inicialização de conexão. O próximo comutador no caminho do equipamento terminal chamado recebe a solicitação, executa o CAC e encaminha ou rejeita a requisição de inicialização. Se o estabelecimento de conexão for rejeitado por um comutador, o comutador anterior pode tentar rotear novamente a conexão para outros comutadores. Quando a solicitação atinge o comutador de um equipamento terminal chamado, o pedido é encaminhado ao equipamento terminal chamado usando a sinalização UNI.

Quando uma conexão virtual é inicializada com descritores de tráfego específicos, estes descritores estabelecem um contrato de tráfego entre o usuário e a rede. A rede ATM vai garantir a Qualidade de Serviço (QoS – *Quality of Service*) negociada para a conexão conforme o contrato de tráfego.

É importante que nenhum equipamento terminal que esteja violando o seu contrato de tráfego possa interferir na QoS de outros terminais. É necessário garantir que um terminal não envie mais tráfego do que lhe é permitido transmitir. A função implementando este "policiamento" é chamada de Controle de Parâmetros de Uso (UPC - *Usage Parameters Control*)[4]. O Algoritmo Genérico de Taxa de Célula (GCRA - *Generic Cell Rate Algorithm*) pode ser usado para definir a conformidade no que diz respeito ao contrato de tráfego da conexão. O GCRA é algumas vezes chamado de algoritmo de estado contínuo do *Leaky bucket*[5]

2.3 Modelo de Referência

O modelo de referência dos protocolos da RDSI-FL, definido na Recomendação I.121, é composto por três planos: Plano do Usuário, Plano de Controle, e Plano de Gerenciamento; como ilustra a Figura 2.3.

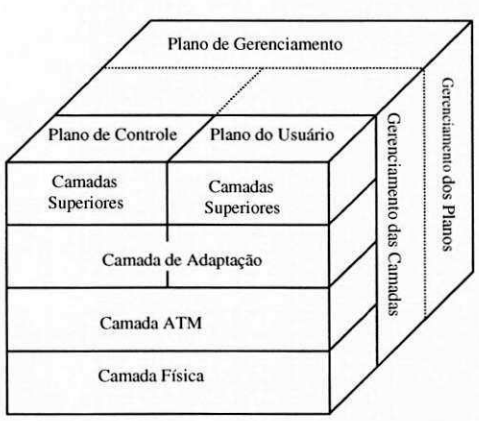


Figura 2.3: Modelo de Referência dos Protocolos da RDSI-FL

O Plano do Usuário é responsável pela transferência das informações do usuário e do controle associado. O Plano de Usuário contém as camadas Física, ATM e AAL. A camada AAL contém vários protocolos necessários para suportar diferentes serviços do usuário (por exemplo serviços CBR - *Constant Bit Rate*, VBR - *Variable Bit Rate*).

O Plano de Controle trata do estabelecimento de conexões e outras funções de controle de conexão, necessárias para oferecer serviços de comutação. A estrutura do Plano de Controle compartilha as camadas Físicas e ATM com o Plano do Usuário conforme foi mostrado na Figura 2.3. O Plano de Controle inclui também os procedimentos da camada AAL específicos para suportar a sinalização, e os protocolos das camadas superiores.

O Plano de Gerenciamento fornece funções de gerenciamento e a possibilidade de se trocar informação entre o Plano do Usuário e o Plano de Controle. O Plano de Gerenciamento tem duas seções: de gerenciamento das camadas e gerenciamento dos planos. A seção de gerenciamento das camadas trata das funções específicas de gerenciamento enquanto que o gerenciamento dos planos trata do gerenciamento e da coordenação das funções referente ao sistema completo.

2.4 As Camadas de protocolos do ATM

O modelo de referência para redes ATM apresentado anteriormente (Figura 2.3) pode também ser ilustrado na Figura 2.4.

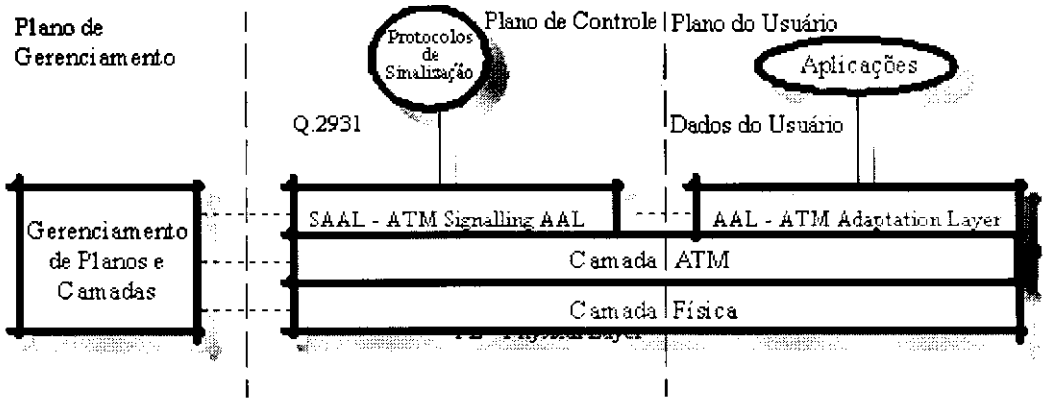


Figura 2.4: Modelo de referência do protocolo ATM.

A camada ATM transporta todas as informações em células ATM que são constituídas de 53 octetos de comprimento. Em cima da camada ATM temos a camada AAL cuja função é oferecer uma interface para as aplicações que estão em cima desta camada. O modo ATM pode ser utilizado em cima de vários protocolos e meios físicos

como fibra multimodo, monomodo, cabo coaxial e até mesmo par trançado. O Plano de Gerenciamento controla o plano de Usuário, o Plano de Controle e as camadas nele contidas. As camadas de protocolos de um plano de Usuário em nós diferentes de uma rede ATM são mostradas na Figura 2.5.

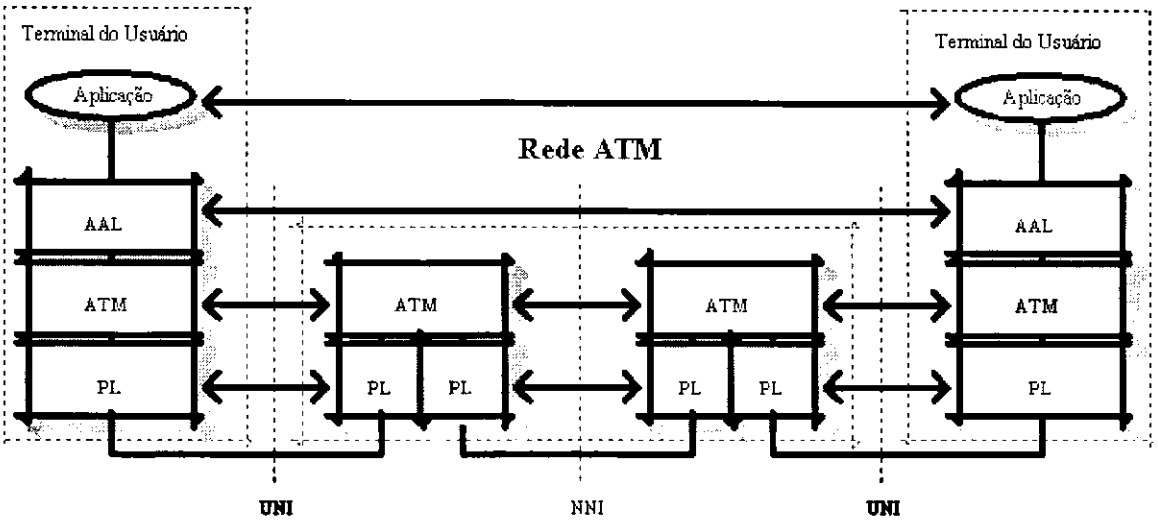


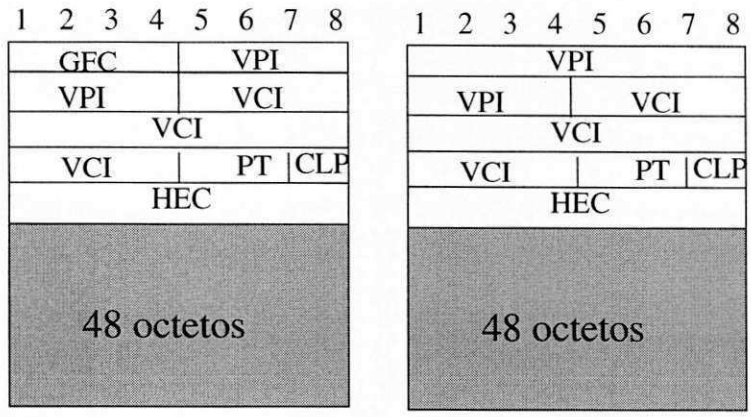
Figura 2.5: As camadas do protocolo de uma rede ATM.

2.4.1 A Camada Física

A camada Física será abordada detalhadamente no Capítulo 3.

2.4.2 A Camada ATM

A flexibilidade de uma rede ATM está principalmente ligada ao uso de pacotes de dados pequenos chamados de células ATM. Essas células são roteadas na rede na camada ATM. A célula ATM contém 53 octetos sendo os 5 primeiros, o cabeçalho. O cabeçalho é modificado em cada comutador que a célula atravessa na conexão virtual, mas a carga fica o mesmo durante todo esse percurso. A estrutura de uma célula ATM está ilustrada na Figura 2.6.



UNI **NNI**

GFC: Generic Flow Control
VPI: Virtual Path Identifier
VCI: Virtual Channel Identifier
PT: Payload Type
CLP: Cell Loss Priority
UNI: User to Network Interface
NNI: Network to Network Interface

Figura 2.6: Estrutura de uma célula ATM na UNI e na NNI.

A estrutura de uma célula ATM é semelhante em diferentes nós da rede com uma exceção. A única diferença é a utilização dos 4 primeiros bits do cabeçalho; na UNI os bits são utilizados para o campo GFC (*Generic Flow Control*) enquanto na NNI eles são utilizados para estender o espaço de numeração do VPI. O campo GFC é definido para uso de mecanismo de controle de fluxo genérico; a sua utilização não foi ainda especificada. O VPI pode ser 8 ou 12 bits dependendo da interface usada, enquanto que o VCI tem sempre um tamanho de 16 bits. Um total de 256 * 65356 ou 4096 * 65356 conexões virtuais podem ser identificadas utilizando estes dois identificadores. O campo VPI em conjunto com o campo VCI formam o rótulo da conexão utilizado pelos comutadores para encaminhar as células ao destino. Algumas combinações são utilizadas para propósitos específicos, por exemplo um VPI/VCI igual a 0/5 identifica uma conexão de sinalização UNI ponto-a-ponto. Os 3 bits do campo PT indicam o tipo de informação carregado na célula. O conteúdo do PT é chamado de identificador do tipo de carga (PTI - Carga *Type Identifier*) e seus valores são mostrados na Tabela 2.1. O terceiro bit do campo PT indica quando a célula é

transporta dados (0) ou quando ela é célula de gerenciamento (1). Em células de dados, o segundo bit do campo PT indica se houve congestionamento (1) ou não (0). O primeiro bit do campo PT na célula de dados, é transportado de forma transparente através a rede ATM; este bit é usado, por exemplo, pelo protocolo AAL5.

PTI	Interpretação
000	Célula de Dado do Usuário, sem congestionamento. Indicação Usuário-ATM-para-Usuário-ATM = 0
001	Célula de Dado do Usuário, sem congestionamento. Indicação Usuário-ATM-para-Usuário-ATM = 1
010	Célula de Dado do Usuário, com congestionamento. Indicação Usuário-ATM-para-Usuário-ATM = 0
011	Célula de Dado do Usuário, com congestionamento. Indicação Usuário-ATM-para-Usuário-ATM = 1
100	Célula de gerenciamento (OAM) tipo F5 no nível de seguimento
101	Célula de gerenciamento (OAM) tipo F5 fim-a-fim
110	Célula de gerenciamento de recurso (RM - *Resource Management cell*)
111	Reservado

Tabela 2.1: Valores do Campo PT da Célula ATM.

O bit CLP (*Cell Loss Priority*) é usado para indicar quais células deverão ser descartadas primeiro se a rede da ficar congestionada (células com CLP=1 são descartadas primeiro). Quando uma conexão ATM for inicializada, um contrato de tráfego fica estabelecido entre a rede e o usuário. O contrato é policiado para dar um alerta quando o tráfego tiver atingido o limite do contrato ou não. Se uma violação for observada, a célula poderá ser descartada ou o seu bit CLP será marcado com 1.

O campo HEC (*Header Error Control*) é utilizado para detectar erros no cabeçalho. O HEC precisa ser "recalculado" em cada enlace físico porque o VPI/VCI é diferente para cada. Não tem controle de erro para a carga da célula na camada ATM. A implementação do controle de erros HEC será detalhada no Anexo C.

2.4.3 Camada de Adaptação ATM (AAL)

Para tornar uma rede ATM prática, é necessário adaptar as características da rede interna às dos vários tipos de tráfegos que vão usar a rede. Este é o objetivo da camada de Adaptação ao ATM (AAL)[12].

A camada AAL é dividida em duas sub-camadas: a sub-camada de segmentação e remontagem (SAR - *Segmentation And Reassembly*) e a sub-camada de Convergência (CS - *Convergence Sublayer*). A subcamada SAR é responsável pela segmentação do fluxo de informação em fragmentos que podem ser acomodados no campo de informação da célula ATM, e pela remontagem desse fluxo a partir das células recebidas. Dependendo do tipo de serviço, a subcamada CS é responsável por efetuar tarefas de multiplexação de serviços, detecção de perdas de células e recuperação da relação temporal da informação original no destino. A AAL não trata de recuperação de erro; essa função é da responsabilidade das camada superiores.

Tem-se atualmente quatro tipos diferentes de protocolos padronizados para a AAL, cada um suportando um certo tipo de serviço. Um protocolo AAL especial é usada para sinalização, o SAAL.

Cada tipo de protocolo AAL está associado a uma classe de serviços definida pelo ITU-T. São quatro tipos de classes genéricas de tráfego de rede que precisam ser tratadas de modo diferente por uma rede ATM. Essas classes são designadas de Classe A até Classe D e são ilustradas na Tabela 2.2[4].

A seguir temos uma breve descrição da utilização de cada classe.

Classe A: esta classe é utilzada para emulação de circuitos. Aplicações que necessitam de serviços isócronos utilizam desse tipo de serviços, como transmissão de voz e vídeo a taxas constantes (sem compactação ou compressão).

Classe B: esta classe é destinada para tráfego de voz e vídeo cujas reproduções são feitas a taxa constante, mas que podem ser codificadas com taxas varáveis através da compactação ou compressão.

Classe C: esta classe trata de serviços encontrados em redes de comutação de pacotes com conexão como X.25, por exemplo. São serviços não isócrono orientados a conexão, onde a variação estática do retardo não causa maiores problemas.

Classe D: esta classe é destinada a serviços sem conexão e com taxa variável. Correspondem aos serviços sem conexão das redes de dados, como os de interconexão de redes com TCP/IP.

Classe X	Classe A	Classe B	Classe C	Classe D
Controle	CBR	VBR	Orientado a Conexão	Não Orientado a Conexão
Outros	Emulação de Circuitos	Voz, Vídeo, Multimídia	Dados	Dados
AAL0 (Nulo)	AAL 1	AAL 2	AAL 5	AAL3/4

Tabela 2.2: Classes de Serviços das AALs.

2.4.4 Sinalização e controle de tráfego

As funções de sinalização referentes ao controle de conexões são as seguintes:

a) Permitir o estabelecimento e a liberação de chamada/conexões entre diversos parceiros;

b) Permitir a negociação de qualidade de serviços de uma conexão;

c) Dar suporte à adição e remoção de conexão em uma chamada;

d) Dar suporte à adição e remoção de participantes em uma chamada multiponto.

Com recursos finitos (em termos de parâmetros de qualidade de serviço, como vazão e variação de atraso), a rede ATM precisa de um mecanismo de controle de tráfego para evitar congestionamento e assegurar a qualidade de serviço estabelecida para uma conexão. Esse esquema de controle de tráfego é baseado em controle de

admissão de conexões (CAC), controle de parâmetros de uso (UPC), controle de prioridades (CLP) e controle de congestionamento. Este controle de tráfego tem por objetivo evitar a degradação do fluxo de tráfego de um sistema devido à excessiva solicitação de parte dos seus recursos.

2.5 Operação e Manutenção

Para prover um bom funcionamento da rede ATM, o ITU-T na sua recomendação I-610[4] definiu as funções de operação e manutenção das camadas Física e ATM. O Forum ATM especifica cinco fases para estas funções[4]:

e) Monitoramento de desempenho - As entidades são monitoradas através da verificação contínua ou periódica das suas funções. Informações de eventos de manutenção são obtidas como resultado desta operação. Esses resultados mostram o estado das entidades monitoradas como a taxa de erros em bits, por exemplo.

f) Detecção de falhas e defeitos - Esta operação detecta qualquer mau funcionamento através de uma verificação contínua ou periódica. Como resultado são produzidas informações de manutenção e podem ser disparados diversos alarmes.

g) Proteção do sistema - O efeito causado pela falha de uma entidade que está sendo gerenciada pode ser minimizado através do bloqueio da entidade em falha ou a transferência das suas funções para outras entidades alternativas. A entidade em falha é excluída da rede.

h) Informação de falha ou desempenho- A informação sobre falhas e avisos é propagada a outras entidades de gerenciamento. Como resultado, são enviadas indicações de alarmes para outros planos de gerenciamento; assim como respostas a pedidos de relatórios de estado.

i) Localização de falhas - Falhas são localizadas através de sistemas de testes de uma entidade que tenha falhado para determinar a exata localização da falha, caso em que a informação de falha não tenha sido suficiente.

Vale ressaltar que algumas destas fases não se encontram descritas na Recomendação I.610 do ITU-T.

2.6 Resumo

Neste presente Capítulo demos uma visão geral da tecnologia ATM. Com isso abordamos as principais características desta tecnologia, para que o leitor possa entender o funcionamento básico de uma rede ATM.

A tecnologia ATM é baseada em células; um conjunto de 53 octetos dividido em 5 octetos que constituem o cabeçalho e os 48 restantes que compoem a carga da célula. As células ATM são roteadas em um comutador de acordo com uma tabela de roteamento do comutador e dos identificadores de conexão (VPI/VCI) na célula. As células ATM só ficam completas quando chegam na camada Física, a qual, a partir das informações contidas no cabeçalho da célula (os 4 octetos) gera o quinto octeto que representa o HEC. Estas células são transmitidas no meio físico para um usuário (por exemplo computador com uma placa ATM) ou para um outro comutador usando um determinado tipo de transmissão como SDH/SONET, FDDI, etc. Vários sistemas de transmissão já foram definidos pelo Forum ATM e no Capítulo seguinte trataremos de alguns deles.

Capítulo

3

Camada Física do Modelo de Referência da RDSI-FL

3.1 Introdução

Vários tipos de padrões de protocolos para a camada Física podem ser utilizados com o modo ATM. O mais comum é o SDH/SONET[1]. Estes dois padrões, um desenvolvido pelo ITU-T, o outro pela ANSI nos Estados Unidos, respectivamente são quase iguais, pois o SDH é baseado no SONET Americano. Ambos padrões podem utilizar fibra multimodo ou monomodo como meio de transmissão. Podem também utilizar cabo coaxial e par trançado, mas neste caso a taxa de transferência fica menor do que no caso da fibra.

Há somente uma diferença arquitetural entre a camada Física do ATM e a camada física do modelo OSI (*Open System Ineterconection*) da ISO (*International Standard Organização*). A camada Física do ATM é composta de duas sub-camadas: de Meio Físico (PM - *Physical Medium*), e de Convergência de Transmissão (TC - *Transmission Convergence*). A subcamada PM é quase igual à camada Física do modelo OSI; ela trata da transmissão dos bits no meio físico. A subcamada TC é mais dependente do modelo e é responsável pelas funções necessárias para suportar o tipo de célula ATM na transferência de dados. Estas funções são:

delimitação de célula, que prepara o fluxo de células para permitir que no lado receptor recupere as fronteiras da célula. Os mecanismos que definem como é feita esta tarefa são descritos na recomendação I.432 do ITU-T[13];

controle de erro do cabeçalho: no quinto octeto do cabeçalho da célula encontra-se um campo denominado HEC (*Header Error Control*), que é um campo de redundância para detecção de erro no cabeçalho. A sub-camada TC fica responsável pelo cálculo e inserção do HEC no cabeçalho da célula, bem como pela sua verificação no lado receptor;

inserção de células ociosas nos períodos de silêncio do meio: para permitir a sincronização na sub-camada PM;

desacoplamento da taxa de transmissão em relação a taxa de geração de células: isto é feito através da inserção de células especiais (ociosas) na transmissão, que deverão ser descartadas na recepção, de forma a ajustar o fluxo de transmissão à taxa de transmissão utilizada no meio.

3.2 Tipos de Células

A célula ATM é independente da camada que a gera ou que a recebe, ela é uniforme na sua estrutura. No entanto, células de "nível físico" podem ser geradas na camada Física durante a transmissão e não serem transmitidas para as camadas superiores; na recepção, este tipo de célula é simplesmente descartado. Dessas células de "nível físico" podemos distinguir:

j) Células ociosas: que são inseridas e extraídas pela camada Física para poder adaptar a taxa de transmissão de células, na interface com a camada ATM, à capacidade disponível de acordo com o sistema de transmissão específico que está sendo utilizado. As células ociosas não causam nenhum efeito no nó receptor, à exceção da delimitação de células e da verificação de cabeçalho. As células ociosas são identificadas através do cabeçalho cuja estrutura é mostrada na Tabela 3.1.

	Octeto 1	Octeto 2	Octeto 4	Octeto 4	Octeto 5
Padrão do Cabeçalho	00000000	00000000	00000000	00000001	código válido do HEC= 01010010

Tabela 3.1: Estrutura de Cabeçalho de uma Célula Ociosa.

k) Células de gerenciamento tipo F1, F2, F3. Estas células executam as funções de Operação e Manutenção (OAM – *Operation, Administration and Maintenance*). A execução destas funções geram fluxos correspondentes de informações bidirecionais denominados de fluxo F1, F2, F3 dependendo do formato do sistema de transmissão assim como das funções de supervisão contidas no TR1 (Terminador de Rede 1) e no TR2, para seção que cruza o ponto de referência $T_{fl}[2]$ conforme ilustrado na Figura 3.1.

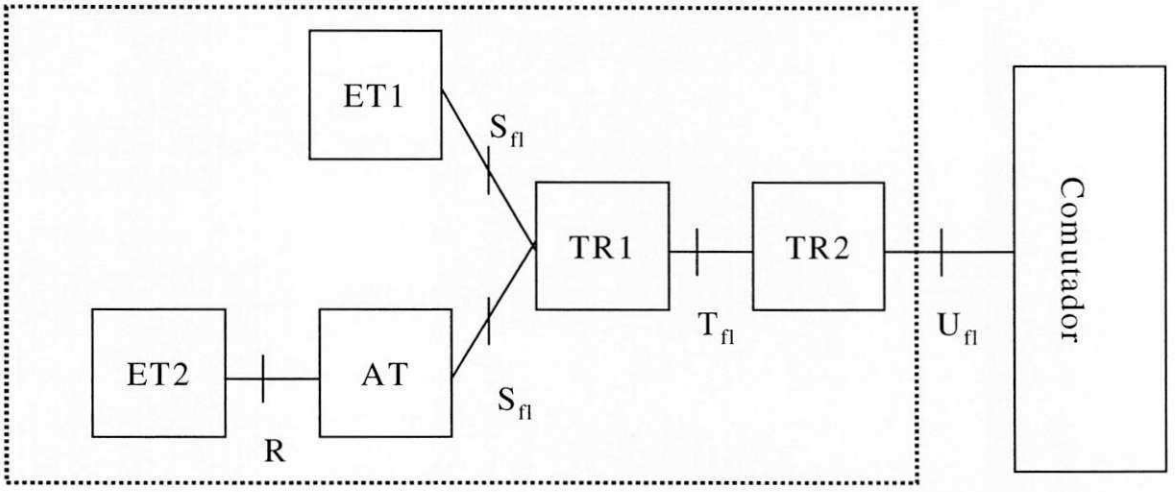


Figura 3.1: Pontos de Referência da Interface usuário-rede (UNI).

A Tabela 3.2 mostra o formato do cabeçalho das células de gerenciamento F1 e F3 (Aas funções do fluxo F2 são suportados pelo fluxo de OAM F3) .

Fluxo	Octeto 1	Octeto 2	Octeto 3	Octeto 4	Octeto 5
F1	00000000	00000000	00000000	00000011	HEC=01011100
F3	00000000	00000000	00000000	00001001	HEC=01101010

Tabela 3.2: Padrão de Cabeçalho para Células de Gerenciamento de Nível Físico.

3.3 Sub-Camada de Meio Físico (PM)

Esta sub-camada trata da transmissão do conjunto de bits através de um Meio Físico de comunicação, fazendo, por exemplo no caso de fibra óptica, as conversões eletro-ópticas. Salvo algumas exceções, as informações que trafegam neste meio são codificadas para garantir uma certa frequência nas transições do sinal enviado, de

forma a facilitar a sincronização entre as partes. Ela também é responsável pela codificação do fluxo de bits recebido da subcamada TC numa forma adequada para o meio de transmissão, para que seja possível uma sincronização entre transmissor e receptor.

3.4 Sub-Camada de Convergência de Transmissão (TC)

A sub-camada TC trata dos aspectos da camada Física que são independentes das características do meio de transmissão. Ela é responsável pela geração/verificação do HEC, geração/recuperação e adaptação de quadros de transmissão, embaralhamento/desembaralhamento da carga da célula, delimitação das células e desassociação da taxa de célula. A implementação destas funções depende do tipo de sistema de transmissão adotado.

3.5 Sistemas de Transmissão

A recomendação do ITU-T I.361 define duas maneiras de transportar as células ATM: dentro de um fluxo contínuo de células (*cell-based*), ou dentro de um quadro tipo SDH. Vários sistemas de transmissão podem ser utilizados para implementar as funcionalidades da camada Física, dentre os quais podemos citar alguns que foram definidos pelo Forum ATM como SDH/SONET, FDDI, DS3, etc[13]. Trataremos a seguir, a título ilustrativo, de duas destas estruturas envolvidas na implementação da interface de linha física, propósito deste trabalho.

3.5.1 Estrutura baseada em SDH/SONET

A Hierarquia Digital Síncrona (SDH - *Synchronous Digital Hierarchy*)[14] nasceu da definição do padrão SONET (*Synchronous Optical NETwork*)[15] que é um padrão americano utilizado por empresas de telefonia de redes ópticas.

A estrutura básica do SONET é um quadro de 810 octetos, transmitido a cada 125 μsec. Isso permite que um único octeto dentro de um quadro seja um canal de voz digital de 64 Kbps. Como o quadro mínimo é de 810 octetos, a taxa mínima na qual o SONET vai operar é de:

(810 octetos x 8000 quadros/sec) * 8 (bits) = 51,84 Mbps

O quadro básico é chamado de Sinal de Transporte Síncrono de grau 1 (STS-1 *Synchronous Transport Level 1*) e é representado como tendo 9 linhas e 90 colunas como mostrado na Figura 3.2.

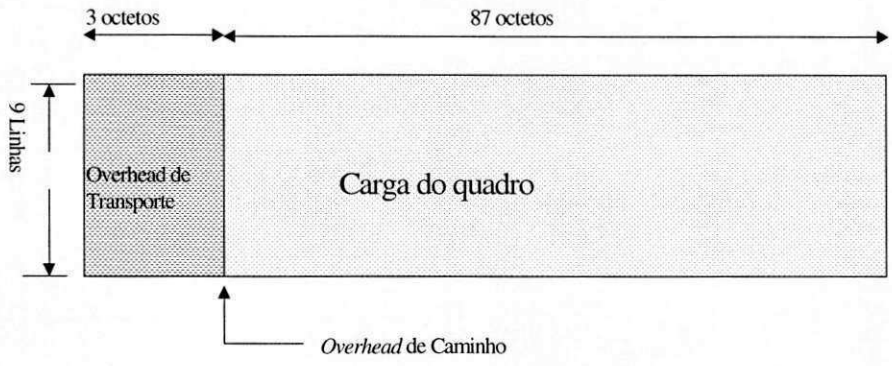


Figura 3.2: Estrutura de um quadro SONET STS-1.

Níveis hierárquicos de mais alta ordem podem ser obtidos multiplexando-se vários quadros STS-1. Uma tal operação resultará na obtenção de níveis hierárquicos STS-2, STS-3, ..., STS-N de acordo com o número de quadros STS-1 multiplexados.

Em vez de multiplexar os quadros STS-1, poder-se-ia fazer uma concatenação desses últimos mantendo os octetos de *overhead* e de carga separados, obtendo-se por exemplo, com três quadros STS-1 um quadro STS-3c. O quadro STS-3c é considerado como o quadro básico definido pelo ITU-T para sistemas SDH: onde foi renomeado para STM-1 (*Synchronous Transport Module Level 1).* O quadro STS-3c ou STM1 serve de base para definir as hierarquias STM-4, STM-16, etc. Com a definição do quadro de base como sendo o STM-1, o ITU-T conseguiu: unificar nesse padrão as taxas definidas no SONET (padrão americano) e no SDH (padrão internacional), aproveitar

a rede totalmente sincronizada, tornar possível a utilização do SDH tanto em fibras ópticas quanto em rádio, etc. A estrutura do SDH está ilustrada na Figura 3.3

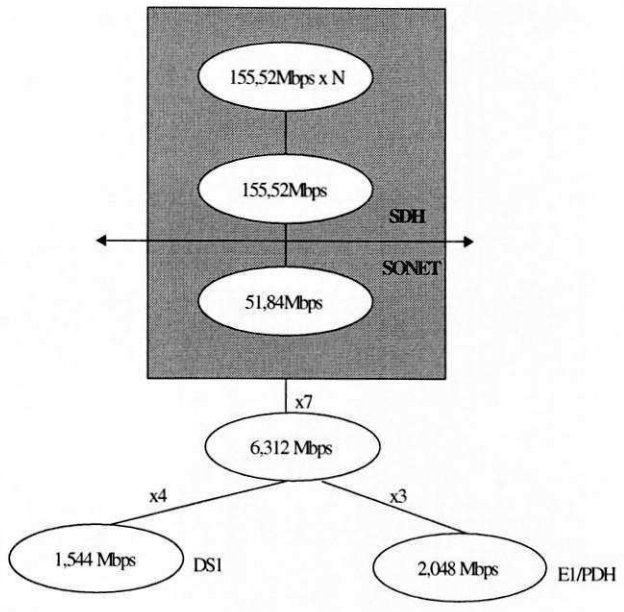


Figura 3.3: Hierarquia Digital Síncrona (SDH).

O quadro STS-3c permite taxas de transferência de 155,52 Mbps em interfaces UNI ou NNI, para redes públicas ou privadas. O sistema de transmissão SONET STS-3c provê uma estrutura de quadros para o transporte de células ATM. Estes quadros contem também octetos de *overhead* que podem transportar as informações de Gerência e Manutenção (OAM) de nível Físico. As funções da camada Física encontram-se nas sub-camadas PMD (*Physical Medium Dependent*) e TC (*Transmission Convergence*).

A Figura 3.4 ilustra um fluxo de células ATM sendo transmitido em um quadro STM-1. As células são concatenadas da esquerda para a direita, de cima para baixo. As células ATM são transmitidas na carga de um *container* virtual VC-4, e, embora sejam alinhadas com o VC-4 a nível de octetos, elas podem ultrapassar os limites de um quadro. O delineamento (determinação das fronteiras) das células dentro dos *containers* é feito através do campo HEC.

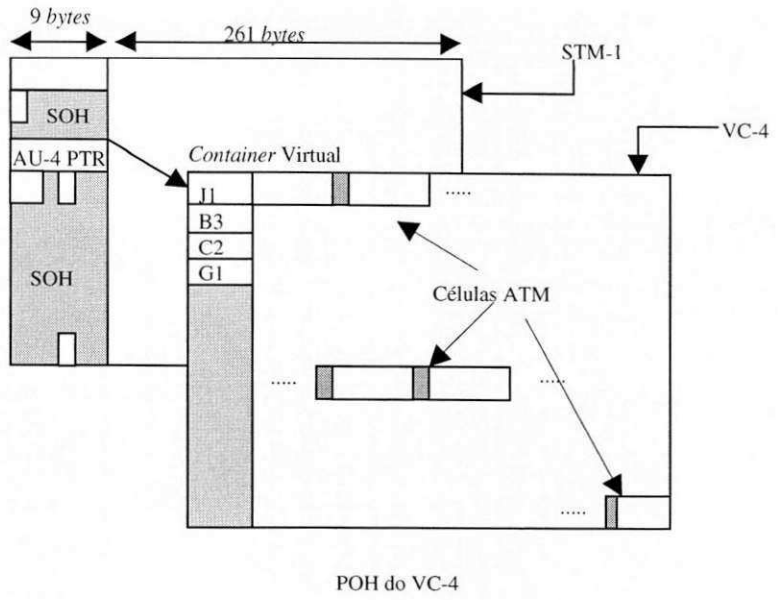


Figura 3.4: Transmissão de células ATM no STM-1.

3.5.1.1 Características da Subcamada PMD para Interfaces SDH a 155 Mbps (STM-1)

A subcamada PMD trata das características físicas e procedimentos dependentes do tipo de meio de transmissão utilizado (como codificação de linha, temporização de bit). As características físicas da UNI a 155 Mbps nos seus pontos de referência são definidas no ANSI T1E1.2/94-002R1 (e.g., OC-3 SMF, OC-3 MMF)[4].

3.5.1.2 Especificações da subcamada TC

A subcamada TC trata dos aspectos físicos que são independentes das características do meio de transmissão. A maioria das funções contidas nesta subcamada está envolvida na geração e processamento de algum tipo de octetos de *overhead* contido no quadro SDH STM-1(SONET STS-3c). A subcamada TC contém todas as funções necessárias para adaptar o serviço oferecido pela camada Física do STM-1 ao serviço requerido pela camada ATM. A subcamada TC é responsável também pelas seguintes funções:

l) Geração e Verificação de HEC: o cabeçalho inteiro (incluindo o HEC) é protegido pela seqüência do HEC (*Header Error Control*). O código do HEC está contido no último octeto do cabeçalho da célula ATM. Com este campo é possível corrigir erros de um bit e detectar erro em múltiplos bits. No lado do transmissor, o HEC é calculado a partir de um polinômio gerador e no lado do receptor, dois modos de operação são definidos: modo de correção e modo de detecção. No modo de correção, somente erros de um bit podem ser corrigidos enquanto que o modo de detecção permite detectar erros de múltiplos bits. No modo de detecção todas as células com um cabeçalho corrompido serão descartadas. Vamos tratar mais em detalhes no Capítulo 4 como são feitas as operações de geração/verificação do HEC

m) Embaralhamento/Desembaralhamento: essa operação permite a geração pseudo-aleatória da carga da célula para evitar uma seqüência contínua de bits igual à do cabeçalho e não perturbar portanto o algoritmo de delimitação das células. O polinômio $x^{43} + 1$ foi escolhido para interfaces baseadas em SDH. Este polinomial foi escolhido para minimizar a multiplicação de erros introduzida pelo embaralhador (evitando a multiplicação de erros no cabeçalho da célula).

n) Mapeamento das células nos quadros de transmissão SDH: este mapeamento é feito alinhando-se linha por linha, a estrutura de octeto de cada célula com a estrutura de octetos da capacidade do SDH STM-1 /SONET STS-3C SPE (*Synchronous* Carga *Envelope*). A capacidade total de carga do SDH STM-1 é preenchida com células, dando assim uma capacidade de transferência para as células ATM de 149,760 Mbps. A carga do quadro SDH STM-1 não é um inteiro múltiplo do tamanho da célula ATM, portanto uma célula pode cruzar as fronteira do SPE.

o) Delimitação das células: esta função como já vimos nas seções anteriores permite a identificação das fronteiras das células na carga. O formato do quadro SDH/SONET é formado de nove octetos de *overhead* seguido pela carga do quadro (SPE). Este padrão de 9 octetos de *overhead* e de 261 octetos de SPE é

repetido nove vezes em cada quadro. O quadro e as fronteira dos octetos são detectados utilizando-se os octetos A1 e A2 presentes no cabeçalho, conforme ilustrado na Figura 3.5.

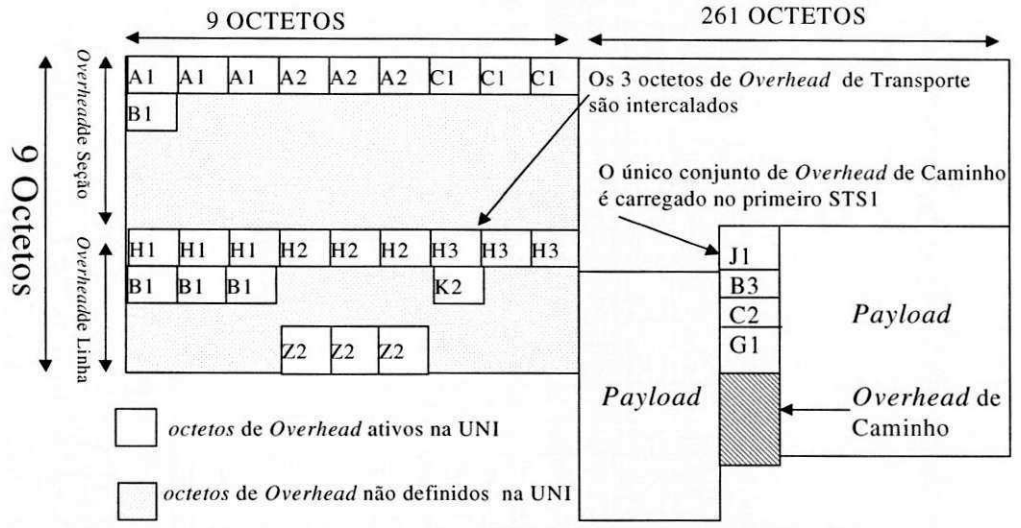


Figura 3.5: Formato do quadro STM-1 ou STS-3C.

A descrição dos octetos de *overhead* de um quadro STS-3c utilizado na RDSI-FL 155 Mbps para interface Usuário-Rede (UNI) é ilustrado na Tabela 3.3.

3.5.1.3 Operação e Manutenção (OAM) no Nível Físico

O princípio de OAM descrito na recomendação I.432 do ITU-T é para prever e remediar eventuais falhas do sistema, como também informar a outras entidades sobre esse acontecimento. Para isso a Recomendação I.610 descreve um mínimo de cinco funções para a manutenção das camadas Física e ATM (Seção 2.5).

As funções de OAM (*Operation Administration and Maintenance*) estão concentradas em três categorias principais como mostrado na Tabela 3.4.

Overhead	Codificação	Função
A1, A2	A1:11110110, A2:00101000	Alinhamento de quadro
C1	00000001-00000010-00000011	Identificador de STM-1***
B1	BIP-8	Monitoramento de Erro no nível de seção (STS-3c anterior)
B2	BIP-24 (STS-3c)	Monitoramento de erro de linha
H1 (1-4)	0110 (norm), 11001 (act)	*Flag* novo de dados (indica modificação no valor do apontador)*
H1-H2 (7-16)	0000000000-1100001110	Valor do apontador
H1*, H2*	10010011, 11111111	Indicação de concatenação*
H3		Ação do apontador ***
K2 (6-8)	111,110, Qualquer valor diferente de 110	AIS (*Alarm Indication Signal*)de linha, RDI (*Remote Defect Indication*)de linha, Remoção de linha RDI
3ª octeto Z2 (2-8)	B2 Contador de erro	FEBE (*Far End Block Error*) de linha (até [(8 vezes 3) + 1]valores para STS-3c)
J1		Trace de caminho STS
B3	BIP-8	Monitoramento de erro de caminho (SPE anterior)
C2	00010011	Indicador de sinal no nível de caminho
G1 (1-4)	B3 Contador de erro	FEBE de caminho (até 9 valores legais)
G1 (5)	0 ou 1	RDI de caminho***

* Os bits H1, H2 são todos setados para 1 no caso AIS caminho
** Também utilizando para indicar perda de delineamento de célula
*** Para todos octetos de *overhead*
** Também utilizando para indicar perda de delineamento de célula.

Tabela 3.3: Octetos de overhead de um quadro STS-3c utilizado na RDSI-FL.

Funções	Octetos de *Overhead*	
Monitoramento de Desempenho	Monitoramento de erro no cabeçalho	Erro(corrigível/ não corrigível)
	Monitoramento de canal	B2, Z2 (18-24)
	Monitoramento de erro de caminho	B3, G1 (1-4)
	Monitoramento de erro de seção	B1
Administração de Falhas	AIS de caminho STS	H1, H2, H3
	RDI de caminho STS	G1(5)
	Perda de delineamento de célula/RDI de caminho	G1(5)
	Indicação de alarme de canal	
	Indicador de Sinal (AIS) e falha remota (RDI)	K2 (6-8)
Facilidade de Teste	Verificação de Conectividade Teste de caminho	J1

Tabela 3.4: Funções de Operação e Manutenção (OAM).

O Monitoramento de Desempenho inclui funções de coleta de informações sobre o comportamento e elementos da rede para avaliar e relatar seu desempenho. Esta operação envolve vários campos de *overhead* do quadro SDH; o *octeto* B1 é utilizado no nível de seção para detectar violações de código, juntamente com os octetos B2, Z2 no nível de linha, e o *octeto* B3 e os bits G1(1-4) no nível de caminho. O monitoramento na UNI é feito calculando-se o BIP-8 (*Byte Interleaved Parity*) de seção, o BIP-24 de linha e o BIP-8 de caminho dos quadros que estão chegando e comparando-se os valores obtidos com os valores codificados nos octetos pelo nó

transmissor. Os octetos de linha B2 e caminho B3 relativos ao FEBE (*Far End Block Error*) são usados para transportar de volta aos equipamentos, em níveis superiores, o número de erros de BIP detectados nos equipamentos terminais de linha ou de caminho.

A Administração de Falhas permite a detecção, o isolamento e a correção de condições de falhas na rede. Isto é feito usando-se os sinais de falhas recebidos, as falhas nos equipamentos, detecção ou remoção de sinais de AIS (*Alarm Indication Signal*) ou RDI (*Remote Defect Indication*). As falhas detectadas nos sinais recebidos são: Perda de Sinal (LOS - *Loss Of Signal*), Perda de Quadro (LOF - *Loss Of Frame*), Perda de Apontador (*Loss Of Pointer*), e rótulo de sinal perdido.

A Facilidade de Teste (*Path Trace*) permite a verificação contínua da conexão entre dois equipamentos terminais de caminho. O equipamento suportando a UNI provê esta função enviando o código apropriado de 64 octetos no octeto J1 do *Overhead* de Caminho[4].

A perda de delineamento de célula provoca a geração de um alarme RDI de caminho para alertar o equipamento terminal de caminho, de nível superior, da falha detectada na camada inferior. O defeito de perda de delineamento de célula é precedido pelo sinal Fora dos Limites de célula (OCD - *Out of Cell Delineation*), que quando persistir torna-se um alarme LCD (*Loss of Cell Delination*).Os equipamentos que suportam a UNI deverão declarar um defeito tipo LCD após anomalias persistentes de OCD e gerar um RDI de caminho.

3.5.2 Camada Física para Interface de 100 Mbps

As funções associadas à camada física para Interface de 100 Mbps estão agrupadas nas subcamadas de Meio Físico (*Physical Medium Dependent* - PMD) e Convergência de Transmissão (*Transmissão Convergence* - TC). Estas funções estão descritas na Tabela 3.5.

Subcamada de Convergência de Transmissão	Delimitação de Célula. Geração/Verificação de HEC.
Subcamada de Meio Físico	Temporização de bit, Codificação de linha. Meio Físico.

Tabela 3.5: Funções da Camada Física.

A UNI privada conecta equipamentos tais como *bridges*, computadores, roteadores, e estações de trabalho, a uma porta de um comutador ATM.

Esta interface da camada Física é baseada na camada Física da FDDI[4].

3.5.2.1 Especificações da Subcamada de Meio Física

A subcamada de Meio Físico trata dos aspectos que são dependentes do meio de transmissão. Ela é responsável da codificação e decodificação dos sinais. O esquema de codificação usada é o 4B/5B para fibra óptica e MLT-3 (*Multi-Level Transmit – 3Levels*) para par trançado.

Esta camada Física segue as especificações da subcamada de meio Físico da FDDI[4]. O enlace física usa uma fibra multimodo a 100 Mbps (125 Mbauds de taxa). O transmissor óptico e a largura de banda da fibra devem respeitar as especificações do ISO DIS 9314-3[4].

O conector da fibra é um conector *duplex* tipo MIC (usado nas redes Token Ring da IBM) especificado para interfaces FDDI. Os cabos de transmissão e de recepção não podem ser acidentalmente trocados com o conector *duplex*.

3.5.2.1.1 Codificação de linha para Enlace de Fibra

O esquema de codificação da enlace de fibra é baseada no código 4B/5B da ANSI X3T9.5 (FDDI)[4]. O sistema ANSI X3T9.5 usa um padrão de dados paralelos

de 8 bits. Esse padrão é dividido em *nibbles* (4-*bit*), codificados em símbolos de 5-*bit*. Dezesseis das trinta e duas possibilidades de símbolos (provido pelos 5 bits) são escolhidos para representar os dezesseis padrões de dados de entrada. Alguns a são reservados como símbolos de comandos.

A Tabela 3.6 mostra os códigos de controle definidos para uma interface local de 100 Mbps

Mnemônico	Definição
JK (Sync)	Ocioso
II	Reservado
TT	Início de Célula
TS	Reservado
IH	Não Recomendado
TR	Reservado
SR	Reservado
SS	Não Usado
HH	Não Recomendado
HI	Não Recomendado
HQ	Não Recomendado
RR	Não Usado
RS	Reservado
QH	Não Recomendado
QI	Não Recomendado
QQ	Perda de Sinal

* Os códigos rotulados "Reservado" são reservados para definições futuras.
* Um receptor não precisar tomar nenhuma ação quando encontrar os códigos "Reservado", "Não Recomendado", ou "Não Usado".

Tabela 3.6: Código de Controle de uma Enlace de Acesso ATM.

3.5.2.2 Codificação de linha para Par Trançado não Blindado (UTP)

Recentemente houve um interesse considerável para transmissão de alta velocidade (100 Mbps ou 155 Mbps) sobre cabos do tipo par trançado não blindado

(UTP) para redes locais. Como há regulamentos estritos de interferência eletromagnética (*Electro-Magnetic Interference* - EMI) para equipamentos de comunicação, os esquemas de transmissão usando cabos UTP são influenciados por essas restrições de radiações eletromagnéticas nas frequências a cima de 30 Mhz. O padrão de cabo do tipo par trançado Ethernet (IEEE 802.3 10BASE-T) bem como o padrão de 16 Mbps IEEE 802.5 podem restringir seus respectivos espectro de frequência de sinal a baixo de 30 Mhz para atingir os requisitos de EMI.

Para isso foi escolhido a codificação MLT-3 (*Multi-Level Transmit – 3Levels*). Essa tipo de codificação é bastante parecido com a codificação NRZI (*Non Return to Zero Inverted*), mas usando 3 níveis de sinal em vez de dois. O princípio da codificação MLT-3 esta ilustrado na Figura 3.6.

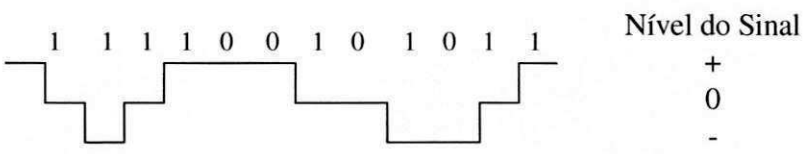


Figura 3.6: Codificação MLT-3.

Como na codificação NRZI, um bit zero é sinalizado por uma ausência de mudança no *status* da linha tomando por referência o bit anterior. Um bit 1 é sinalizado por uma transição. Tem-se três níveis de sinais em vez de dois. As regras do MLT-3 são as seguintes:

um bit 1 é sinalizado por uma mudança no estado da linha a partir do bit anterior;

um bit 0 é sinalizado pela ausência de mudança no estado da linha;

quando a linha está no estado 0, a próxima mudança de estado (para sinalizar o próximo bit 1) deve estar no estado inverso ao estado imediatamente anterior. Portanto se a linha estivesse no estado –1 e um bit 1 é sinalizado para ir ao estado 0 e um próximo bit 1 é sinalizado para ir ao estado +1.

A justificativa desse código é a utilização menor da largura de banda em relação a codificação NRZI[16]. Isso significa que o sinal é concentrado em frequências menor do que as do NRZI conservando uma estrutura relativamente simples. Uma consequência disso é a utilização de cabos UTP-5 com codificação MLT-3 em interfaces FDDI.

3.5.2.3 Taxas de Transmissão e Temporização de Bit

O enlace físico opera a uma taxa de 100 Mbps, a mesma taxa definida para FDDI. A temporização no enlace *full duplex* usa a mesma frequência base para cada direção. O enlace físico opera como dois enlaces *simplex* com a mesma frequência nominal[4]. A unidade de interface de rede pode gerar seu relógio de transmissão localmente.

3.5.2.4 Subcamada de Convergência de Transmissão (TC)

A subcamada de convergência de transmissão trata dos aspectos da camada física que são independentes das características do meio de transmissão. A maioria das funções que provê a subcamada de convergência de transimssão é voltada para a geração e processamento de alguns octetos de *overhead* contidos no *overhead* de transmissão e no cabeçalho da célula ATM.

3.5.2.4.1 Linha Ociosa

Quando ocioso, o meio de transmissão contém códigos contínuos sinalizando o estado de linha ociosa. Esses códigos na enlace ATM são o código JK. A linha está ociosa a não ser que células estejam sendo ativamente transmitidas.

3.5.2.4.2 Delimitação de Células

As fronteiras das células são assíncronas isto é, elas podem aparecer a qualquer momento que a linha estiver ociosa. A camada Física provê um fluxo descontínuo de células para a camada ATM. Cada célula é precedida por um código TT seguida de 53

pares de símbolos de célula. O código TT sinaliza o Início da Célula no receptor. Os 54 pares de símbolos (53 símbolos representando 53 octetos mais o código de Início de Célula) devem ser contíguos na linha.

3.5.2.4.3 Geração/Verificação de HEC

Esta operação é igual para todos os sistemas de transmissão independentemente da interface utilizada.

3.6 Resumo

Neste Capítulo demos ênfase à camada Física do modelo de referência RDSI-FL, pois isso nos permitirá entender melhor a proposta da arquitetura da Interface de Linha Física para o COMATM. Foram escolhidos alguns exemplos que para nós, são mais relevantes: a interface baseada em SDH, o padrão predominante da tecnologia ATM e a interface 100 Mbps baseada no FDDI.

Capítulo

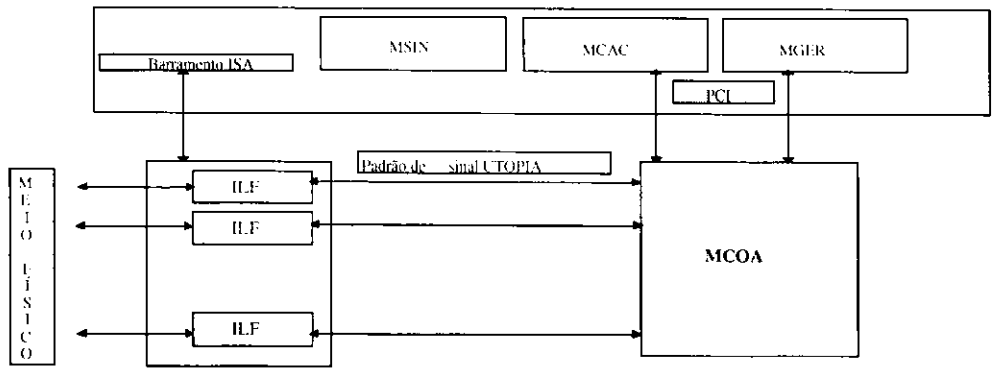
4

Interface de Linha Física para o COMATM

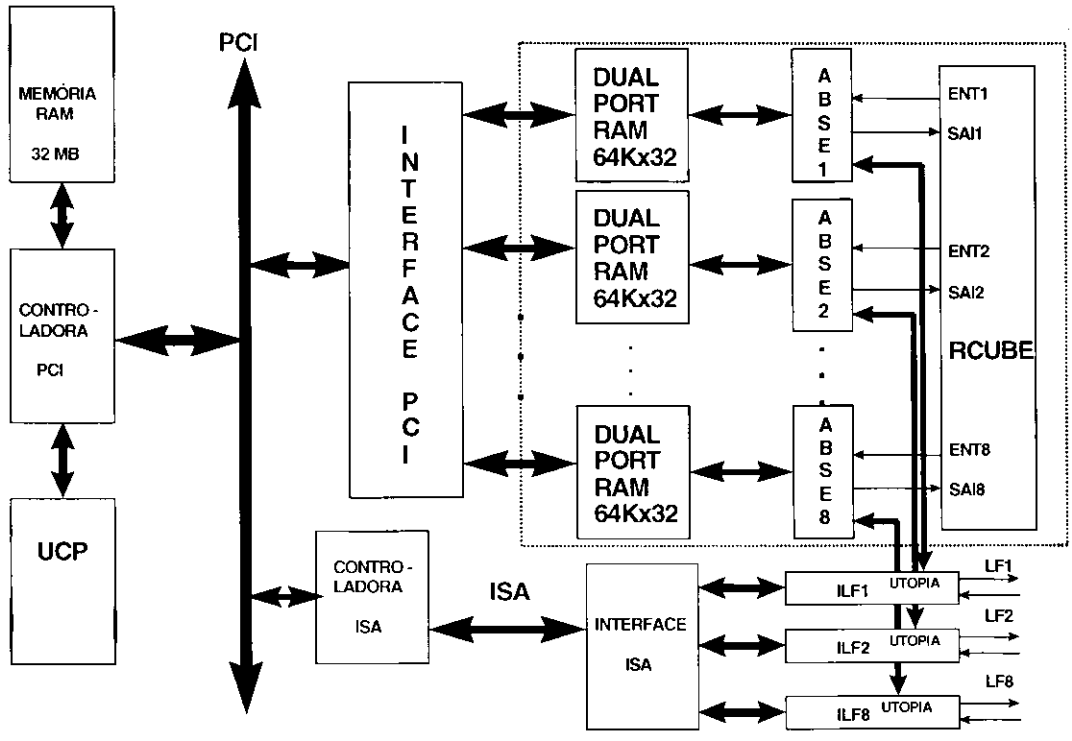
O presente Capítulo trata da especificação da Interface de Linha Física (ILF) para o projeto de um comutador ATM (COMATM)[8,17]; traz um roteiro de implementação para tais interfaces propondo uma ILF genérica para qualquer tipo de sistema de transmissão.

4.1 O COMATM.

O COMATM é um comutador ATM concebido por Giozza e Zerrouk[8]. O protótipo COMATM, cuja arquitetura é mostrada Figura 4.1, é uma interface de um sistema hospedeiro, que pode ser um microcomputador IBM-PC ou estação de trabalho que possua barramento padrão PCI. O COMATM transmite as células ATM no meio físico através da Interface de Linha Física (ILF) usando o padrão de sinalização UTOPIA [8]. A ILF é uma interface do barramento ISA provendo todas as funcionalidades de transmissão e recepção das células do COMATM para o meio físico e também para o sistema hospedeiro, e vice-versa.



a) Módulos Funcionais Segundo Modelo de Referência COMATM



p) ILF: Interface de Linha Física MCOA: Módulo de Comutação ATM

q) MGER: Módulo de Gerência MCAC: Módulo de Controle de Admissão

r) MSIN: Módulo de Sinalização

s) a seta ——— indica linha paralela a seta —— indica linha serial

b) Diagrama de Bloco do COMATM.

Figura 4.1: Arquitetura do COMATM.

O Módulo de Comutação ATM (MCOA), ilustrado Figura 4.1, é constituído de um conjunto de circuito integrados em volta da matriz de comutação RCUBE[8]. Cada MCOA é constituído de 8 módulos chamados de elementos básicos de comutação ATM (ABSE), um para cada entrada/saída do RCUBE, que podem ser conectados a até 8 linhas físicas através das ILFs. As ILFs se ligam ao sistema hospedeiro através do barramento ISA para enviar informações de gerenciamento e de inicialização e, aos ABSEs, através de uma interface padronizada pelo Forum ATM chamada de UTOPIA.

As ILFs recuperam do meio físico, de forma serial, as células que uma vez paralelizadas, vão ser entregues aos ABSEs. Os ABSEs por sua vez consultam a Tabela de Estado de Conexão, contida em uma memória RAM, extraem os parâmetros de roteamento, inserem os dados em um formato de 32 octetos, e os envia em série para o roteador RCUBE. O roteador RCUBE roteia e coloca os dados na saída especificada para o correspondente ABSE. O ABSE na saída do RCUBE que paraleliza os dados, e os entrega à ILF correspondente através de um barramento de 8 bits. A ILF, por sua vez, serializa os octetos recebidos do ABSE e os coloca no meio físico.

A ILF implementa as seguintes funções:

t) Interface com o meio físico e sincronização dos bits;

u) Geração da seqüência de HEC (*Header Error Control*) e verificação de erros no cabeçalho da célula.

v) Embaralhamento e desembaralhamento no caso do sistema de transmissão baseado em SDH/SONET ;

w) Mapeamento das células dentro dos quadros SDH. ;

x) Delimitação das células;

y) Detecção e Geração de Células de Gerenciamento de nível Físico.

A ILF pode ser utilizada para prover mecanismos de transferência de células ATM usando os sistemas de transmissão SDH/SONET ou TAXI [18]. A escolha de um desses dois sistemas de transmissão é feita a partir de um *software* de inicialização que acompanha a ILF.

A interface ILF, no seu funcionamento típico de transmissão, recebe células das saídas do Módulo de Comutação ATM (MCOA). A partir do cabeçalho das células recebidas, a ILF gera o octeto de controle de erros HEC (*Header Error Control*) e serializa a célula ATM completa para transmissão no meio físico, segundo um protocolo específico (por exemplo TAXI 100 Mbps, SDH/SONET 155 Mbps etc...). Por outro lado, no funcionamento de recepção, a ILF recebe do meio físico seqüências de bits em série, identifica as células com seus cabeçalhos (delimitação de células), verifica erros no cabeçalho e transmite para o MCOA, em paralelo, as células válidas, e descartando as corrompidas. Além disso, a ILF é também responsável pela detecção e geração de células de gerenciamento OAM de nível Físico.

A ligação física entre cada ILF e o MCOA é feita através dos circuitos elementares de comutação ATM chamados ABSE (- *ATM Basic Switch Element*). Os ABSEs são realizados com ASICs e compõem as portas da matriz de comutação ATM implementada pelo MCOA[17]. Em cada circuito ABSE existe um sub-bloco denominado Interface com a Camada Física (ICF/ABSE) que implementa as funções de interfuncionamento Camada Física/Camada ATM segundo o padrão UTOPIA[19]. A comunicação entre a ILF e o ABSE é feita na base de octetos seguindo o padrão de sinais de controle UTOPIA[19] conforme ilustrado na Figura 4.2. Na implementação baseada no protótipo COMATM, a ILF provê também funções de serialização/paralelização em conformidade com o padrão de transmissão TAXI (*Transparent Asynchronous Xmitter Interface*)[18] ou SDH/SONET.

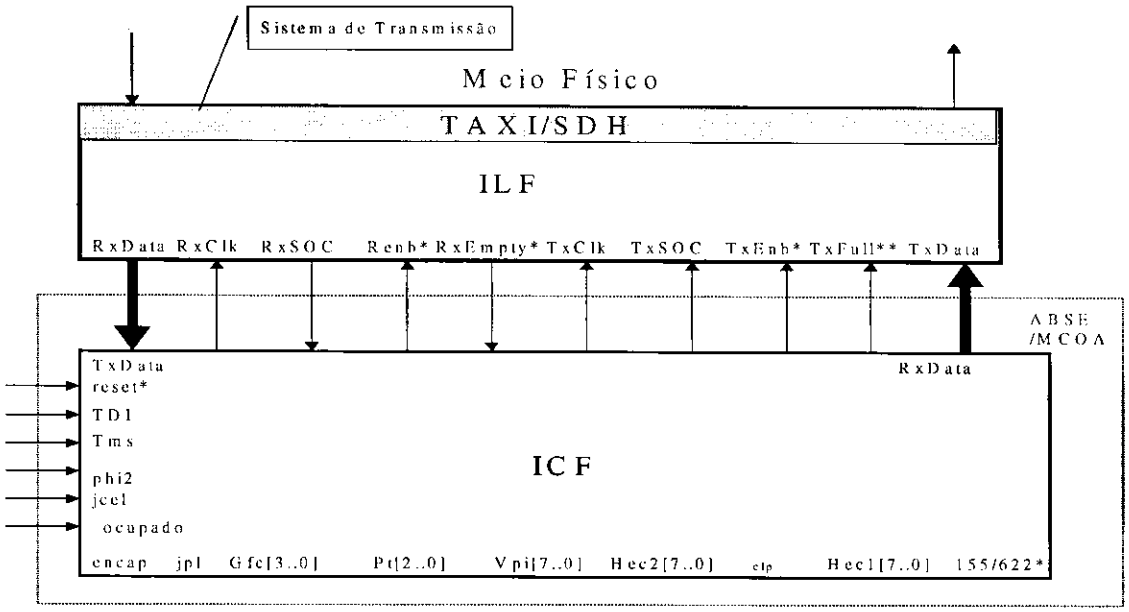


Figura 4.2: Interface UTOPIA entre a ILF e o MCOA (ICF/ABSE).

4.2 A Interface de Linha Física (ILF) do Comutador ATM COMATM

Para desenvolver uma arquitetura flexível e modular, partimos do princípio de particionar a Interface de Linha Física (ILF) em vários blocos funcionais independentes mas interligados entre si, cada bloco efetuando tarefas específicas bem determinadas.

A ILF recebe as células ATM do meio físico numa taxa de 100 Mbps no caso do padrão TAXI, e 155 ou 622 Mbps no caso do SDH/SONET, envia e recebe informações de gerenciamento OAM de nível Físico ao sistema hospedeiro através de um barramento ISA, e se comunica com o ABSE do Módulo MCOA, através da interface UTOPIA padronizada pelo Forum ATM[20]. A ILF é composta de dois módulos (Transmissão e Recepção) conforme ilustrada na Figura 4.3. A descrição detalhada destes módulos será feita nas seções a seguir. Além destes dois módulos temos 3 blocos funcionais que completam a arquitetura da ILF; sendo eles uma memória ROM, uma memória RAM e o bloco Teste.

4.2.1 Módulo de Transmissão

O módulo de Transmissão da ILF, constituído de um conjunto de blocos funcionais, é responsável pela recepção das células oriundas do MCOA (ABSE), da geração de cabeçalho no nível físico, do embaralhamento e mapeamento das células (se for empregado o SDH como sistema de transmissão), da serialização das células montadas, da sua codificação (por exemplo 4B/5B para sistema de transmissão baseado no padrão TAXI), e da transmissão destas no meio físico. O Módulo de Transmissão detalhado na Figura 4.4 é composto dos blocos: Controle de Transmissão, Gerador de Cabeçalho, *Buffer* de Transmissão, *TaxiChip Transmitter*, Mux para Sistema de Transmissão, Embaralhador, Mapeador de Célula e do *SDH Transmitter*. A descrição funcional destes blocos será feita a seguir.

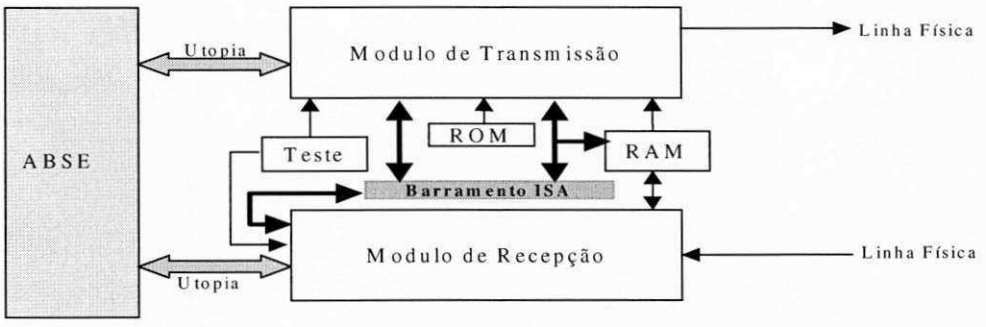


Figura 4.3: Diagrama em Módulos da ILF

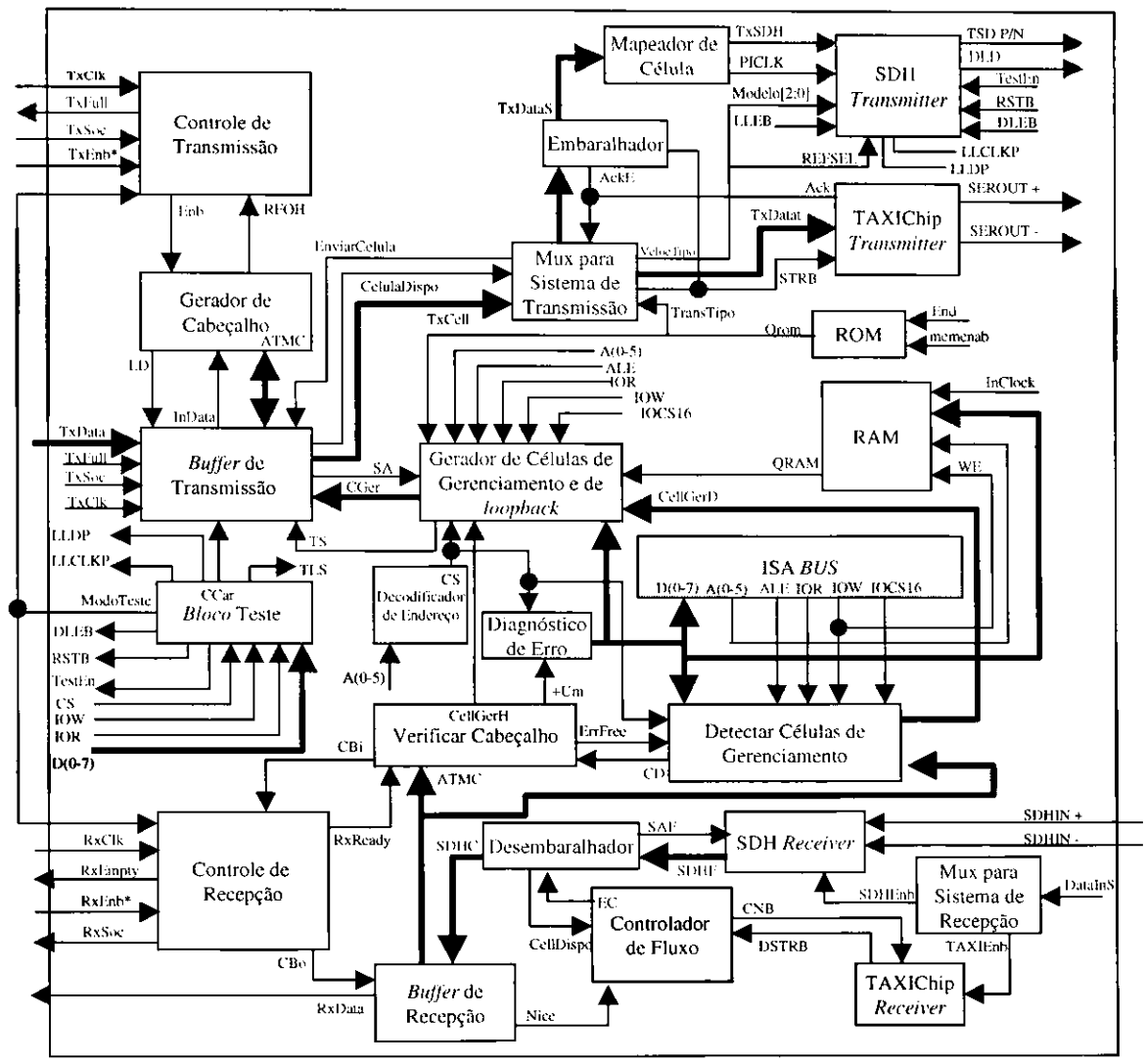


Figura 4.4: Arquitetura da Interface da Linha Física ILF.

4.2.1.1 Controle de Transmissão

O bloco Controle de Transmissão, ilustrado Figura 4.5 trata da transferência das células vindas do MCOA para a ILF e armazenadas no *Buffer* de Transmissão. O protocolo utilizado para receber e controlar o fluxo das células vindas da ABSE é o padrão UTOPIA. O bloco Controle de Transmissão se comunica com o bloco Gerador de Cabeçalho recebendo dele o sinal RFOH (*Ready For an Other Header*), e enviando o sinal Enb. O sinal RFOH avisa ao bloco Controle de Transmissão que o

Gerador de Cabeçalho está pronto para iniciar uma nova operação de cálculo do HEC (*Header Error Control*) e assim gerar um novo cabeçalho. Com a ativação do sinal TxEnb* pelo MCOA, o bloco Controle de Transmissão espera a ativação do sinal TxSoc antes de verificar se o Gerador de Cabeçalho está pronto para iniciar uma operação de cálculo do HEC. Assim se o sinal RFOH estiver ativado, o Controle de Transmissão solicita, através da ativação do sinal TxFull*, que o MCOA envie mais 4 octetos se a interface for de 8 bits e, mais 8 octetos, se a interface for de 16. Em seguida, o bloco Controle de Transmissão ativa o sinal Enb para indicar ao Gerador de Cabeçalho a presença de dados válidos.

Figura 4.5: Bloco Controle de Transmissão.

4.2.1.2 Gerador de Cabeçalho

O bloco Gerador de Cabeçallho, ilustrado na Figura 4.6, calcula o HEC e o insere no *Buffer* de Transmissão para completar o cabeçalho da célula a ser transmitida. A geração do cabeçalho é feita utilizando o polinômio gerador $X^8 + X^2 + X + 1$ que vai servir de divisor dos 4 primeiros octetos da célula constituída pelos campos GFC, VPI, VCI, PT, CLP (armazenados no *Buffer* de Transmissão), deslocados de 8 posições a esquerda. Após a divisão, uma operação XOR será efetuada entre o resto da divisão e o octeto 01010101, resultando no 5° octeto, o HEC. Uma vez terminada esta operação, o Gerador de Cabeçalho insere o HEC no *Buffer* de Transmissão, ativando o sinal ATMC (*ATM Cell*), e ativa os sinais, RFOH (*Ready For Anothe Header*) para avisar

ao bloco Controle de Transmissão que está pronto para montar uma nova célula, e o sinal LD (Libera Dados) , utilizado para solicitar a liberação da célula armazenada no *Buffer* de Transmissão. O Gerador de Cabeçalho espera a ativação do sinal Enb *(Enable)* por parte do Controle de Transmissão para iniciar uma nova operação de cálculo do HEC.

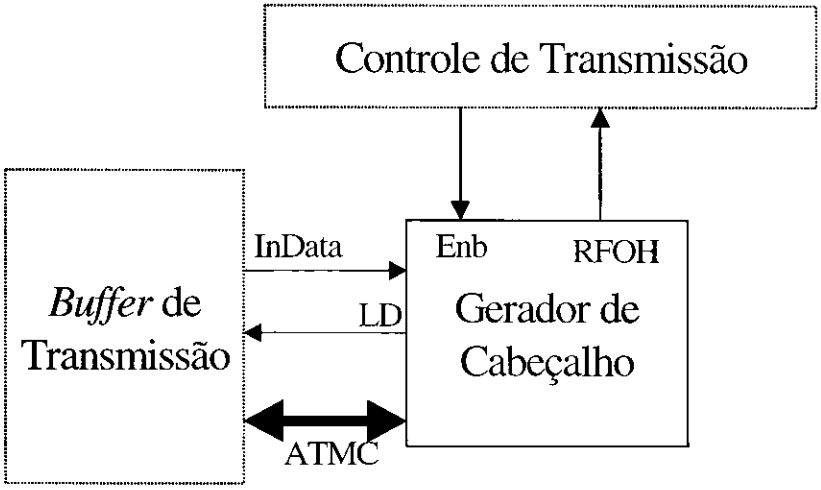


Figura 4.6: Bloco Gerador de Cabeçalho.

Para resolver o gargalo temporal que pode gerar o cálculo do HEC, projetamos uma arquitetura para implementar o algoritmo de Rocksoft[21]. Este algoritmo permite o cálculo de CRCs de forma paralela. O HEC é um CRC com algumas características específicas. No Anexo A é descrito como funciona o CRC e quais são as alternativas de implementação.

4.2.1.3 *Buffer* de Transmissão

O *Buffer* de Transmissão, ilustrado Figura 4.7, recebe as células do MCOA (ABSE) na base de octetos e as transmite para o Mux para Sistema de Transmissão. A transferência das células (octetos) armazenadas no *Buffer* de Transmissão é controlada pelo bloco Gerador de Cabeçalho através do sinal LD (Libera Dados) que fica ativado quando o HEC tiver sido inserido no cabeçalho da célula. O *Buffer* de Transmissão se comunica com o bloco Mux para Sistema de Transmissão transmitindo para ele a célula já montada. O *Buffer* de Transmissão pode receber também células (de

gerenciamento OAM, etc...) do sistema hospedeiro através do Gerador de Células de Gerenciamento e de *Loopback*, células estas a serem transmitidas para o meio físico. A transmissão das células de gerenciamento de nível Físico, é feita através de troca dos sinais TS (Transmissão Solicitada) para solicitar a permissão de transmitir. Quando o *Buffer* de Transmissão estiver pronto para atender esta solicitação, ele envia o sinal SA (Solicitação Aceita) para o Gerador de Células de Gerenciamento e de *Loopback* que, por sua vez, envia as células de gerenciamento OAM de nível Físico pelo sinal CGer. O mesmo protocolo de sinais é usado pelo bloco Detectar Células de Gerenciamento para solicitar a geração de células de gerenciamento de nível Físico.

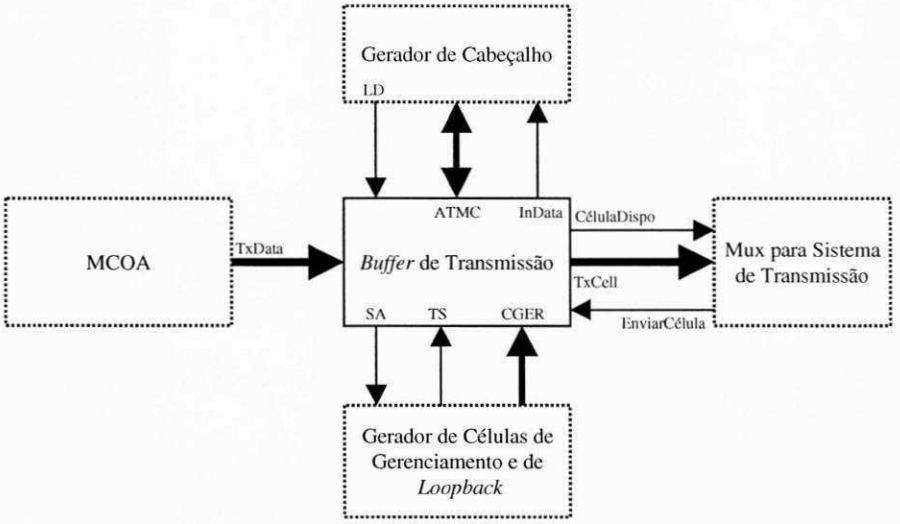


Figura 4.7: Buffer de Transmissão.

4.2.1.4 Mux para Sistema de Transmissão

Durante a inicialização do sistema, este bloco ilustrado na Figura 4.8, através do sinal TransTipo recebido da memória ROM que armazena a informação sobre o tipo de sistema de transmissão, é ativado para usar um dos dois sistemas de transmissão. Uma vez ativado, o Mux para Sistema de Transmissão habilita uma das saídas para o *TaxiChip Transmitter* ou para o Embaralhador do *SDH-Transmitter*, e sua função será de receber células do *Buffer* de Transmissão e transmití-las ao dispositivo de transmissão inicialmente selecionado. Um protocolo de controle de fluxo é estabelecido com o *Buffer* de Transmissão pelos sinais EnviarCelula e CélulaDispo.

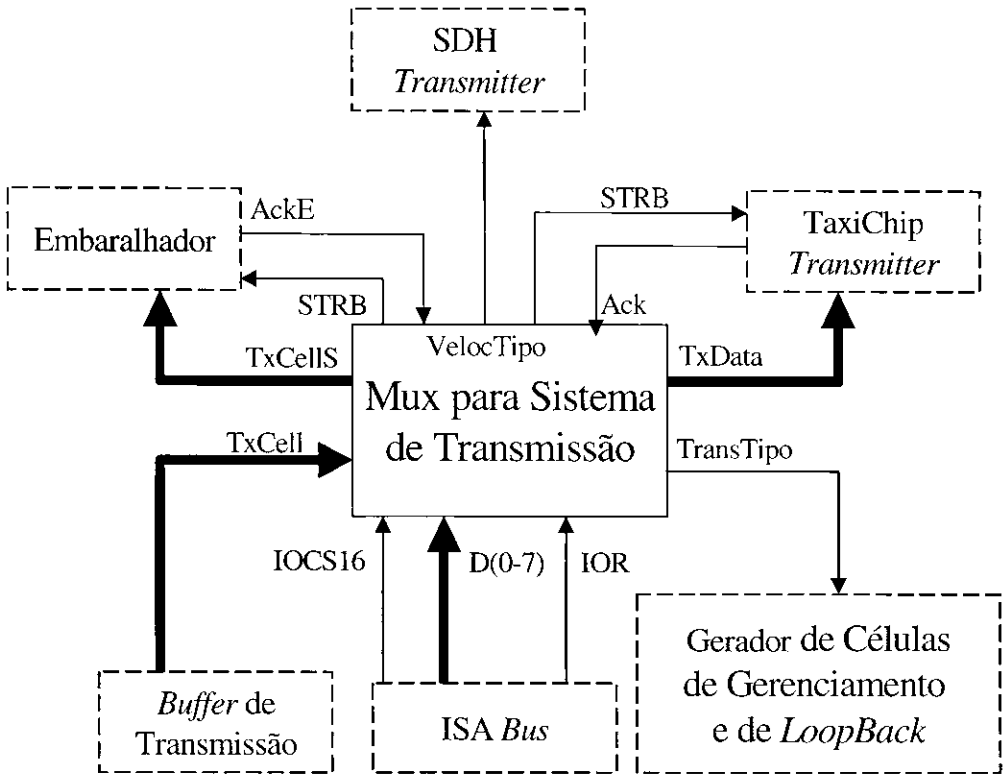


Figura 4.8: Mux para Sistema de Transmissão.

4.2.1.5 TaxiChip Transmitter

O TaxiChip Transmitter é um bloco consistindo de um *Latch* de entrada, um codificador, um *Shift Register* paralelo-serial, um multiplicador PLL *(Phase Locked Loop)* e algumas lógicas de controle [18]. Ele recebe um conjunto de octetos do Mux para Sistema de Transmissão, sendo que esses octetos ficam armazenados no *latch* após serem codificados e depois colocados na sua saída serial. O esquema de codificação utilizado é o 4B/5B especificado pelo padrão ANSI X3T9.5. Esta codificação divide 8 bits em dois *nibbles* (i.e. 2 blocos de 4 bits), sendo que cada *nibble* é codificado em um símbolo de 5 bits. Os conjuntos de símbolos de 5 bits codificados são formatados numa fila de dados NRZI (*Non Return to Zero Invert on One*) para a saída do *TaxiChip Transmitter.* O *TaxiChip Transmitter* usa a padronização TAXI para sua comunicação com o meio físico[17]. Uma descrição detalhada do circuito encontra-se no Anexo D.

4.2.1.6 Embaralhador

O bloco Embaralhador, ilustrado na Figura 4.9, é responsável, no caso da interface SDH/SONET, pelo embaralhamento da carga da célula. Para fazer o embaralhamento, aplica-se no campo de carga da célula o polinômio gerador de seqüência pseudo-aleatória X^{43} +1. O embaralhador recebe a célula completa (sinal TxCell) do Mux para Sistema de Transmissão, efetua o embaralhamento seguindo o procedimento acima citado, e entrega a célula (sinal TxDataS) embaralhada para o bloco Mapeador de Célula.

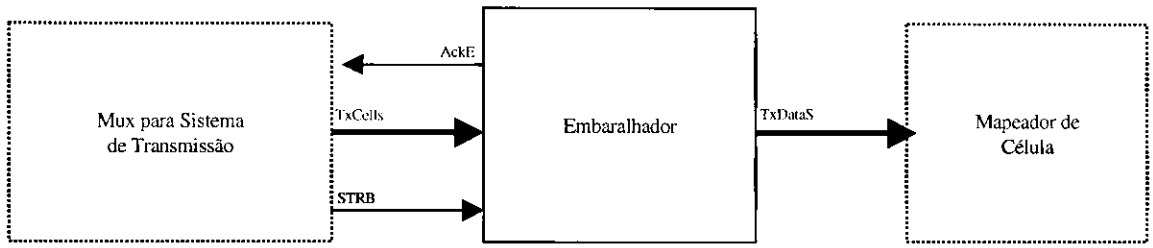


Figura 4.9: Embaralhador.

4.2.1.7 Mapeador de Célula

O bloco Mapeador de Célula, ilustrado na Figura 4.10, recebe células ATM (sinal TxDataS) já prontas para serem transmitidas e, as mapeia dentro dos quadros SDH para serem transmitidos (sinal TxSDH) para o *SDH-Transmitter*. No caso da interface baseada no SDH/SONET, as células são inicialmente mapeadas no C-4 [2], e então transmitidas no container virtual VC-4 que contém adicionalmente octetos de enchimento (adaptação da taxa de transmissão) e informações de controle e gerenciamento. Como a capacidade do C-4 (2340 octetos) não é um múltiplo o tamanho da célula ATM (que é de 53 octetos), uma célula poderá ultrapassar as fronteiras do C-4, embora elas estejam alinhadas com o VC-4 ao nível de octetos.

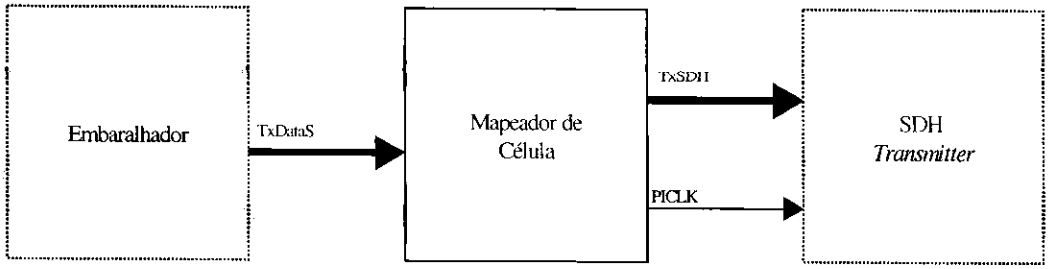


Figura 4.10: Mapeador de Célula.

4.2.1.7.1 SDH-Transmitter

O *SDH-Transmitter* [22] é um circuito integrado de serialização que trabalha com taxas de transmissão de 139,264 Mbps para sistemas de transmissão E4 e 155,52 a 622,08 para sistemas de transmissão do tipo SDH/SONET OC-3 e SDH/SONET OC-12, respectivamente. Esse circuito integrado realiza todas as funções de serialização das células vindas do Mapeador de Célula, em conformidade com os padrões definidos para os sistemas de transmissão SDH e E4. Uma descrição detalhada deste circuito encontra-se no Anexo E.

4.2.2 Módulo de Recepção

O Módulo de Recepção recebe uma série de bits do meio físico, os paraleliza, delimita as células, desembaralha a carga da célula se for preciso, extrai e analisa o cabeçalho. Se não houver nenhum erro no cabeçalho da célula recebida, esta é transmitida para o MCOA ou para o sistema hospedeiro (caso em que a célula for de gerenciamento), caso contrário, a célula é descartada. O Módulo de Recepção detalhado na Figura 4.4 é composto dos blocos: Controle de Recepção, *Buffer* de Recepção (sub-bloco do Controlador de Fluxo), Verificar Cabeçalho, Diagnóstico de Erros, Controlador de Fluxo de Recepção, Desembaralhador, Detectar Células de Gerenciamento, ISA *Bus, SDH-Receiver*, Mux para Sistema de Recepção e *TaxiChip Receiver*.

4.2.2.1 Controle de Recepção

O bloco Controle de Recepção da ILF, ilustrado na Figura 4.11, é responsável pelo controle da transferência das células recebidas do meio físico (sem erro de cabeçalho) para o MCOA, usando sinais padrão UTOPIA. O bloco Controle de Recepção controla o fluxo de células trafegando da ILF para o MCOA, envia um sinal (CBo) para o *Buffer* de Recepção através do Controlador de Fluxo, para habilitar a transmissão da célula para o MCOA (desativando CBo=0) ou, para solicitar o seu descarte (ativando CBo=1). Quando o bloco Controle de Recepção recebe do MCOA a permissão para transmitir uma nova célula (com RxEnb* = 0), ele avisa então ao bloco Verificar Cabeçalho, através do sinal RxReady que está pronto para receber uma nova célula. Se o bloco Controle de Recepção recebe do bloco Verificar Cabeçalho o sinal CBi desativado ou ativado, ele, respectivamente, desativa ou ativa também o sinal CBo, para transmitir ou descartar a célula armazenada no *Buffer* de Recepção do bloco Controlador de Fluxo.

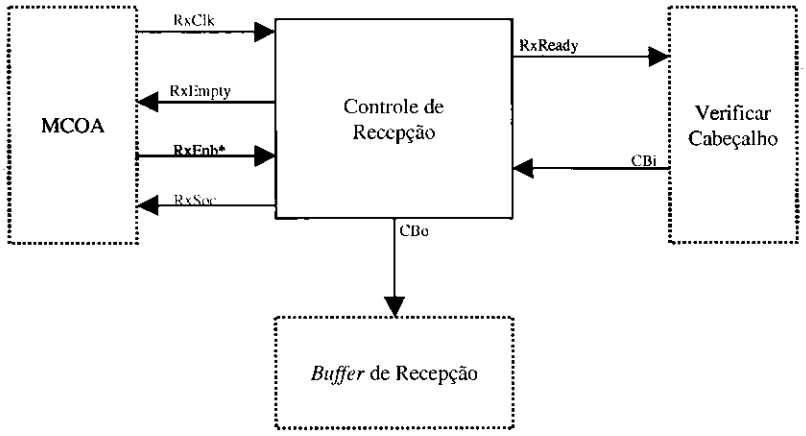


Figura 4.11: Bloco Controle de Recepção.

4.2.2.2 Controlador de Fluxo de Recepção

O bloco Controlador de Fluxo de Recepção, ilustrado na Figura 4.12, é responsável pelo controle do fluxo das células vindas de uma das interfaces de

serialização (*TaxiChip Receiver* ou *SDH-Receiver*) e destinadas aos blocos Verificar Cabeçalho, Detectar Células de Gerenciamento e ao MCOA.

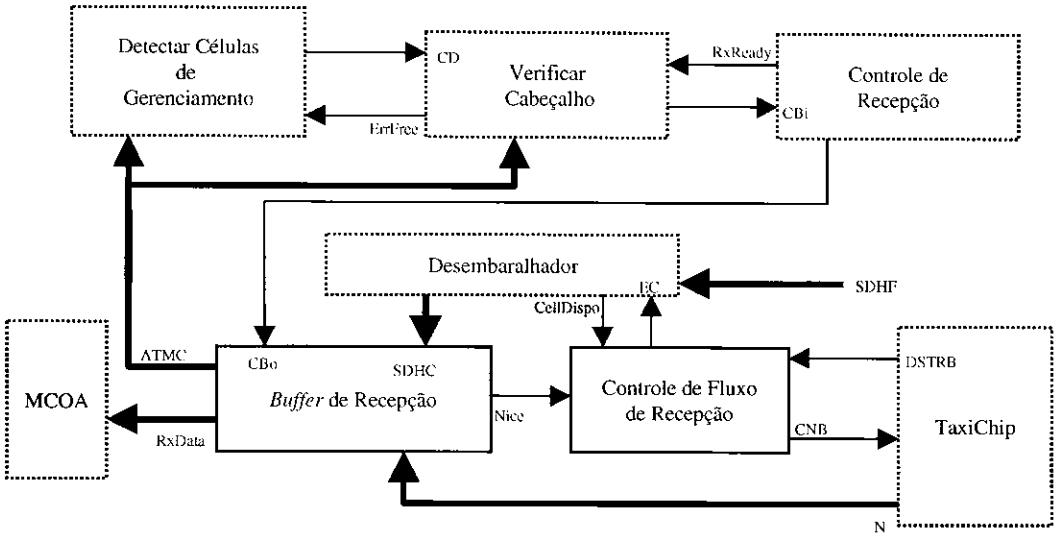


Figura 4.12: Controlador de Fluxo de Recepção e Buffer e Recepção.

Caso o sistema de transmissão empregado seja o SDH/SONET, uma vez a célula desembaralhada, o bloco Desembaralhador avisa ao bloco Controle de Fluxo de Recepção a presença de células ATM através do sinal CellDispo. Quando o sinal Nice vindo do *Buffer* de Recepção for ativo (i.e., Nice = 1), o bloco Controle de Fluxo de Recepção ativa o sinal EC (EC = 1) permitindo assim que o Desembaralhador coloque a célula no *buffer* de Recepção através do sinal SDHC. A célula é então transmitida do *buffer* de Recepção para os blocos Detectar Células de Gerenciamento e Verificar Cabeçalho através do sinal ATMC. Do bloco Controle de Recepção, o *Buffer* de Recepção recebe o sinal CBo solicitando-o transmitir para o MCOA a célula armazenada no *Buffer* de Recepção, caso CBo=0; ou esvaziar o *Buffer* de Recepção, caso contrário. No caso de um sistema de transmissão baseado no padrão TAXI, o Controlador de Fluxo de Recepção fica avisado da presença de dados válidos através do sinal DSTRB (*Data STRoBe)* do *TaxiChip Receiver* e então pode autorizar a liberação das células, ativando o sinal CNB (*Catch Next Byte*). Uma vez que CNB for ativo (CNB = 1), o *TaxiChip Receiver* transmite a célula para o *buffer* de Recepção através do sinal N.

4.2.2.3 *Buffer* de Recepção

O *Buffer* de Recepção, ilustrado na Figura 4.12 é responsável pelo armazenamento das células vindas do *TaxiChip Receiver* e do SDH Receiver. Os cabeçalhos das células armazenadas nele, serão tratados pelo bloco Verificar Cabeçalho antes das células serem descartadas ou transmitidas para o MCOA. Para transmitir ou descartar as células, o *Buffer* de Recepção espera uma sinalização positiva ou negativa do bloco Controle de Recepção através do sinal CBo.

Uma sinalização positiva do bloco Desembaralhador (e do Controle de Recepção) significa para o *Buffer* de Recepção que a célula recebida está sem erro no cabeçalho e pode ser entregue ao MCOA. Neste caso, o sinal CBo (Clear *Buffer*) é colocado em nível baixo (i.é CBo=0). Ao avisar o *Buffer* de Recepção (por intermédio do Desembaralhador) que o MCOA está pronto para receber a célula, o bloco Controle de Recepção ativa o sinal RxReady para avisar ao bloco Verificar Cabeçalho que o *Buffer* de Recepção está pronto para liberar uma outra célula para o MCOA. O bloco Controle de Recepção solicita uma liberação das células armazenadas no *Buffer* de Recepção ao MCOA, se e somente se, o sinal RxEnb* (vindo do MCOA) estiver ativo (i.e., RxEnb*=0).

Uma sinalização negativa do bloco Verificar Cabeçalho significa que a célula recebida está com erro no cabeçalho e deve ser descartada isto é, o *Buffer* de Recepção deve ser esvaziado. Neste caso, o sinal CBo é ativo (i.é CBo = 1).

Quando o *Buffer* de Recepção estiver pronto a receber uma nova célula, ele ativa o sinal Nice colocando-o em nível alto (i.e., Nice = 1).

4.2.2.4 Verificar Cabeçalho

O bloco Verificar Cabeçalho, ilustrado na Figura 4.13, é responsável pela extração do cabeçalho das células oriundas do meio físico. Ele verifica a integridade do cabeçalho efetuando uma divisão polinomial usando o polinômio gerador padrão (X^8 +

$X^2 + X + 1$) e comparando o resultado obtido com o HEC recebido. No caso do resultado da comparação indicar erro, a célula deve ser descartada devido a erro no seu cabeçalho e o bloco Diagnóstico de Erros é avisado a partir do sinal +Um. Por outro lado caso a célula seja de gerenciamento OAM de nível Físico, o sinal CD será ativado em 1 e então o bloco Detectar Células de Gerenciamento é avisado da validade ou não da célula detectada através da ativação ou desativação respectiva do sinal ErrFree. Caso a célula não seja de gerenciamento, este aviso é direcionado ao bloco Controle de Recepção, através do sinal CBi que vai estar no nível baixo.

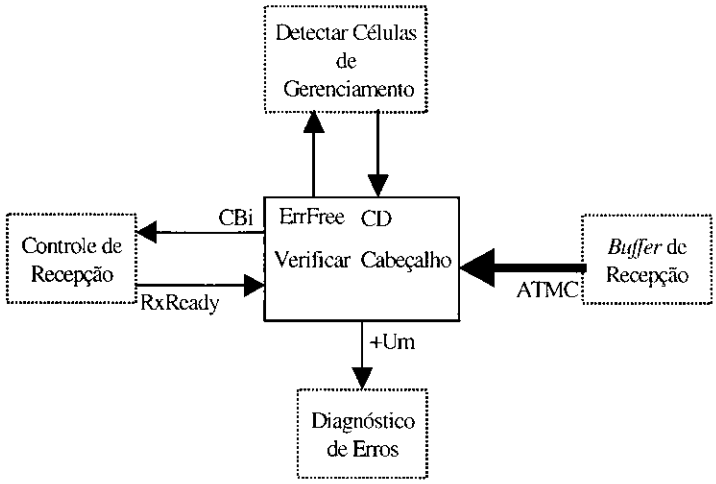


Figura 4.13: Bloco Verificar Cabeçalho.

4.2.2.5 Detectar Células de Gerenciamento

O bloco Detectar Células de Gerenciamento, ilustrado na Figura 4.14, é responsável pela detecção de células de gerenciamento OAM de nível Físico tipo F1, F2 e F3[2]. Uma vez detectadas células de gerenciamento OAM de nível Físico, o bloco Detectar Células de Gerenciamento ativa o sinal CD colocando-o no nível lógico 1. O sinal ErrFree avisa ao bloco Detectar Células de Gerenciamento se houve ou não a erro na célula. As células de gerenciamento são entregues ao sistema hospedeiro via o barramento ISA, usando operações de I/O. Caso a célula de gerenciamento OAM de nível Físico seja de um tipo que necessita a transmissão de uma outra célula de gerenciamento de nível Físico, por exemplo, quando se recebe uma célula de

gerenciamento do tipo TP-AIS (*Transmission Path Alarm Indication Signal*), LOC (*Loss of Cell Delineation*) ou LOM (*Loss of OAM cell*), isso provoca a geração e transmissão para a interface de transmissão, de células de gerenciamento do tipo TP-FERF (*Transmision Path Far End Receive Failure).* O bloco Detectar Células de Gerenciamento através do sinal CellGerD ativa o Gerador de Células de Gerenciamento e *Loopback* para gerar e transmitir célula de gerenciamento de nível Físico ao transmissor caso for preciso.

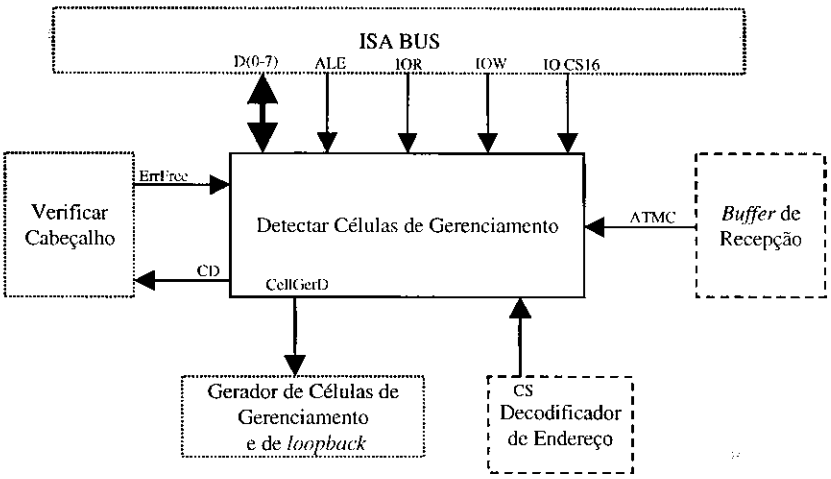


Figura 4.14: Detectar Células de Gerenciamento.

4.2.2.5.1 TaxiChip Receiver

O *TaxiChip Receiver* [18] recebe as células em um conjunto de bits em série codificados (4B/5B) na sua entrada, efetua a decodificação e paraleliza os dados para serem entregues ao bloco *Buffer* de Recepção. Caso forem detectados erros pelo *TaxiChip Receiver,* ele ativa o sinal VLTN para avisar o bloco Diagnóstico de Erros. A presença de dados válidos nas saídas do *TaxiChip Receiver* é sinalizada pelo nível alto do sinal DSTRB. Quando CNB está em nível lógico alto, o *TAXIChip Receiver* captura os próximos símbolos (dados válidos) presentes na linha serial. Uma vez que o fluxo de dados recebido for paralelizado, ele é colocado no *Buffer* de Recepção através do sinal N. O sinal DMS do *TAXIChip Receiver* está conectado ao referencial de tensão elétrica VDD para que possamos trabalhar em modo 8 bits de dados. Uma descrição detalhada do funcionamento do circuito encontra-se no Anexo D.

4.2.2.6 Desembaralhador

O bloco Desembaralhador, ilustrado na Figura 4.15, é responsável pelo desembaralhamento da célula recebida do SDH Receiver (SDHF) usando para tal o polinômio $X^{43} + 1$. Uma vez a operação efetuada, o Desembaralhador transmite a célula pronta ao *Buffer* de Recepção através do SDHC e ativa o sinal SAF para solicitar uma outra célula ao SDH Receiver. Para transmitir a célula desembaralhada, o Desembaralhador ativa o sinal CellDispo para avisar ao *Buffer* de Recepção a presença de célula válida nas suas saídas. Se o *Buffer* de Recepção estiver cheio, o Controlador de Fluxo de Recepção avisa ao Desembaralhador desativando o sinal EC, caso contrário, o sinal EC é ativado permitindo ao Desembaralhador enviar a célula através do sinal SDHC.

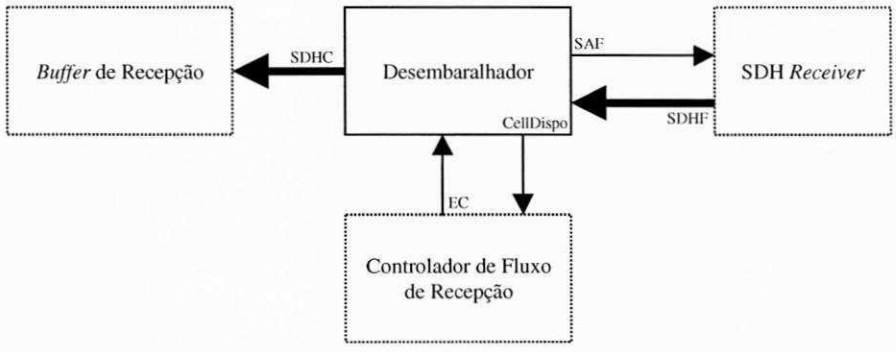


Figura 4.15: Desembaralhador.

4.2.2.7 SDH Receiver

O SDH Receiver implementado pelo circuito SERI (*Synchronous Electrical Receiver Interface*) S3006[22], ilustrado na Figura 4.16, é um dispositivo integrado de paralelização trabalhando com sistemas de transmissão E4 (139,264 Mbps), SONET OC-3 (155,52 Mbps) e SONET OC-12 (622,08 Mbps). Ele efetua todas as funções de serialização e paralelização em conformidade com os padrões de transmissão SDH/SONET e E4. O circuito S3006 efetua a recuperação do relógio, sincronizando-se diretamente ao fluxo de dados. O circuito S3006 realiza também a detecção de quadro SDH/SONET. O *chipset* pode ser utilizado com relógios de 19,44; 38,88; 51,84

e 77,76 Mhz quando opera os modos OC-3c OC-12 do SDH/SONET. O sinal SDHEnb habilita o circuito SDH *Receiver* permitindo-o efetuar as operações de paralelização. A células vindas do meio de transmissão chega ao SDH *Receirver* através do sinal SDHIN. Uma descrição detalhada encontra-se no Anexo E.

z)

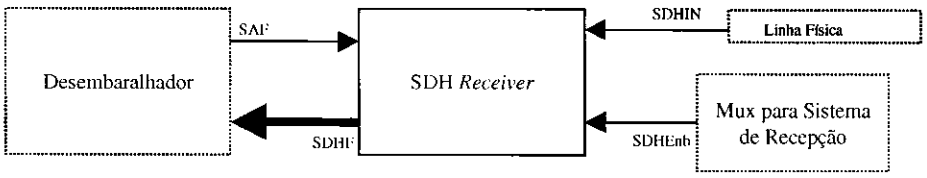


Figura 4.16: SDH Receiver.

4.2.2.8 Mux para Sistema de Recepção

O bloco Mux para Sistema de Recepção ilustrado na Figura 4.16 é responsável pela escolha dos circuitos que vão implementar as funcionalidades de recepção dos padrões SDH (SERI 3006) ou TAXI (TAXIChip *Receiver*). Durante a fase de inicialização do COMATM, o sistema de gerenciamento envia os parâmetros de funcionamento (taxa e sistema de transmissão e serem empregados) ao Mux para Sistema de Transmissão através do barramento de dados D(90-7) usando o sinal de controle IOR. Uma vez definido o sistema de transmissão empregado, através do sinal TransTipo, o circuito SERI S3006 é habilitado ativando o sinal SDHEnb quando TansTipo for diferente de 0, caso contrário, o circuito TAXIChip *Receiver* é habilitado ativando-se o sinal TAXIEnb.

4.2.2.9 Diagnóstico de Erros

O bloco Diagnóstico de Erros, ilustrado na Figura 4.17, é utilizado para avisar ao sistema hospedeiro de algumas circunstâncias de falhas repetidas no cabeçalho da célula, ou da ocorrência de erro na transmissão. Cada vez que acontecer um erro no cabeçalho da célula recebida, o bloco Diagnóstico de Erros fica informado através do sinal +Um. O bloco Diagnóstico de Erros por sua vez aciona um contador para

registrar o acontecimento, e avisa ao sistema hospedeiro através do sinal ErroStat usando o barramento ISA. Quando houver um erro de Transmissão, o bloco Diagnóstico de Erros fica avisado através do sinal VLTN, que lhe notifica de um erro de bit duplo (*Double Bit Error*) no dado, deixando ao sistema hospedeiro a tarefa de efetuar o tratamento adequado relativo a esse tipo de ocorrência.

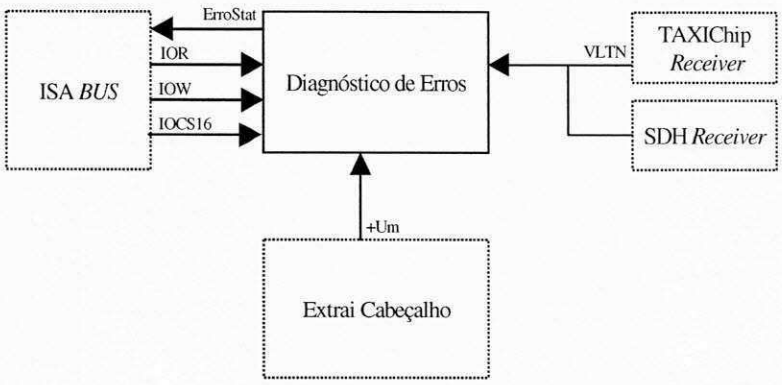


Figura 4.17: Bloco Diagnóstico de Erros.

4.2.3 Memória ROM

No bloco Memória ROM, ilustrado na Figura 4.18, são armazenados os parâmetros de inicialização da ILF (taxa de transmissão, sistema de transmissão etc...) e também algumas células padrões de gerenciamento OAM e de teste.

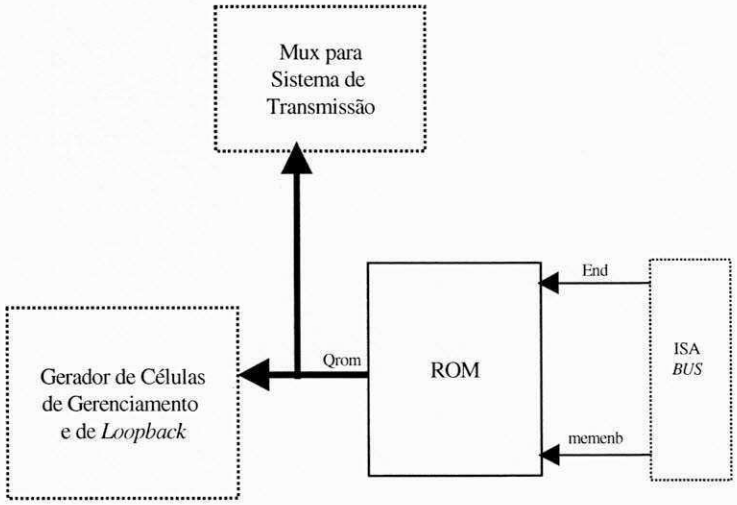


Figura 4.18: Memória ROM.

4.2.4 Memória RAM

O bloco Memória RAM ilustrado na Figura 4.19, é responsável em armazenar células e dados que o sistema hospedeiro quer transmitir para o COMATM ou para a interface de recepção, nos casos em que o sistema hospedeiro quer testar o *status* ou o desempenho do meio de transmissão, por exemplo. Pelo ISA *BUS*, dados como resultado de testes e de diagnóstico das células recebidas são armazenadas na RAM para efeito de análise pela gerência do sistema. Quando o Gerador de Células de Gerenciamento e de *Loopback* necessitar pegar algum dado da RAM, ele aponta para o endereço da RAM e lê o conteúdo usando o sinal QRAM, como ilustra a Figura 4.19. O sistema hospedeiro poderá escrever na RAM selecionando o seu endereço e habilitando o sinal WE através do barramento ISA. A operação de leitura é semelhante à do Gerador de Células de Gerenciamento e *Loopback.*

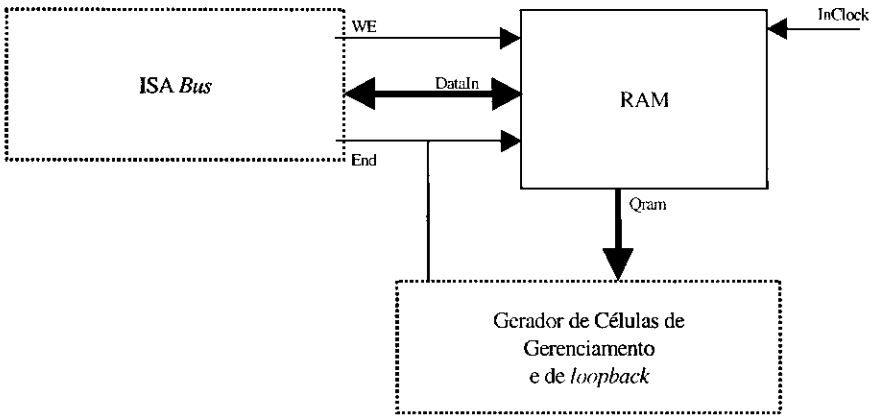


Figura 4.19: Memória RAM da ILF.

4.2.5 Bloco de Teste

Com o bloco de Teste, ilustrado na Figura 4.20, o sistema hospedeiro, através do sistema de gerência, pode executar na ILF operações de teste do meio físico e também de desempenho da interface. Os comandos são transmitidos do sistema hospedeiro para a ILF via o barramento ISA, em operações de I/O, usando um endereço predefinido para apontar ao Bloco de Teste. Este, por sua vez, ativa o sinal

ModoTeste para avisar a todos os blocos envolvidos que a ILF está em modo teste. Assim serão bloqueadas as operações de transmissão e recepção de células ATM. Ao mesmo tempo, dependo do sistema de transmissão empregado, serão ativados os sinais necessários para colocar em modo teste os blocos SDH Transmitter/ SDH Receiver ou TAXIChip Transmitter/TAXIChip Receiver.

4.2.6 ISA Bus

O bloco ISA Bus é o barramento por onde trafegam as células da ILF para o sistema hospedeiro e vice-versa. Utilizamos operações de I/O para transmitir e receber células para/do sistema hospedeiro. Escolhemos este barramento pela vazão satisfatória que ele provê para as necessidades de transferência de células de gerenciamento OAM (64 Mbps), e a simplicidade para a implementação do *software* para interface com o sistema hospedeiro.

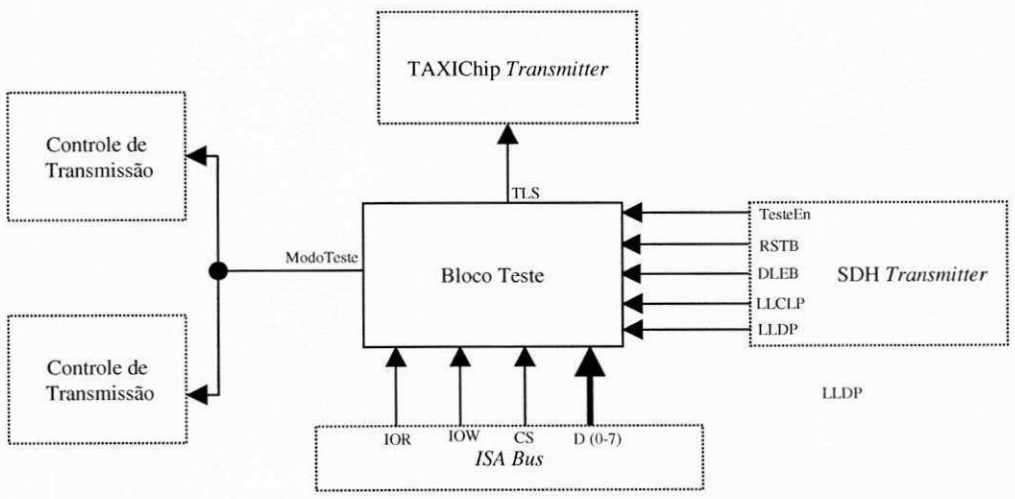


Figura 4.20: Bloco Teste.

4.3 Arquitetura Genérica de Interface de Linha Física para Comutadores ATM

4.3.1 Arquitetura

O objetivo desta proposta é dar uma referência para projetos de arquitetura de Interface de Linha Física para comutadores ATM. A partir desta arquitetura poder-se-ia implementar uma ILF usando SDH/SONET a 155 ou 622 Mbps, par trançado blindado ou não a 100 ou 155 Mbps, DS3 etc...

Para poder elaborar tal arquitetura partimos do princípio de se usar o conceito de herança funcional que permite a todos os objetos (i.e., os sistemas de transmissão, por exemplo sistemas de transmissão baseadas em: SDH/SONET, TAXI, DS3, etc) pertencendo a uma mesma classe (por exemplo, sistemas de transmissão baseados em célula tais como TAXI, 155 Mbps usando par trançado não blindado) possam herdar das funcionalidades dos módulos que atendem a este tipo de objetos. Por exemplo, sabemos que geralmente a codificação usada para sistemas de transmissão de 100 Mbps com fibra óptica é o 4B/5B; portanto, podemos criar um bloco de codificação 4B/5B atendendo a todos sistemas de transmissão que necessitam tal operação. Um outro exemplo mais simples é a geração e verificação do HEC das células ATM, na transmissão e na recepção, respectivamente, operações usadas para todos os tipos de sistemas de transmissão. Portanto, podemos usar somente um bloco de geração de HEC e um de verificação do mesmo para todos os tipos de sistemas de transmissão. Foi usando este conceito que, criamos blocos funcionais atendendo a classes (conjunto de objetos que têm o mesmo comportamento em determinadas situações) de sistemas de transmissão. A arquitetura desta proposta[20] é ilustrada na Figura 4.21.

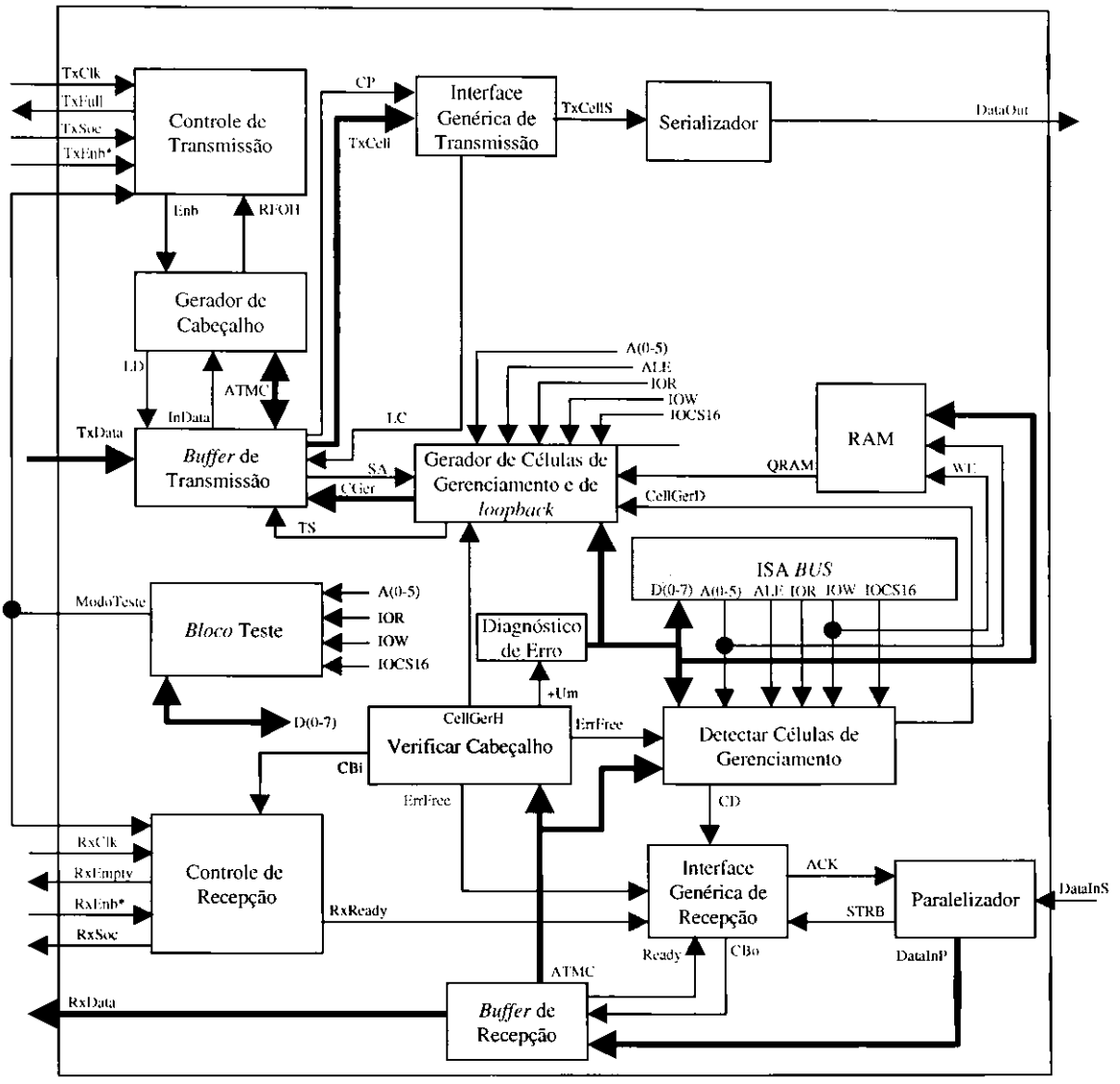


Figura 4.21: Arquitetura de uma ILF Genérica.

Exceto os blocos Serializador, Paralelizador, Interface Genérica de Transmissão e Interface Genérica de Recepção, todos os outros se comportam do mesmo modo que os da arquitetura da ILF para o COMATM, descritos anteriormente. Os blocos Serializador e Paralelizador, como os seus nomes sugerem, efetuam as operações de serialização e paralelização das células a serem transmitidas ou recebidas, respectivamente, pela ILF.

4.3.2 Interface Genérica de Transmissão

O bloco Interface Genérica de Transmissão, ilustrado na Figura 4.22, é responsável pela transmissão e o alinhamento dos bits recebidos do bloco Serializador, efetuando a sua codificação e a conversão eletro-óptica se for necessário. Dependendo do sistema de transmissão, este bloco vai tratar também do mapeamento das células ATM na estrutura de transmissão empregada. A Interface Genérica de Transmissão recebe a célula ATM do *buffer* de Transmissão através do sinal TxCell. Antes do *Buffer* de Transmissão liberar a célula, ele ativa o sinal CP (Célula Pronta) colocando-o no nível lógico 1. Quando a Interface Genérica de Transmissão recebe este sinal (CP = 1), e estiver pronta para tratar a célula ATM a ser enviada, ela ativa o sinal LC (Libera Célula) colocando-o no nível lógico 1 (i.e., LC = 1). Uma vez a célula recebida, a Interface Genérica de Transmissão acescenta ao fluxo de células ATM as informações apropriadas para a delimitação das mesmas e para o transporte de informações de operações de manutenção (OAM) relativas a este fluxo de células. Finalmente, o quadro contendo as células ATM é entregue ao Serializador através do sinal TxCellS.

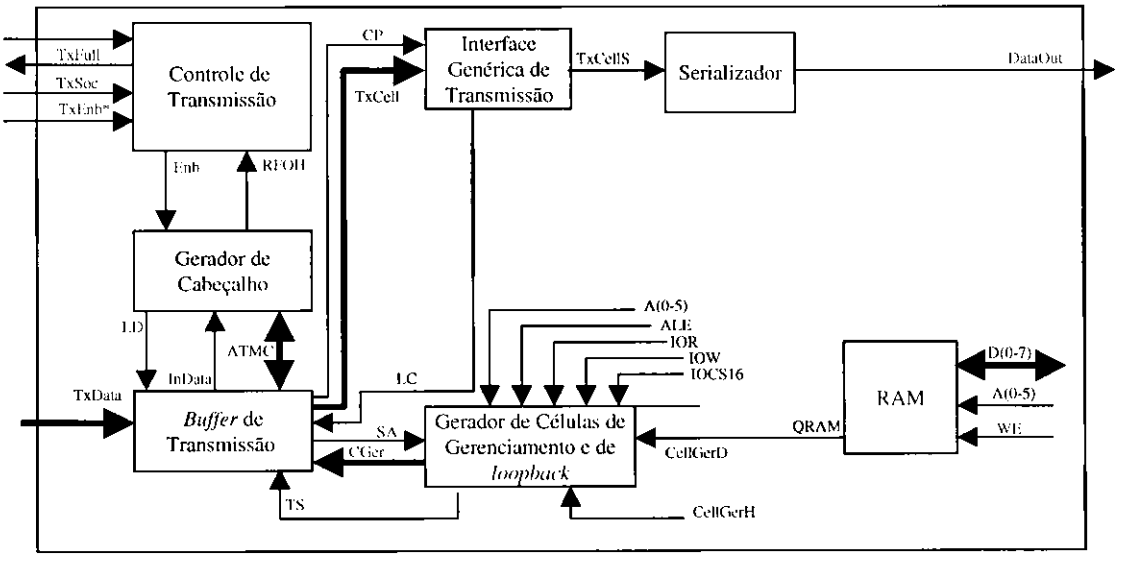


Figura 4.22: Interface Genérica de Transmissão.

4.3.3 Interface Genérica de Recepção

O bloco Interface Genérica de Recepção, ilustrado na Figura 4.23, será encarregado de receber quadros ou células vindas do meio físico, delimitar as células e entregar estas aos blocos Detectar Células de Gerenciamento e Verificar Cabeçalho. Caso for detectado que a célula é de gerenciamento OAM de nível Físico, ou vem com erro no cabeçalho, o *Buffer* deste bloco é esvaziado; caso contrário (i.e., cabeçalho sem erro: ErrFree=1, e a célula não for de gerenciamento CD=0) então o conteúdo do *Buffer* (a célula) é transmitido para o MCOA quando o sinal RxReady estiver em nível alto (i.e., RxReady=1). A Figura 4.23 mostra como o bloco Interface Genérica de Recepção está ligado com os outros elementos do sistema.

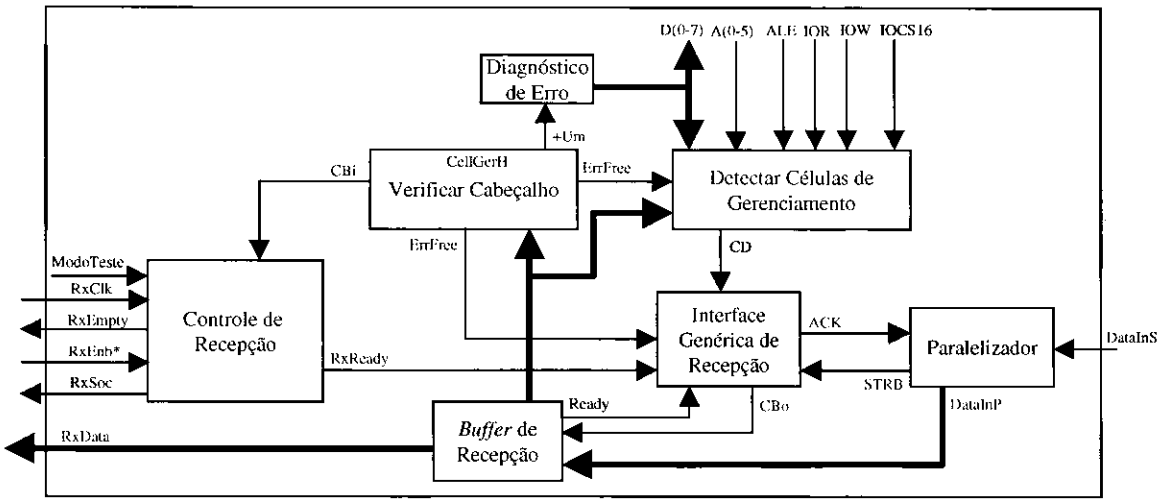


Figura 4.23: Interface Genérica de Recepção.

4.4 Resumo

Apresentamos no presente Capítulo, a arquitetura flexível e modular da Interface de Linha Física do Comutador ATM (COMATM). Propusemos também uma arquitetura mais genérica que pode servir de referência para implementação de interfaces de linha física para comutadores ATM usando os mais variados sistemas de transmissão.

No caso específico do COMATM utilizamos o par trançado como meio de transmissão por ele ser o mais utilizado nas redes locais. Com a proposta feita da arquitetura de uma ILF genérica poderemos futuramente adaptar o COMATM a uma outra tecnologia emergente ou dominante. Veremos no Capítulo 5 quais foram as estratégias de implementação que nos levaram à concepção de um circuito integrado para a nossa interface e quais foram os resultados obtidos.

Capítulo

5

Estratégia de Implementação , Testes e Simulação

5.1 Introdução

A implementação de uma arquitetura como a da ILF exigiu um estudo prévio para determinar entre as várias tecnologias, ferramentas e linguagens, as que melhor se adequam não somente à arquitetura proposta mas também às especificações globais do projeto COMATM.

A implementação envolveu duas fases, *hardware* e *software.*

A fase de implementação do *hardware* corresponde à implementação do sistema eletrônico usando uma linguagem de descrição do circuito integrado como VHDL, AHDL etc. Na fase da implementação do *software*, desenvolveu-se uma aplicação para fazer interface entre o usuário e o dispositivo físico. Este *software*, no caso do COMATM, é um *driver* permitindo ao sistema de gerência do comutador administrar a parte da camada Física do COMATM, efetuando tarefas como testes de meio físico (através de células de *loopback* ou de gerenciamento) ou da interface remota de comunicação.

A implementação do *hardware* requereu a escolha de uma tecnologia, de uma linguagem e de ferramenta(s) de implementação. Na Figura 5.1 ilustramos as diferentes alternativas de projetos para se chegar ao dispositivo final[23].

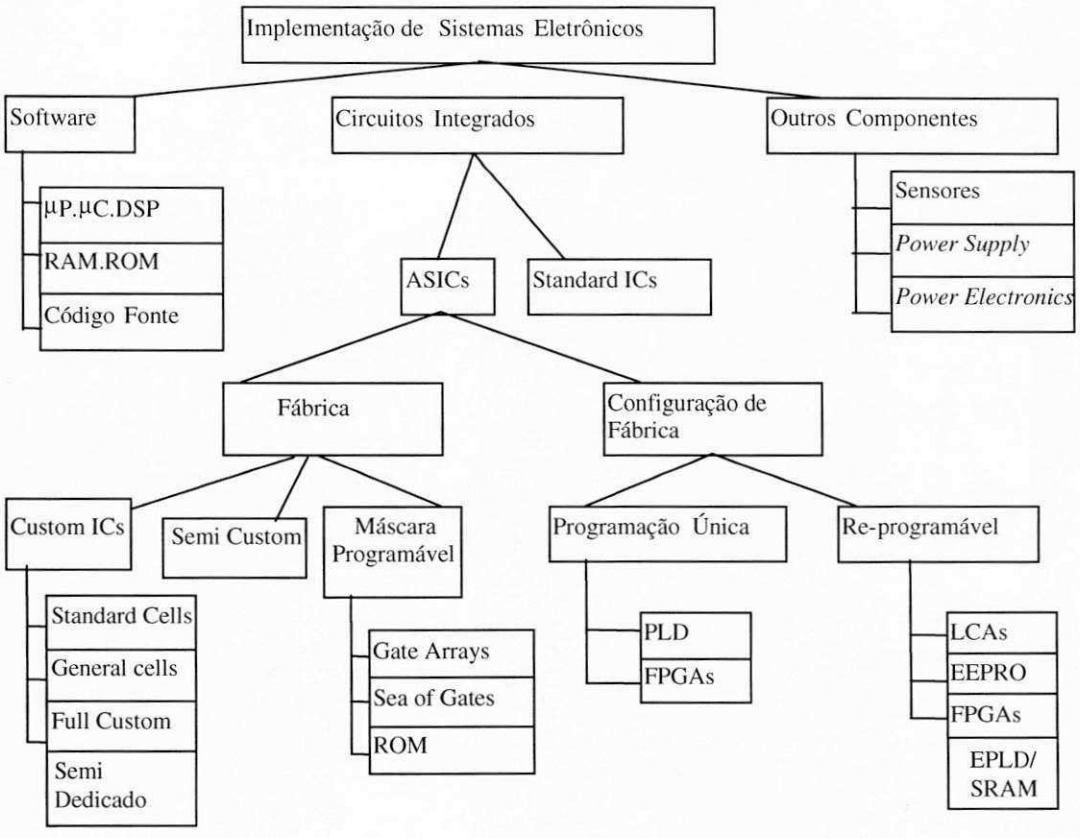


Figura 5.1: Alternativas de Projetos de Circuitos.

5.2 Implementação do *Driver* da Placa

O sistema operacional Linux (plataforma do COMATM) é um *clone* do Unix e no seu funcionamento os dispositivos são apresentados (ou melhor comunicam-se) aos programas como arquivos "especiais". Portanto, estes dispositivos implementam uma semântica de arquivo dentro do *kernel* (o núcleo do sistema operacional). Nós apresentamos no Anexo A como os arquivos de forma genérica são tratados pelo Linux.

5.3 Estratégia de implementação em *Hardware*

Na implementação da ILF várias considerações foram feitas na escolha das ferramentas adequadas a serem utilizadas para se chegar ao melhor circuito. Para o projeto COMATM, tínhamos algumas restrições a serem consideradas, sendo elas: o

tempo de desenvolvimento e o desempenho do circuito. Isso nos levou a traçar uma estratégia de implementação que consistia em usar duas linguagens distintas: VHDL e AHDL.

5.3.1 Justificativa para Escolha da Linguagem VHDL

A linguagem VHDL (VHSIC (*Very High Speed Integrated Circuits) Hardware Description Language)* está se tornando muito popular como sendo o meio para capturar circuitos eletrônicos complexos para simulação e síntese. Circuitos digitais capturados por VHDL podem ser facilmente simulados, podem ser sintetizados em várias tecnologias e posteriormente modificados e reutilizados.

VHDL é uma linguagem de programação que foi projetada e otimizada para descrever o comportamento de circuitos e sistemas digitais [24], combinando as seguintes características:

aa) Uma linguagem de modelamento de simulação;

bb) Uma linguagem de codificação do circuito;

cc) Uma linguagem de teste;

dd) Uma linguagem de *Netlist;*

ee) Uma linguagem padrão.

Utilizamos VHDL para o nosso projeto porque: ela aumenta dramaticamente o desempenho e a produtividade; ela se adequa muito bem a uma abordagem de projeto *top-down*, que foi a abordagem que utilizamos; supre algumas insuficiências presentes em linguagem como AHDL (*Altera Hardware Description Language*) que, por exemplo, não provê um meio de se implementar um *loop* infinito, entre outras. Como a ILF é um componente do COMATM, utilizamos a VHDL para implementar os blocos da ILF que fazem interface com outros componentes do COMATM como ABSE e,

finalmente, para poder aproveitar a poderosas ferramentas de *design* (como a CADENCE) para simulação e verificação.

5.3.2 Justificativa para Escolha de FPGA (*Field-Programmable Gate Arrays*)

A tecnologia VLSI (*Very Large Scale Integration*) abriu as portas para implementação de potentes circuitos digitais a um baixo custo. Tornou-se possível fabricar *chips* com mais de um milhão de transistores. Tais *chips* são projetados usando uma abordagem *full custom*, onde todas as partes do circuitos VLSI são minuciosamente elaboradas para respeitar um conjunto de requisitos específicos. Abordagens usando *Standard Cell* e *Mask-Programmed Gate Arrays* (MPGAs) proveram meios mais simples para se projetar e fabricar Circuitos Integrados para Aplicações Especificas (ASICs - *Application-Specific Integrated Circuits*) [25].

A utilização dessas tecnologias requer um esforço de projeto e fabricação imensos, e meses de dedicação, desde o início do projeto até a obtenção do *chip*. Na indústria microeletrônica, é vital chegar ao mercado com novos produtos no mínimo de tempo possível, portanto, reduzir o tempo de projeto e de produção, é essencial para o sucesso. Além disso é importante que os riscos financeiros para o desenvolvimento de um produto novo seja limitado. Os FPGAs emergiram como a uma solução para responder às exigências da indústria. Os FPGAs constituem um meio atrativo de implementação de circuitos lógicos dando resultados quase que imediatos, com custos pequenos. Os FPGAs oferecem uma solução viável para personalizar VLSIs.

Um dispositivo FPGA é um dispositivo no qual a estrutura lógica final pode ser diretamente configurada pelo usuário final. Um FPGA consiste de um *array* de elementos que podem ser interconectados de modo genérico. As interconexões podem ser programáveis pelo usuário. Os FPGAs podem ser utilizados para aplicações que usam dispositivos lógicos programáveis (PLD - *Programmable Logical Device*), dispositivos

com integração de pequena e larga escala (respectivamente SSI - *Small Scale Integration* e VLSI – Very Large Scale Integration). Os FPGAs são geralmente muito eficientes para aplicações de prototipagem. O baixo custo de implementação e o pouco tempo necessários para realizar fisicamente um projeto com FPGAs, provê grandes vantagens em relação a abordagens tradicionais de prototipação de *hardware*. Versões iniciais de prototipagem podem ser implementadas rapidamente com FPGAs, e alterações subsequentes no protótipo podem ser feitas facilmente a um custo baixo.

O ponto inicial do processo de implementação com FPGAs é a entrada da descrição lógica do circuito a ser projetado. Esta etapa envolve tipicamente a utilização de um programa de captura esquemática, ou entrar com uma descrição VHDL, AHDL etc., ou especificar expressões booleanas. A descrição do circuito é geralmente traduzida em uma forma padrão como expressões booleanas, por exemplo. As expressões booleanas são então ser processadas por uma ferramenta de otimização lógica. O objetivo é modificar estas expressões e otimizar a área ou velocidade do circuito final.

As expressões booleanas otimizadas devem ser a seguir transformadas em um circuito de blocos lógicos FPGA. Isso é feito através de um programa de mapeamento de tecnologia.

Tendo mapeado o circuito em blocos lógicos, é necessário decidir onde colocar cada bloco no *array de* FPGA. Um programa de alocação é utilizado para resolver este problema. A etapa final no sistema é feita por um programa de roteamento que aloca seguimentos de fios e escolhe conectores (*switches*) para estabelecer conexões entre os blocos lógicos. A Figura 5.2 mostra um fluxograma de um projeto de circuito usando uma ferramenta CAD para FPGAs.

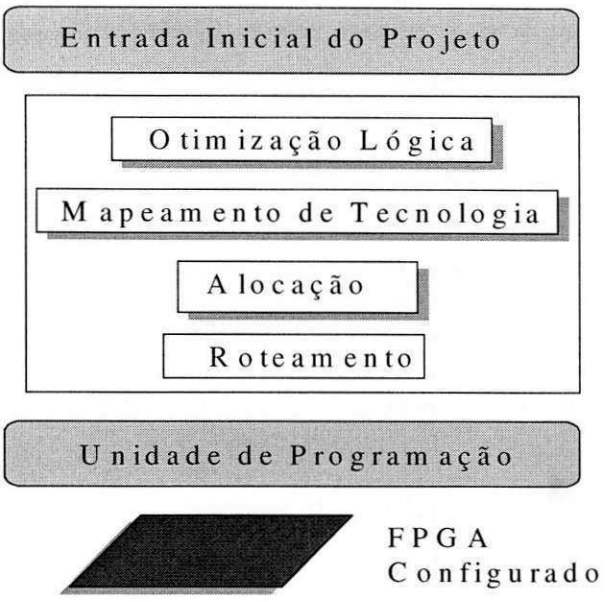


Figura 5.2: Fluxograma de um projeto de circuito usando uma ferramenta CAD para FPGAs.

5.4 Simulação

Para verificar a funcionalidade da ILF, um conjunto de simulações foi feito e os resultados dessas simulações são abordados neste Capítulo.

Utilizamos duas linguagens de implementação da ILF:

ff) a VHDL para poder aproveitar as características de desempenho e de otimização;

gg) a AHDL (Altera *Hardware Description Language*), pela rapidez em se obter resultados a curto prazo, o baixo custo de implementação e as grandes vantagens de prototipação.

5.4.1 Simulação da Entrada do Bloco de Controle de Transmissão

Como já sabemos a partir dos Capítulos 3 e 4, as funções do módulo de Transmissão são: entrada de dados segundo a interface UTOPIA, a geração do HEC para completar o cabeçalho de célula e a transmissão da célula. As figuras 5.3 e 5.4

ilustram o diagrama de simulação do Módulo de Transmissão. txclk, txdata e txsoc são sinais de transmissão segundo a interface UTOPIA.

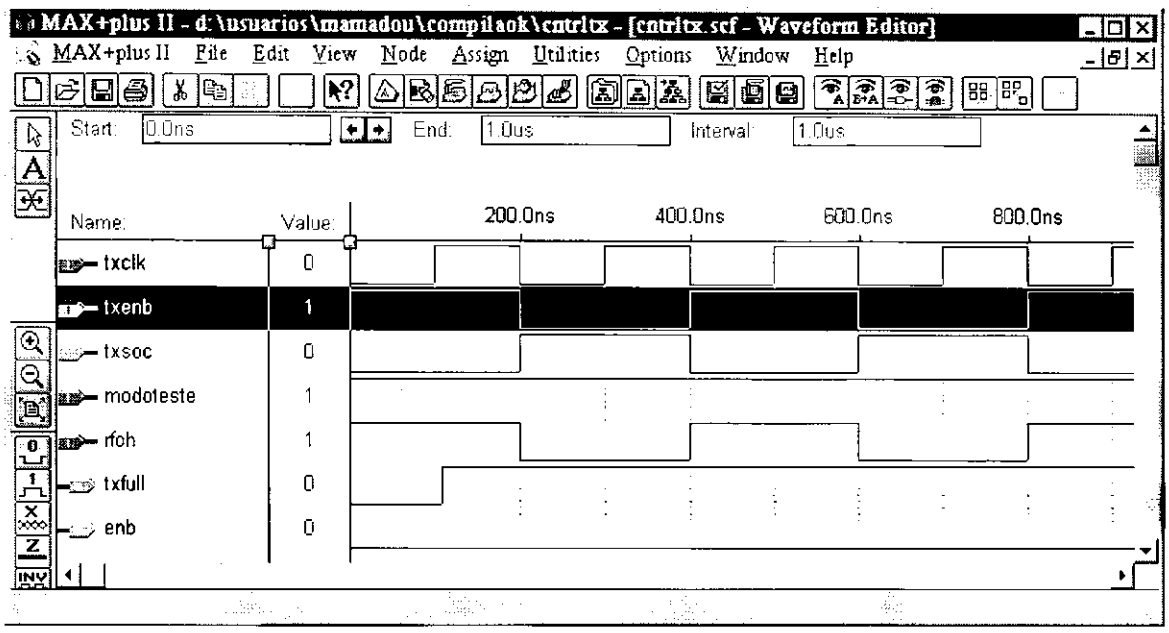


Figura 5.3: Diagrama de Simulação do módulo de Transmissão em modo de Teste.

Quando o sinal modoteste estiver com nível lógico alto, a ILF opera em modo teste como ilustrado na Figura 5.3. Neste estado, (i.e. quando o sinal modoteste = 1) o sinal Txfull* (sinal Utopia) é desativado, portanto txfull = 0. Assim não haverá transmissão de células da camada ATM para a camada Física. O sinal Enb pode ser habilitado (conforme ilustrado na figura 5.4), como aconteceu após 300 ns, correspondendo a segunda borda de subida do *clock*. O sinal RFOH é habilitado (RFOH =1) cada vez que o Gerador de Cabeçalho estiver pronto para calcular um outro HEC, este tempo correspondendo a 200 ns como mostrado na Figura 5.4.

Quando o sinal modoteste estiver com nível lógico baixo, a ILF opera em modo normal. Este estado é ilustra na Figura 5.4. Portanto, quando o Gerador de cabeçalho estiver pronto (i.e. quando RFOH =1), então txfull* é ativado (i.e., txfull = 1 ou melhor txfull* = 0)

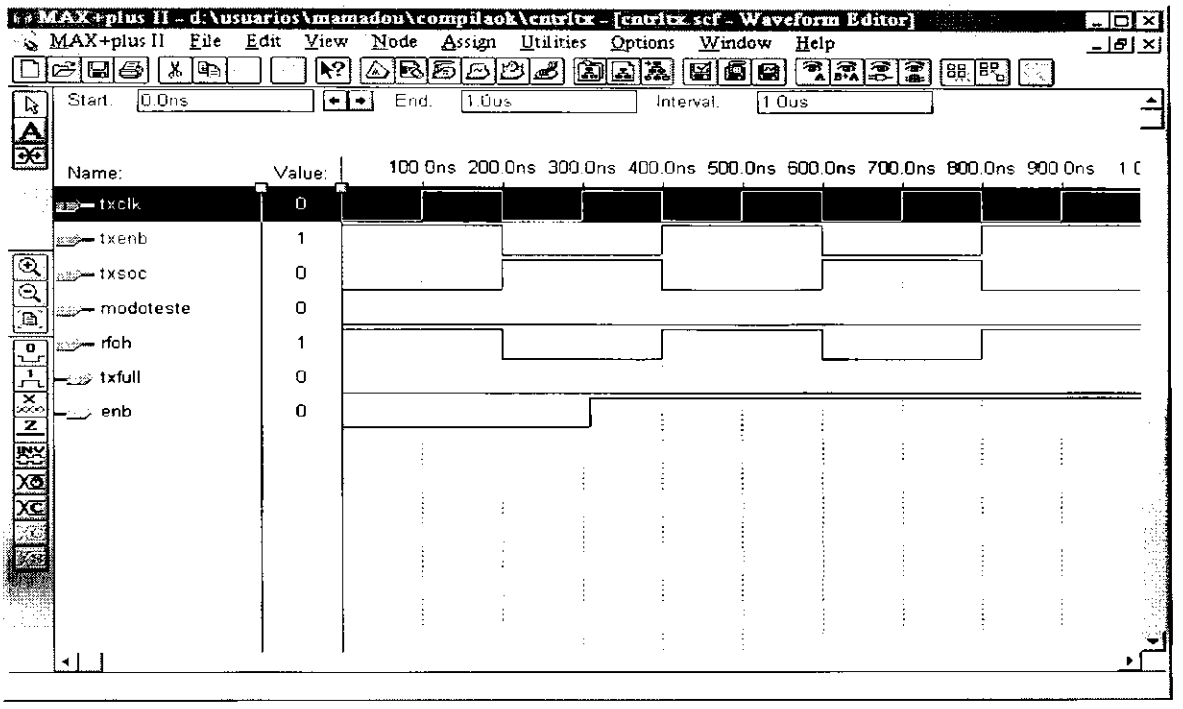


Figura 5.4: Diagrama de Simulação do módulo de Transmissão em Modo Normal.

5.4.2 Simulação da Saída do Módulo de Recepção

A Figura 5.5 mostra o diagrama de simulação da saída do módulo de Recepção. O sinal CelulaDispo sendo igual a 1 significa que a célula inteira está pronta a ser transmitida. Consequentemente faz com que EC se torne igual a 0, permitindo que a célula seja transferida para a saída do *buffer*. No diagrama da Figura 5.5, podemos observar que o conteúdo de reg4_[7..0] correspondendo à célula armazenada no *buffer* é transmitida para o MCOA através do sinal de UTOPIA RxData. Como TransTipo = "00H" então a célula a ser transmitida vem da interface de transmissão TAXI. Assim, a célula vindo da interface TAXI é armazenada no *buffer* se o sinal CBo (CBo =0) não for ativado, conforme ilustrado na Figura 5.5. Caso o sinal CBo seja ativado (cabeçalho corrompido), então o conteúdo do *buffer* é zerado descartando a célula recebida. Este estado está ilustrado na Figura 5.6

Quando o sinal TransTipo for igual a "11H", o registrador armazena as células vindas da interface SDH/SONET e depois, as transmite para a camada ATM. Esta transmissão inicia-se somente quando o sinal TransTipo for igual a 11 e o bloco estiver pronto para receber a célula ativando o sinal EC (EC =1) e CBo desativado (CBo =0). A Figura 5.7. ilustra a operação de transmissão de um octeto vindo da interface SDH.

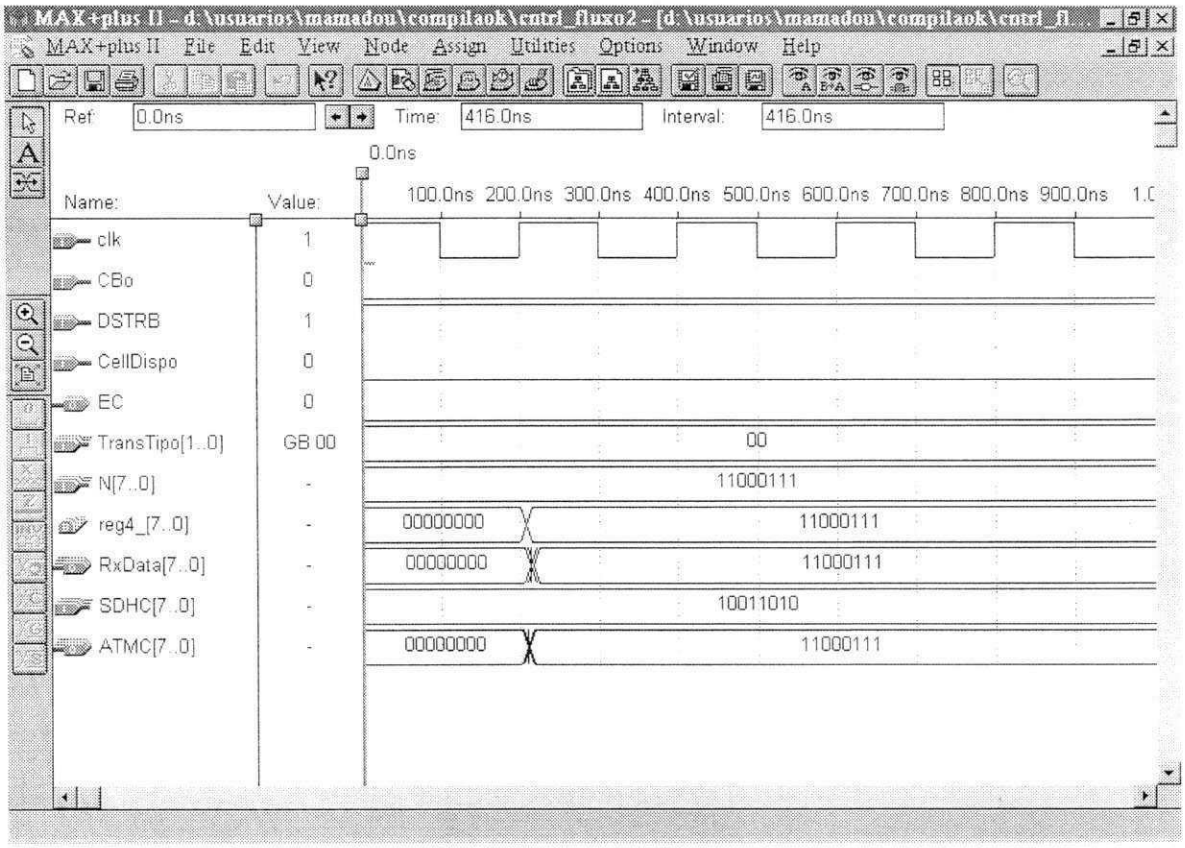


Figura 5.5: Diagrama d saída do módulo de Recepção das Células Vindas da Interface TAXI(CBo = 0)

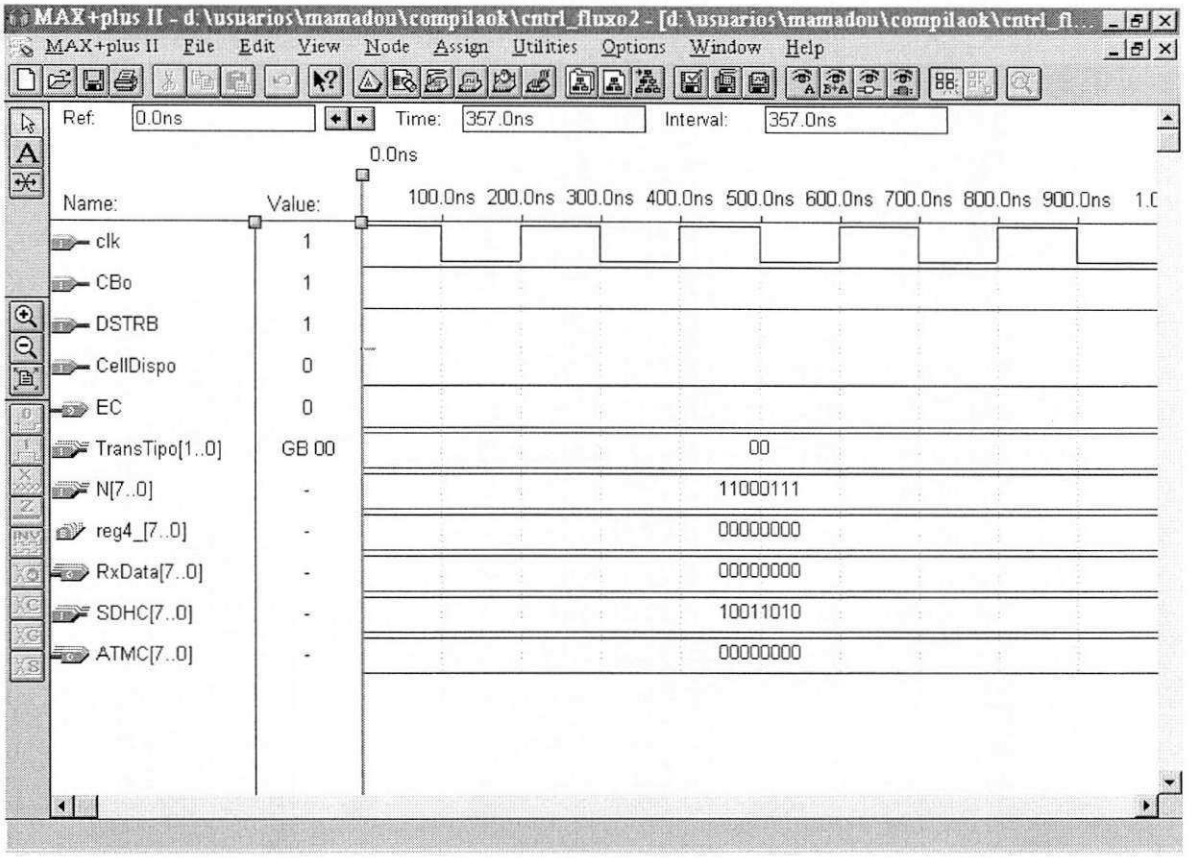


Figura 5.6: Diagrama da saída do módulo de Recepção das Células Vindas da Interface TAXI(CBo = 1)

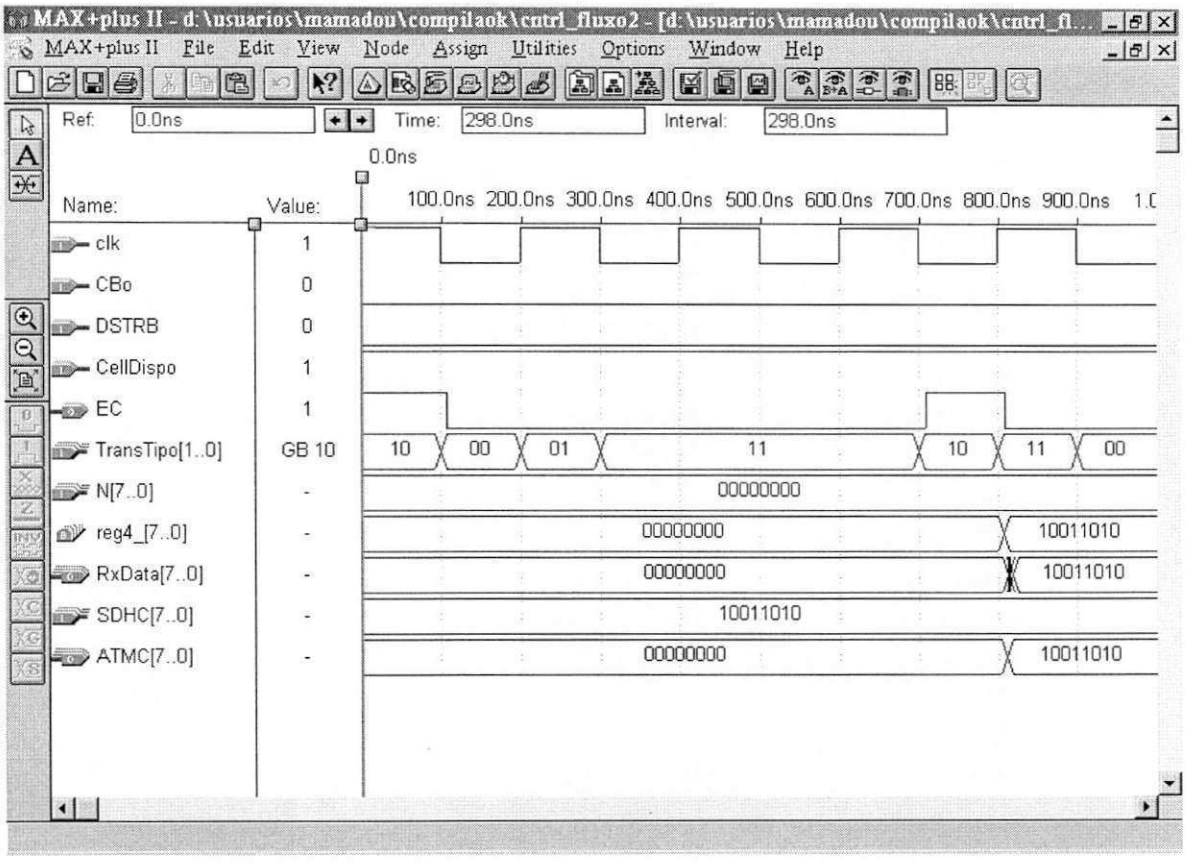


Figura 5.7: Diagrama e Transmissão de Células vindo da Interface SDH.

Quando o sinal CBo for ativado, o *buffer* é zerado e a célula descartadas, conforme ilustrado na Figura 5.8.

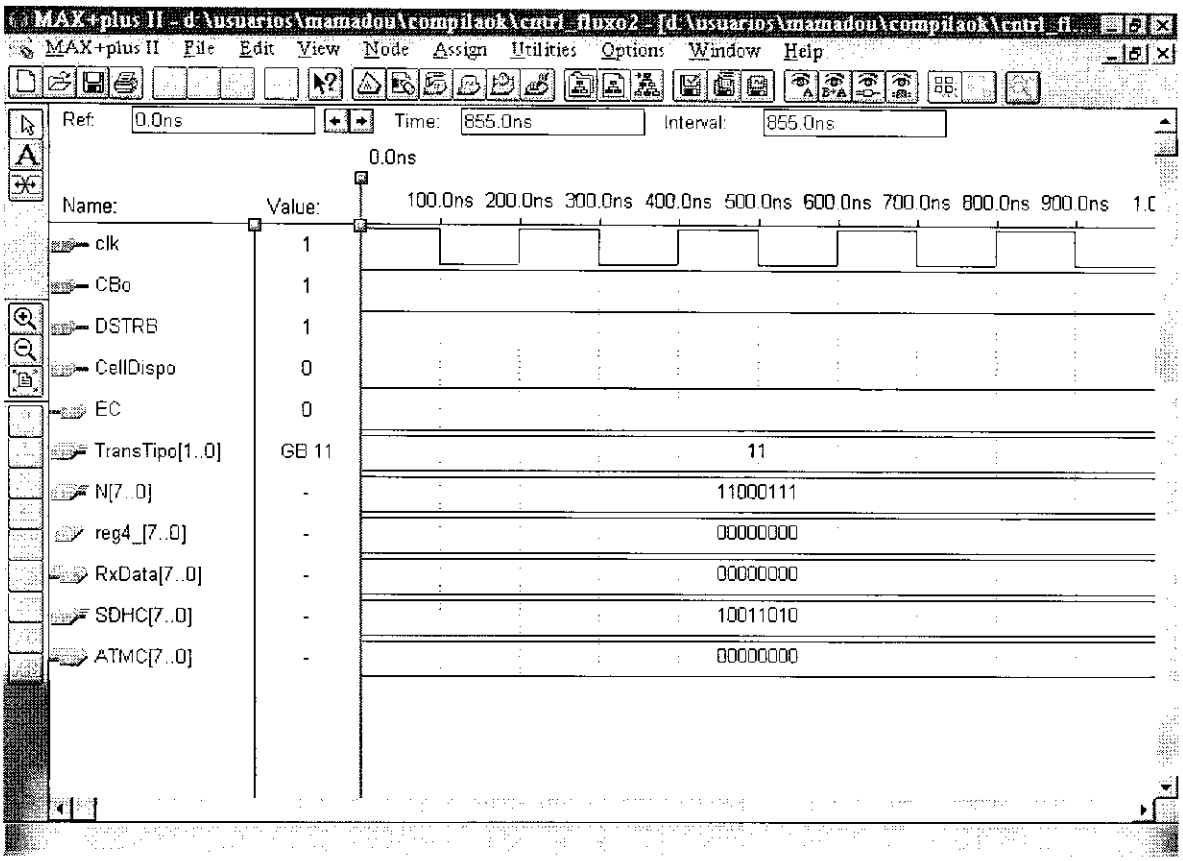


Figura 5.8: Diagrama e Transmissão de Células vindo da Interface SDH.(CBo=1)

5.4.3 Simulação da Saída do Bloco de Controle de Recepção

As figuras 5.9, e 5.10 mostram como sinais no módulo de Recepção da ILF se comportam para transmitir para o MCOA células vindas do meio físico ou descartá-las quando houver erro no cabeçalho. A Tabela 5.1 mostra o comportamento dos sinais de saída para alguns vetores de entrada no bloco de Controle de Recepção.

Rxclk	1	1	1	X	1
RxEnb	1	0	0	X	1
Modoteste	0	0	0	1	0
CBi	0	1	0	X	1
RxSoc**	0	1	0	0	0
RxReady**	0	1	1	1	0
RxEmpty**	0	0	1	0	0
CBo**	0	1	0	X	1

X=Qualquer valor binário 0 ou 1

** Sinais de saída do bloco

Tabela 5.1: Alguns Vetores de Simulação e os Respectivos Resultados Esperados.

Na figura 5.9, observamos que quando RxEnb for igual a 1 os sinais RxSoc, RxReady e RxEmpty também ficam desativados mesmo tendo os sinais CBi e CBo iguais a 0. Depois de 200 ns quando RxEnb for ativado (RxEnb = 0) significando que o MCOA está pronto para receber uma nova célula, os sinais RxSoc e RxReady ficam ativados alguns nanosegundos depois, para sinalizar respectivamente a presença do início de uma célula e avisar ao bloco verificar cabeçalho que pode efetuar uma nova operação de verificação do HEC. Ao receber o *status* do cabeçalho (não corrompido) aos 400 ns através do sinal CBo, o sinal RxEmpty é ativado para permitir a transferência da célula. Esta operação está ilustrada na Figura 5.9.

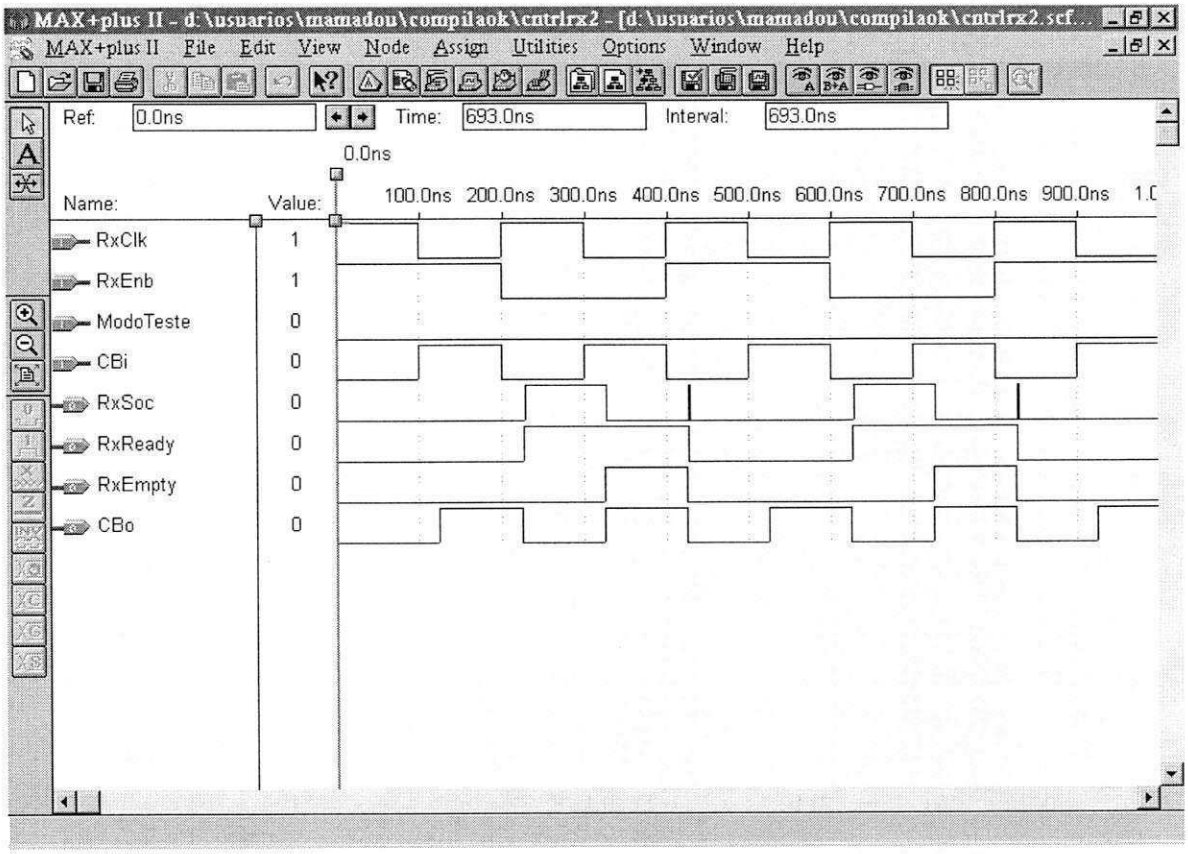


Figura 5.9: Diagrama de Fluxo do Bloco de Recepção.

A Figura 5.10 ilustra o estado da transmissão quando a ILF está em modo de teste (sinal ModoTeste = 1). Neste caso não há nenhuma transmissão de célula da ILF para o MCOA; isto se ilustra com os sinais RxSoc, RxReady e RxEmpty desativados. O RxEmpty é sempre ativado em nível lógico baixo conforme está descrito no anexo B.

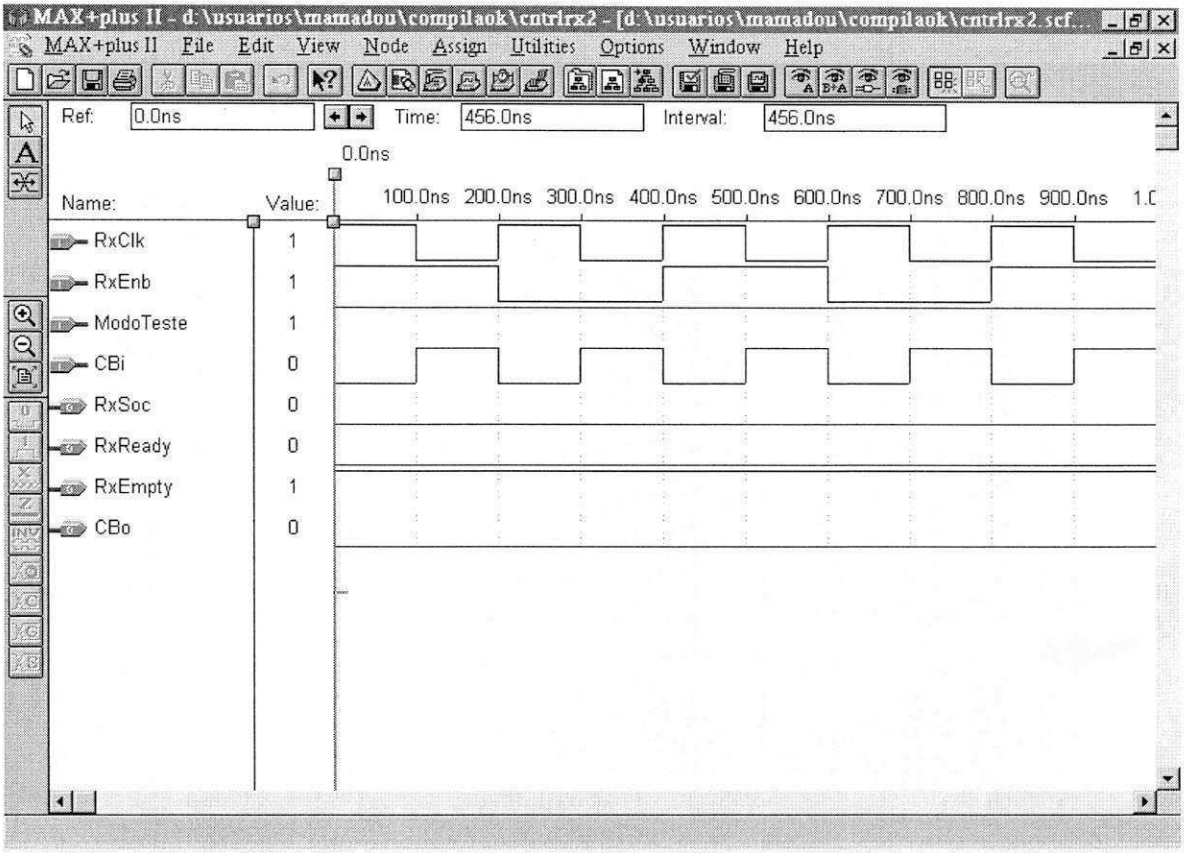


Figura 5.10: Diagrama de Fluxo do Bloco de Recepção em Modo. Teste

5.5 Resultados

Neste Capítulo mostramos as tecnologias e ferramentas adotadas para implementação da ILF. A utilização de duas linguagens de descrição de circuitos(VHDL e AHDL) foi motivada pela preocupação em se obter um protótipo de bom desempenho que atendesse aos requisitos do sistema. Cada bloco da ILF foi implementado individualmente e os resultados são mostrados no Anexo II. O chip da ILF sintetizado pela ferramenta da Altera, MaxPlus II versão 8.0 está ilustrado na

Figura 5.12. Sua configuração é apresentada na Tabela 5.2. O componente utilizado é um EPF10K100GC503-3 da família do Flex 10K

.

A Figura 5.13 mostra o *layout* da placa que contém além do *chip* da ILF as interface TAXI, SDH/SONET e ISA..

chip_ilf EPF10K100GC503-3		
Total dedicated input pins used:	*6/6*	*(100%)*
Total I/O pins used:	*85/400*	*(21%)*
Total logic cells used:	*100/4992*	*(2%)*
Total embedded cells used:	*0/96*	*(100%)*
Total EABs used:		*0*
Total input pins required:		*56*
Total input registers required:		*0*
Total output pins required:		*35*
Total output registers required:		*0*
Total buried I/O cell registers required:		*0*
Total bidirectional pins required:		*0*
Total reserved pins required		*0*
Total logic cells required:		*100*
Total flipflops required:		*32*
Total packed registers required:		*0*
Total logic cells in carry chains:		*0*
Total number of carry chains:		*0*
Total logic cells in cascade chains:		*0*
Total number of cascade chains:		*0*
Total single-pin Clock Enables required:		*0*
Total single-pin Output Enables required:		*0*

Tabela 5.2: Configuração do chip da ILF.

O resultado completo da síntese do *chip* da ILF, bem como as informações sobre número de pinos, memória e *chip* utilizado estão ilustrados na Tabela 5.2. Por utilizarmos uma ferramenta com licença acadêmica, não foi possível escolher um melhor componente para a nossa aplicação.

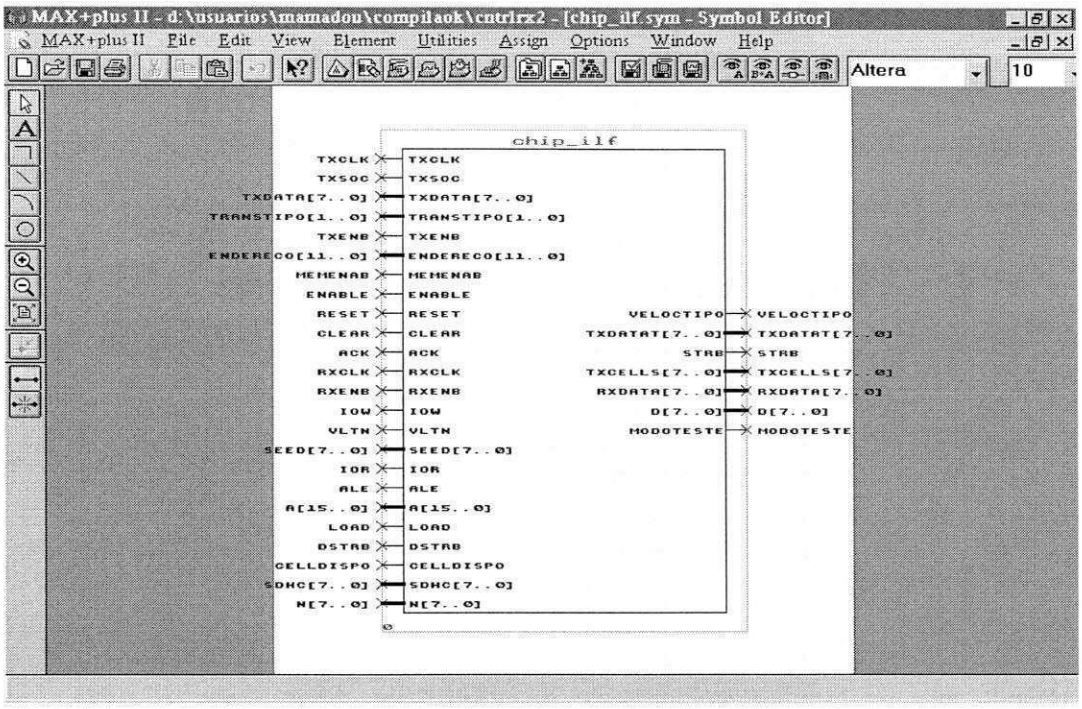


Figura 5.11: A Interface de Linha Física Gerada pelo MaxPlus II da Altera.

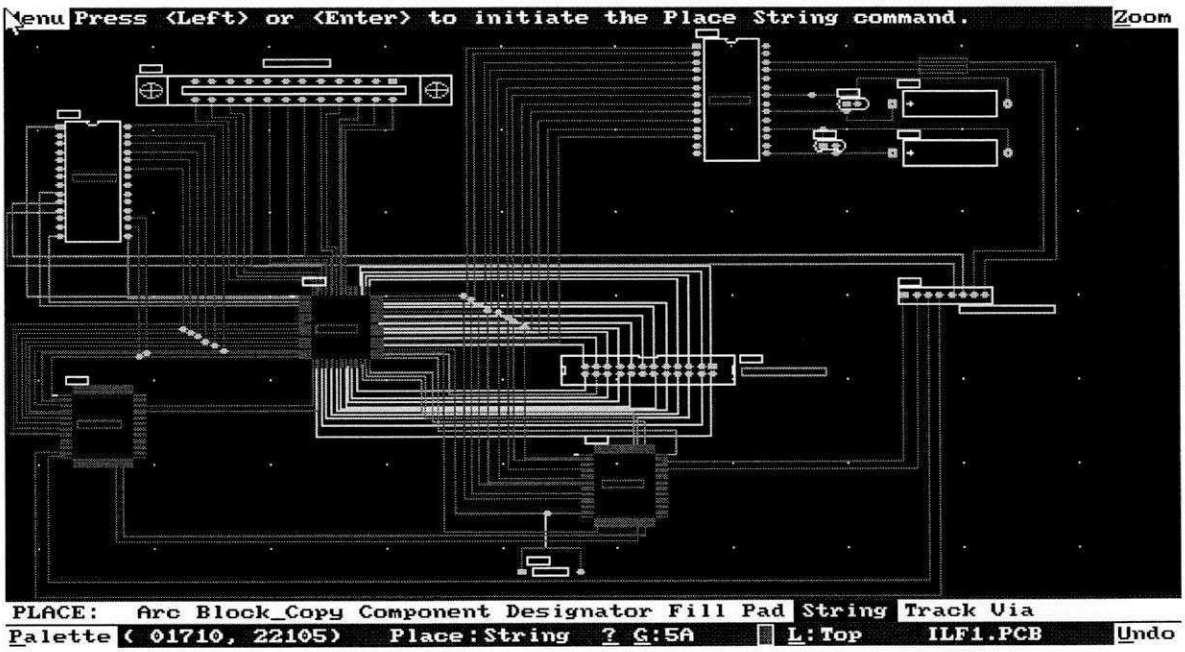


Figura 5.12: Layout da Placa da ILF.

Capítulo

6

Conclusões

Apresentamos neste Capítulo algumas considerações finais de nosso trabalho com enfoque sobre a nossa contribuição, as dificuldades encontradas na especificação e implementação da Interface de Linha Física (ILF) de modo geral, e da ILF para o COMATM em particular.

Neste trabalho apresentamos uma arquitetura flexível e modular de uma Interface de Linha Física para o Comutador ATM (COMATM). A ILF é uma placa que implementa as funcionalidades da camada Física do modelo de referência RDSI/FL. A ILF é flexível no sentido em que ela pode ser adaptada para trabalhar a taxas superiores a 100 Mbps, como por exemplo, taxas de até 622 Mbps. Como os FPGAs nos permitem um desenvolvimento de protótipos em um curto tempo e um custo muito baixo, escolhemos essa tecnologia para a implementação da ILF. Utilizamos ainda uma ferramenta de descrição VHDL para implementar alguns blocos da ILF no sentido de se aumentar o desempenho do circuito . A ferramenta utilizada foi o MAXPLUS II da Altera, a mais acessível no mercado em termo de custo.

No caso específico do COMATM utilizamos o par trançado como meio de transmissão (por ele ser o mais utilizado nas redes locais) e a interface de transmissão TAXI. Com a arquitetura proposta de uma ILF genérica, pode-se futuramente adaptar o COMATM a uma outra tecnologia, emergente ou predominante, para aumentar a sua taxa de transferência ou o seu desempenho.

Neste trabalho, nós conseguimos combinar várias tecnologias num único projeto. Trabalhamos com a linguagem VHDL usando CADENCE, e AHDL usando o MAXPLUS II da Altera.

Para o desenvolvimento de uma interface eficiente entre o meio físico e o comutador necessitou-se desenvolver uma Interface de Linha Física cuja descrição foi feita neste documento. Recentemente, fabricantes de comutadores começaram a fornecer interfaces ATM com par trançado não blindado (UTP) de categoria 5 a 155 Mbps para diminuir os custos dos produtos ATM. Com isso diminui-se o custo relativamente alto da operação de conversão eletro-óptica, tanto nas portas dos comutadores ATM quanto nas placas de adaptador ATM nos terminais dos usuários. Isto tende a diminuir bastante os custos dos produtos ATM.

Escolhemos os circuitos integrados AM7968 e AM7969 da AMD para interface entre a ILF e o meio físico, por eles atenderem às nossas especificações e as do Fórum ATM.

No final do nosso trabalho conseguimos chegar ao *chip* principal da ILF para o COMATM que era o objetivo inicialmente fixado como foi relatado no Capítulo 1.

O desempenho da ILF, em termo de velocidade de transmissão e recepção das células ATM, está fortemente relacionado com a maneira com a qual ela efetua a geração e verificação do quinto octeto da célula, o HEC. Várias propostas existem para resolver tal problema. A mais simples e menos eficiente em termos de tempo para obtenção/verificação do HEC, tem sido a da divisão polinomial de forma serial (conforme discutimos no Capítulo 4). A outra alternativa que consiste em se utilizar um algoritmo tratando as células e efetuando a operação de forma paralela é mais eficiente permitindo taxas de transferência satisfatórias para qualquer velocidade empregada. Destes algoritmos foi destacado o de Rocksoft que é um algoritmo genérico que adaptamos para os nossos propósitos. O resultado obtido foi um circuito HEC funcionando a uma taxa superior a 100 MHz e 800 Mbit/s.

Por tratar-se de uma ILF genérica, necessitou-se de algum mecanismo para configurá-la segundo o meio de transmissão empregada. Este mecanismo consiste na programação de uma EPROM que informa ao sistema todos os parâmetros necessários para a sua inicialização e operação.

A ILF se comunica com o sistema hospedeiro (interface entre a ILF e o sistema de gerência do sistema) através do barramento ISA. Escrevemos um *driver* em linguagem C no ambiente Linux para permitir que o sistema de gerência do COMATM possa enviar e receber células de *gerencia*mento OAM de nível Físico em uma posição de memória previamente definida (300H a 31FH), como também testar tanto o circuito quanto o meio de comunicação.

O resultado obtido no final deste trabalho é uma placa contendo um circuito sintetizado e simulado usando ferramentas do MAXPLUS II da Altera e a ferramenta da CADENCE.

Como contribuição mais significativa propusemos um roteiro para projetos e implementação de interfaces físicas para comutadores ATM. Mostramos como se pode combinar diferentes tecnologias para se chegar a um circuito que combina desempenho de ASICs e requisitos de prototipagem. Esta síntese poderá ser futuramente utilizada em outros projetos para se obter resultados rápidos e eficientes. O outro ponto a destacar neste trabalho como contribuição é a ILF genérica proposta para interfaces físicas que usam UTOPIA como interface com a camada ATM, e qualquer sistema de transmissão como interface com o meio de transmissão. Isso permitirá aos projetistas ganharem bastante tempo na fase da elaboração da arquitetura do circuito.

Por fim, consideramos este trabalho uma referência para interessados na área de redes de alta velocidade, de modo geral, e de circuitos para interface física de comutadores ATM, em particular, pois este trabalho contém um levantamento bibliográfico bastante atual sobre as novas tecnologias, as tendências e as padronizações nesta área.

Durante o desenvolvimento deste trabalho a principal dificuldade que encontramos foi a utilização da ferramenta MAXPLUS II da Altera por duas razões:

hh) a quantidade muito limitada da bibliografia, com uma quantidade reduzida de livros, artigos e documentos técnicos.

ii) algumas limitações inerentes à linguagem AHDL para implementar algumas funções como um *loop* infinito ou circuitos combinacionais descritos em VHDL (por exemplo o uso de ***wait on ..*** ou ***wait for ...***). O jeito que encontramos de contornar estas dificuldades foi o de utilizar máquinas de estados finitos.

Destacamos a seguir trabalhos que poderão ser feitos no sentido de melhorar a ILF para o COMATM ou mesmo desenvolver uma nova ILF:

jj) implementação de uma ILF para SDH/SONET. Com a utilização de dois sistemas de transmissão (TAXI, SDH/SONET), a implementação da ILF aumentou em complexidade. Com uma ILF exclusivamente para SDH, teremos um circuito simples de alto desempenho sem necessidade, por exemplo, de multiplexador para escolher o sistema de transmissão, nem de um mecanismo para escolher dentre os diversos parâmetros para a inicialização do *chip*;

kk) ILF usando uma interface de transmissão própria. A utilização de interfaces comerciais para interface com o meio de transmissão, diminuiu a complexidade de implementação e, portanto, conseguimos ter resultados significativos com um tempo menor. Entretanto, isso afetou tanto o custo final da ILF quanto o seu tamanho, tendo que acoplar o *chip* comercial na placa da ILF. Com uma interface de transmissão própria (i.e., integrado no *chip* principal da ILF), teremos um *chip* de menor tamanho, com custo bastante reduzido;

ll) e, finalmente, para obter taxas de transferências na ordem de Gbps, no sentido de fornecer uma placa para comutadores ATM com taxas de transferência das mesma ordem de grandeza, faz-se necessário a utilização de uma interface PCI em vez de ISA. Portanto, é de fundamental importância aumentar o *throughput* entre a

ILF e o sistema hospedeiro para não afetar o comutador caso ele venha a operar em taxas de Gbps.

Vale ressaltar por fim que o projeto da ILF para o COMATM foi desenvolvido dentro do projeto COMATM em colaboração com as universidades UFPB, UFPe e USP e com o apoio do CNPq/RAHE/PROTEM II.

Bibliografia

[1] Tanenbaum, A. S. Computer Networks. 3rd Edition . Prentice Hall - 1994

[2] Monteiro José, A. S. Rede Digital de Serviços Integrados de Faixa Larga (RDSI-FL). IX Escola de Computação - Recife - Agosto 1994.

[3] Soares, L. F. G., Lemos, G e Colcher, S. Redes de Computadores das LANs MANs e WANs às Redes ATM. Ed Campus - 1995

[4] The ATM Forum Technical Commitee. User-Network Interface (UNI) Specification Version 3.1 - 1994

[5] Hari Hansén. Connecction Management Functions of a Private Wireless ATM Network. Helsinki University of Tecnology – DEE, March 1996

[6] Yiwei, T.H e outros. Overview of Implementing ATM-Based Enterprise Local Area Network for Desktop Multimedia Computing. IEEE Communication, Magazine, Abril 1996

[7] Gigabit Ethernet Alliance. Gigabit Ethernet over Copper. www.gigabit-ethernet.org/technology/whitepapers/gige_11.97/papers97_toc.html. Novembro 1997.

[8] Giozza, W F, Zerrouk Belkacem. Arquitetura de Comutador ATM Usando O Roteador Rcube. Anais Do 12º SBRC, CURITIBA, 18-20 MAIO 1994.

[9] ITU-T draft Recommendation I.361, B-ISDN ATM Layer Specification, 1993

[10] ITU-T Recommendation Q.2931 "B-ISDN User-Network Interface Layer 3 Specificaton for Basic Connection Control", March 1994

[11] The ATM Forum Technical Commitee. UNI Signaling 4.0 Release 6 – 1996

[12] ITU-T I.363: B-ISDN ATM Adaptation Layer (AAL) Specification, Março 1993.

[13] ITU-T. Draft Recomendation I.432. B-ISDN User-Network Interface – Physical Layer Specification.

[14] ANSI T1.107. Digital Hierarchy: Format Specification, Agosto 1988.

[15]ANSI T1.105-1991 "Digital Hierarchy – Optical Interface Rates and Format Specifications (SONET).

[16] IBM Book Manager Book Server-Multi Level Transmit – 3 Levels (MLT-3) High Speed Networking, 1996.

[17] Silva, H S.; Lima, J. A. G, Monteiro Suruagy, Giozza W. F. Projeto COMATM - Arquitetura Básica e Descrição Funcional do Comutador ATM - Versão 3.0 - Relatório Técnico Rt-01.96 - Fevereiro de 1996.

[18] TAXIChip(TM) Integrated Circuit – Transparent Asynchronous Transmitter/Receiver Interface. Data Sheet and Technical Manual. Advanced Micro Devices. Setembro 1995.

[19] ATM Forum - Utopia ATM Physical Interface Specification Level 2, Version 1.0 USA - Junho 1996.

[20] Touré Mamadou; Giozza, W. F; Silva. H. S. Arquitetura Flexível de uma Interface de Linha Física para um Comutador ATM (COMATM). Anais Do 15º SBRC, SÃO CARLOS, Maio 97.

[21] Ross, N. W. A painless Guide to CRC Error Detection Algorithm. www.adelaide.edu.au Agosto 1993.

[22] AMCC ATM Products. SETI-S3005/SERI-S3006 SONET/SDH/ATM OC-3/OC-12 Transmitter and Receiver(http://www.amcc.com).

[23] Rosenstiel, W. *Hardware* Development of Embedded Systems. Brazil963.12.

[24] Zhang, Q. Design of na ATM to HIC/HS Interface Multilane Backpressure Flow Control in ATM. Master of thesis - Novembro 1997.

[25] Brown D. S, Robert J. F e outros. FPGAs Application Handbook. Kluwer Academic Publishers, 1993.

Anexo

A

Implementação de um *Device Drivers para Plataforma Linux*

O sistema operacional Linux (plataforma do COMATM) é um clone do Unix; e no seu funcionamento os dispositivos são apresentados (ou melhor comunicam-se) aos programas como arquivos "especiais". Portanto, estes dispositivos implementam uma semântica de arquivo dentro do *kernel* (o núcleo do sistema operacional). Por causa disto nós vamos mostrar nas seções a seguir como os arquivos de forma genérica são tratados pelo Linux antes de falar do *drivers* em si.

1.1 Características dos Arquivos do Linux

O cabeçalho genérico do sistema de arquivo, <linux/fs.h>, define várias estruturas para acessar arquivos. *super_block* contem a informação básica sobre cada sistema de arquivo, e *super_operation* é uma estrutura de apontadores para funções associadas com o superbloco do sistema de arquivo. E é através desta estrutura que podem ser acessadas *inode_operations* e *file_operations*, a última, definindo as funções que podem ser usadas para acessar arquivos. Em sistemas

de arquivos comuns, tem-se somente um conjunto de operações sobre arquivo para todos os arquivos no sistema de arquivo, mas eles (os sistemas de arquivos) não tentam definir nenhuma operação nos arquivos especiais de dispositivos. Em vez disto, esses dispositivos definem suas próprias funções de operação e registram sua própria estrutura de *file_operations* com o Sistema Virtual de Arquivo (VFS - *Virtual File System*).

1.1.1 VFS

Operações genéricas de sistema de arquivo são tratadas por um código genérico de sistema de arquivo, e somente quando operações de *filesystem-dependent*, ou um *device-dependent*, precisam ser efetuadas é que o código para essa sistema de arquivo ou dispositivo é efetivamente chamado. A função necessária é procurada na própria instância de uma das estruturas de **_operation* e chamada. O código VFS fica no subdiretório **/fs** da fonte do *kernel* do Linux, e o código de cada sistema de arquivos é guardado em subdiretórios do subdiretório **/fs**.

1.1.2 Arquivos Especiais e Sistemas de Arquivo

Todas versões do Unix tem arquivos especiais para dispositivos. Todavia, O VFS é tão flexível que não somente arquivos especais podem ser criados mas sistemas de arquivos também. O sistema operacional Linux tem um sistema de arquivos chamado de sistema de arquivo proc (**profs** -*proc filesystem*) que é essencialmente um sistema de arquivo. Estes arquivos são muito parecidos aos dispositivos de hardware, porque eles geram arquivos de dados que não são de

nenhum arquivo e os apresentam ao usuário na forma de arquivo. Eles são, dispositivos virtuais projetados para relatar o estado do *kernel*.

1.1.3 Operações sobre Arquivos

A ILF, para comunicar-se com o sistema hospedeiro, precisa efetuar operações de leitura e escrita no barramento ISA que é a interface de comunicação. Para efetuar tais operações precisa-se definir uma estrutura de *file_operations* da seguinte maneira:

```
struct file_operations {

    int (*read) ( struct inode *, struct file *, char *, int);

    int (*write) ( struct inode *, struct file *, char *, int);

    int (*mmap) ( struct inode *, struct file *, struct vm_area_struct *);

    int (*release) ( struct inode *, struct file *);

                }
```

Como o *kernel* do Linux é monolítico. Somente um controle de thread criado pela chamada de sistema fica ativo a qualquer momento, então o driver da ILF não vai precisar bloquear as suas estruturas de dados.

1.1.3.1 Interface do Kernel

A maneira mais fácil de se desenvolver um *device driver* é usando um módulo de *kernel* carregável **Erro! A origem da referência não foi**

encontrada.]. Quando forem bem escritos, esses módulos podem ser facilmente utilizados na sua forma normal como módulos carregáveis, ou "re-compilado" e ligado com o resto do *kernel*. O símbolo *MODULE* é definido cada vez que um módulo está sendo compilado. O símbolo *_KERNEL_* é sempre definido quando se está compilando o código do *kernel*, mesmo quando se está compilando somente os módulos.

Agora podemos apresentar a implementação do driver da ILF . Nós vamos compilar o código usando a seguinte sintaxe:

```
/* gcc -O -DMODULE -D_KERNEL -c ilf.c */
```

Todo código de *kernel* do Linux deverá ser otimizado porque ele requer algumas extensões do gcc que são ativadas somente com a otimização

```
#include <linux/config.h>
```

Qualquer *device driver* deve incluir *<linux/config.h>* antes de incluir qualquer outro arquivo.

Símbolos do *kernel* que foram exportados para os módulos devem Ter seus nome manuseados de mesma maneira que é feito em C++, assim as mudanças nas estruturas do *kernel* serão notificadas. Isso provoca a não carga do módulo, porque se uma estrutura de *kernel* é mudada, carregar um módulo antigo que foi compilado com uma versão diferente da estrutura poderá danificar a integridade do sistema.

```
#ifdef MODULE
#include <linux/module.h>
#include <linux/version.h>
#else
#define MOD_INC_USE_COUNT
#define MOD_DEC_USE_COUNT
#endif
#include <linux/types.h>
#include <linux/fs.h>
#include <linux/mm.h>          /* Para verificar a área */
#include <linux/errno.h>       /* Para -EBUSY */
#include <asm/segment.h>       /* Para put_user_byte */
```

No funcionamento da ILF o sistema de gerência do COMATM ou a camada ATM podem querer ler o conteúdo da posição de memória onde se encontra as células de gerenciamento OAM geradas pela *camada Física.* Para isso definimos uma operação de leitura da seguinte forma:

```
static int read_ilf (struct inode *, struct file * file, char buf *, int count, int left);
main()
{
if ( verify_area(VERIFY_WRITE, buf, count0 ==-EFAULT) return -EFAULT;
```

```
for (left = count; left = 0; left --)
        {
        put_user_byte(0,buf);
        buf ++;
        }
        return count;
}
```

Cada vez que a chamada de sistema *read* for utilizada para a ILF, *read_ilf* é chamada. A função *put_user_byte* coloca um *byte* na posição de memória anteriormente definida. *read_zero* vai retornar o número de *byte*s que foi realmente escrito no *buffer* de leitura.

Antes de escrever no *buffer*, nós verificamos se é legal (i.e. não haverá violação) escrever no *buffer* inteiro, usando a função *verify_area*(). Isso nos evita gerar uma falha no *kernel*. A função de escrita na memória é declarada da seguinte maneira:

```
static int write_ilf(struct inode * inode, struct file * file, char buf * , int count) return
count;
```

Cada vez a chamada de sistema write for chamada para a ILF, write_ilf é chamada. A função *write_zero* vai retornar o número de *byte*s que foi colocado na memória. *write_ilf* ignora suas entradas e retorna o resultado bem sucedido de uma operação de escrita.

```
static struct file_operations ilf_fops = {
        read_ilf
        write_ilf,
        NULL,     /* Não usaremos a função ilf_readdir */
```

```
NULL,     /* Não usaremos a função ilf_select */
NULL,     /* Não usaremos a função ilf_ioctl */
mmap_ilf,
NULL,     /* Não usaremos a função ilf_open_ilf */
open_ilf,
release_ilf,
NULL,     /* Não usaremos a função ilf_fsync */
NULL,     /* Não usaremos a função ilf_fasync */
NULL,     /* Não usaremos a função ilf_revalidate */
};
```

Esta é a estrutura da *file_operations* a ser passada ao VFS.

```
#ifndef MODULE
long ilf_init (long mem_start, long mem_end)
{
/* Verificaremos se a posição de memória está válida e não conflitante */
return mem_start;
}
```

Este dispositivo sendo compilado diretamente no *kernel* a função de inicialização vai precisar ser chamada em algum lugar quando o *kernel* estiver fazendo *boot*. Muitos dispositivos são inicializados a partir de *mem_init* em ***mem.c***. É permitido (e usamos esta técnica), alocar um bloco de memória colocando um apontador para *mem_start*, adicionando a quantidade de memória necessária para *mem_start*, e retornando um novo apontador não menor que *mem_end*.

Anexo

A

Implementação de um *Device Drivers para Plataforma Linux*

O sistema operacional Linux (plataforma do COMATM) é um clone do Unix; e no seu funcionamento os dispositivos são apresentados (ou melhor comunicam-se) aos programas como arquivos "especiais". Portanto, estes dispositivos implementam uma semântica de arquivo dentro do *kernel* (o núcleo do sistema operacional). Por causa disto nós vamos mostrar nas seções a seguir como os arquivos de forma genérica são tratados pelo Linux antes de falar do *drivers* em si.

1.1 Características dos Arquivos do Linux

O cabeçalho genérico do sistema de arquivo, <linux/fs.h>, define várias estruturas para acessar arquivos. *super_block* contem a informação básica sobre cada sistema de arquivo, e *super_operation* é uma estrutura de apontadores para funções associadas com o superbloco do sistema de arquivo. E é através desta estrutura que podem ser acessadas *inode_operations* e *file_operations*, a última, definindo as funções que podem ser usadas para acessar arquivos. Em sistemas

de arquivos comuns, tem-se somente um conjunto de operações sobre arquivo para todos os arquivos no sistema de arquivo, mas eles (os sistemas de arquivos) não tentam definir nenhuma operação nos arquivos especiais de dispositivos. Em vez disto, esses dispositivos definem suas próprias funções de operação e registram sua própria estrutura de *file_operations* com o Sistema Virtual de Arquivo (VFS - *Virtual File System*).

1.1.1 VFS

Operações genéricas de sistema de arquivo são tratadas por um código genérico de sistema de arquivo, e somente quando operações de *filesystem-dependent*, ou um *device-dependent*, precisam ser efetuadas é que o código para essa sistema de arquivo ou dispositivo é efetivamente chamado. A função necessária é procurada na própria instância de uma das estruturas de **_operation* e chamada. O código VFS fica no subdiretório **/fs** da fonte do *kernel* do Linux, e o código de cada sistema de arquivos é guardado em subdiretórios do subdiretório **/fs**.

1.1.2 Arquivos Especiais e Sistemas de Arquivo

Todas versões do Unix tem arquivos especiais para dispositivos. Todavia, O VFS é tão flexível que não somente arquivos especais podem ser criados mas sistemas de arquivos também. O sistema operacional Linux tem um sistema de arquivos chamado de sistema de arquivo proc (**profs** -*proc filesystem*) que é essencialmente um sistema de arquivo. Estes arquivos são muito parecidos aos dispositivos de hardware, porque eles geram arquivos de dados que não são de

nenhum arquivo e os apresentam ao usuário na forma de arquivo. Eles são, dispositivos virtuais projetados para relatar o estado do *kernel*.

1.1.3 Operações sobre Arquivos

A ILF, para comunicar-se com o sistema hospedeiro, precisa efetuar operações de leitura e escrita no barramento ISA que é a interface de comunicação. Para efetuar tais operações precisa-se definir uma estrutura de *file_operations* da seguinte maneira:

```
struct file_operations {

    int (*read) ( struct inode *, struct file *, char *, int);

    int (*write) ( struct inode *, struct file *, char *, int);

    int (*mmap) ( struct inode *, struct file *, struct vm_area_struct *);

    int (*release) ( struct inode *, struct file *);

                }
```

Como o *kernel* do Linux é monolítico. Somente um controle de thread criado pela chamada de sistema fica ativo a qualquer momento, então o driver da ILF não vai precisar bloquear as suas estruturas de dados.

1.1.3.1 Interface do Kernel

A maneira mais fácil de se desenvolver um *device driver* é usando um módulo de *kernel* carregável **Erro! A origem da referência não foi**

encontrada.]. Quando forem bem escritos, esses módulos podem ser facilmente utilizados na sua forma normal como módulos carregáveis, ou "re-compilado" e ligado com o resto do *kernel*. O símbolo *MODULE* é definido cada vez que um módulo está sendo compilado. O símbolo *_KERNEL_* é sempre definido quando se está compilando o código do *kernel*, mesmo quando se está compilando somente os módulos.

Agora podemos apresentar a implementação do driver da ILF . Nós vamos compilar o código usando a seguinte sintaxe:

```
/* gcc -O -DMODULE -D_KERNEL -c ilf.c */
```

Todo código de *kernel* do Linux deverá ser otimizado porque ele requer algumas extensões do gcc que são ativadas somente com a otimização

```
#include <linux/config.h>
```

Qualquer *device driver* deve incluir *<linux/config.h>* antes de incluir qualquer outro arquivo.

Símbolos do *kernel* que foram exportados para os módulos devem Ter seus nome manuseados de mesma maneira que é feito em C++, assim as mudanças nas estruturas do *kernel* serão notificadas. Isso provoca a não carga do módulo, porque se uma estrutura de *kernel* é mudada, carregar um módulo antigo que foi compilado com uma versão diferente da estrutura poderá danificar a integridade do sistema.

```
#ifdef MODULE
#include <linux/module.h>
#include <linux/version.h>
#else
#define MOD_INC_USE_COUNT
#define MOD_DEC_USE_COUNT
#endif
#include <linux/types.h>
#include <linux/fs.h>
#include <linux/mm.h>          /* Para verificar a área */
#include <linux/errno.h>       /* Para -EBUSY */
#include <asm/segment.h>       /* Para put_user_byte */
```

No funcionamento da ILF o sistema de gerência do COMATM ou a camada ATM podem querer ler o conteúdo da posição de memória onde se encontra as células de gerenciamento OAM geradas pela *camada Física*. Para isso definimos uma operação de leitura da seguinte forma:

```
static int read_ilf (struct inode *, struct file * file, char buf *, int count, int left);
main()
{
if ( verify_area(VERIFY_WRITE, buf, count0 ==-EFAULT) return -EFAULT;
```

```
for (left = count; left = 0; left --)
        {
        put_user_byte(0,buf);
        buf ++;
        }
        return count;
}
```

Cada vez que a chamada de sistema *read* for utilizada para a ILF, *read_ilf* é chamada. A função *put_user_byte* coloca um *byte* na posição de memória anteriormente definida. *read_zero* vai retornar o número de *byte*s que foi realmente escrito no *buffer* de leitura.

Antes de escrever no *buffer*, nós verificamos se é legal (i.e. não haverá violação) escrever no *buffer* inteiro, usando a função *verify_area*(). Isso nos evita gerar uma falha no *kernel*. A função de escrita na memória é declarada da seguinte maneira:

```
static int write_ilf(struct inode * inode, struct file * file, char buf * , int count) return
count;
```

Cada vez a chamada de sistema write for chamada para a ILF, write_ilf é chamada. A função *write_zero* vai retornar o número de *byte*s que foi colocado na memória. *write_ilf* ignora suas entradas e retorna o resultado bem sucedido de uma operação de escrita.

```
static struct file_operations ilf_fops = {
        read_ilf
        write_ilf,
        NULL,       /* Não usaremos a função ilf_readdir */
```

```
NULL,     /* Não usaremos a função ilf_select */
NULL,     /* Não usaremos a função ilf_ioctl */
mmap_ilf,
NULL,     /* Não usaremos a função ilf_open_ilf */
open_ilf,
release_ilf,
NULL,     /* Não usaremos a função ilf_fsync */
NULL,     /* Não usaremos a função ilf_fasync */
NULL,     /* Não usaremos a função ilf_revalidate */
};
```

Esta é a estrutura da *file_operations* a ser passada ao VFS.

```
#ifndef MODULE
long ilf_init (long mem_start, long mem_end)
{
/* Verificaremos se a posição de memória está válida e não conflitante */
return mem_start;
}
```

Este dispositivo sendo compilado diretamente no *kernel* a função de inicialização vai precisar ser chamada em algum lugar quando o *kernel* estiver fazendo *boot*. Muitos dispositivos são inicializados a partir de *mem_init* em ***mem.c***. É permitido (e usamos esta técnica), alocar um bloco de memória colocando um apontador para *mem_start*, adicionando a quantidade de memória necessária para *mem_start*, e retornando um novo apontador não menor que *mem_end*.

Anexo

B

UTOPIA

A flexibilidade da tecnologia ATM no uso de vários tipos de meio físico ou sistemas de transmissão está atrelada ao uso do padrão UTOPIA (*Universal Test and Operation PHY Interface*) para a conexão entre a camada ATM e a camada Física. A UTOPIA define então o protocolo de comunicação entre a camada Física e os módulos das camadas superiores com a camada ATM. Este padrão permite uma interface física comum no subsistema ATM com os vários tipos de meios de transmissão e conexões com taxas que variam de 100 Mbps a 622 Mbps.

A especificação define dois grupos de sinais independentes entre as camadas ATM e Física: Interface de Transmissão e Interface de Recepção. Esta especificação define as características mínimas primeiramente da camada Física do dispositivo e em segundo nível o dispositivo (SAR - *Segmentation And Ressembly*) do ATM. A Figura 1 mostra o relacionamento da Interface de Transmissão com os componentes do ATM e PHY de um subsistema ATM.

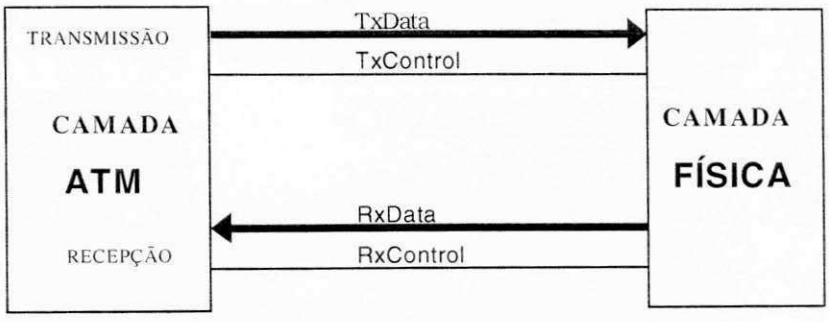


Figura 1 Diagrama da Interface UTOPIA

A Interface de Transmissão é composta de um barramento unidirecional de 8 bits (**TxData**) onde trafegam sinais oriundos da camada ATM, sob o controle dos sinais do conjunto (TxControl) como ilustrado na Figura 1

A Interface de Recepção é composta de um barramento unidirecional de 8 bits (**TxData**) onde trafegam sinais oriundos da camada ATM, sob o controle dos sinais do conjunto (TxControle) como ilustrado na Figura 6

1.1 Descrição Funcional

Por convenção, a interface por onde os dados passam vindo da camada ATM para a camada Física é chamada de Interface de Transmissão, e a interface por onde os dados passam vindo da camada Física para a camada ATM é chamada de Interface de Recepção. E todos os sinais são ativos no nível lógico alto, menos os sinais que possuem um asterix "*" por exemplo:

a) SINAL_1 Ativo alto;

b) SINAL_2* Ativo baixo.

A transmissão e recepção de dados é sincronizada através dos relógios das respectivas interfaces. Em um caminho de 8 bits de dados e uma taxa

máxima de relógio de 25 MHz, a interface suporta taxas variando de 100 a 155 Mbps. Alguns exemplos de interface com tais características são citados abaixo:

a) 155,52 Mbps (SONET/OC-3)

b) 155,52 Mbps (com codificação 8B/10B)

c) 100 Mbps (TAXI, com codificação 4B/5B)etc..

Taxas superiores (por exemplo 622 Mbps) podem ser suportadas estendendo-se o caminho de dados para 16 bits, e usando-se uma freqüência de relógio mais alta. Isso nos leva a tratar dos modos de transferência.

1.1.1 Modos de Transferência de Dados

Dois modos de transferência entre a camada ATM e a camada Física são previstos pela UTOPIA, sendo eles: o modo 8 bits e o modo 16 bits.

1.1.1.1 Modo 8 Bits

O modo 8 bits define a transferência de dados em palavras de 8 bits, i.e., o barramento de dados utilizado para transferir a célula é de 8 bits; e a célula portanto é transmitida por octeto. A Figura 2 ilustra o formato de transferência de uma célula no modo 8 bits:

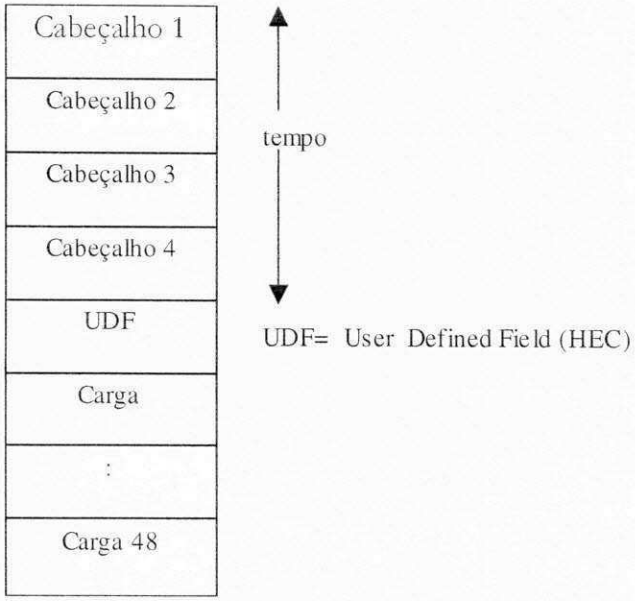


Figura 2 Formato da Célula para Transferência em modo 8 bits

1.1.1.2 Modo 16 Bits

O modo 16 bits define a transferência de dados em palavras de 16 bits; a célula portanto é transmitida usando um barramento de dados de 16 bits. A Figura 3 ilustra o formato de transferência de uma célula no modo 16 bits.

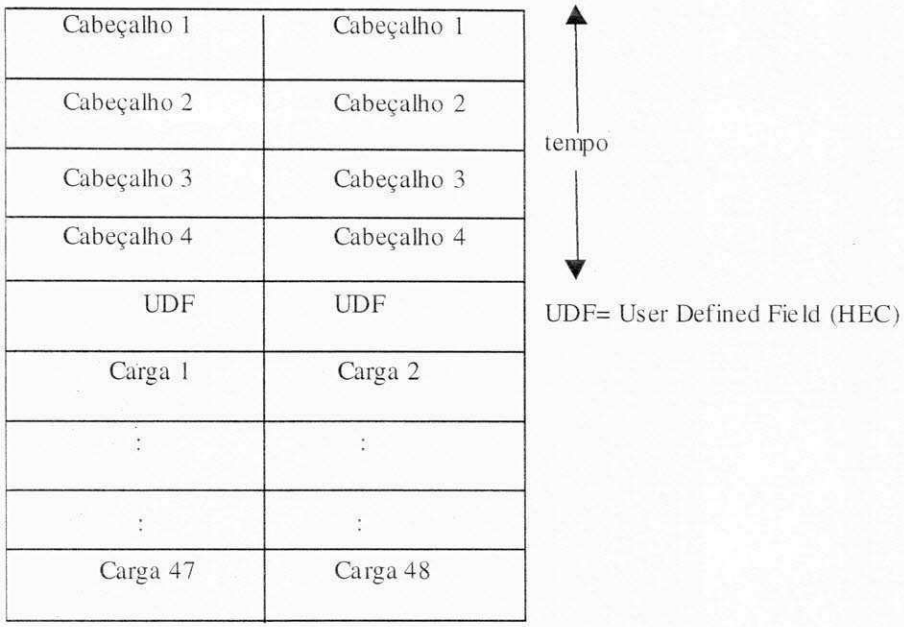


Figura 3 Formato da Célula para Transferência em modo 16 bits

1.1.2 Sincronismo

As Interfaces de Transmissão e a Interface de Recepção, ilustrados na Figura 4 são responsáveis pela transmissão e recepção dos dados. Para estabelecer um nível de sincronismo entre as camadas, usam-se os respectivos relógios (Tx*Clock*/Rx*Clock*) e os sinais indicando a primeira célula válida no barramento de transporte *Start of Cell* (**TxSoc**/**RxSoc**)

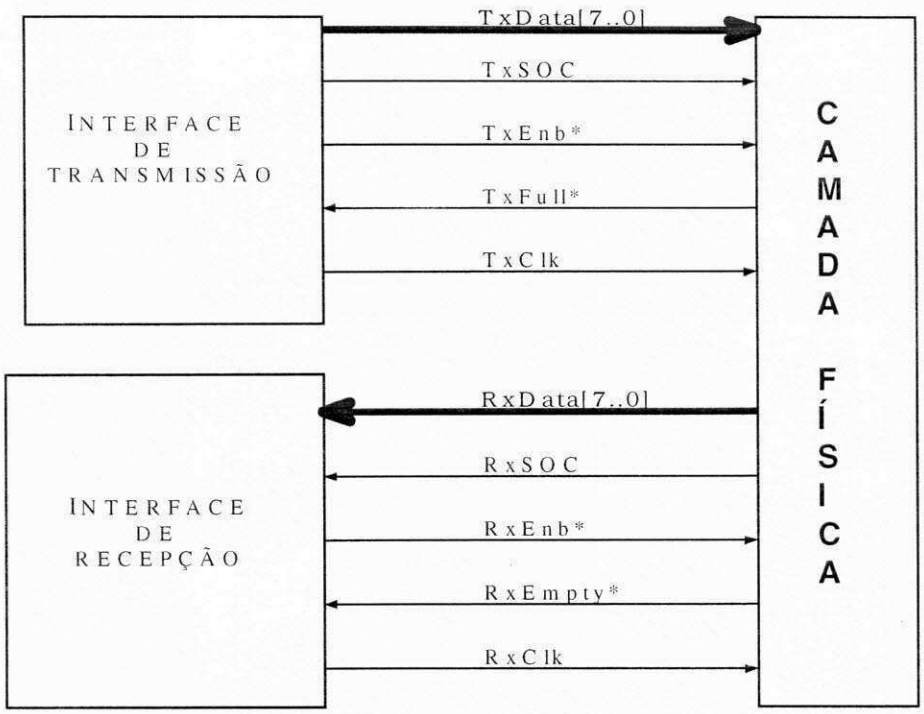


Figura 4 Interface de Transmissão e Recepção

1.1.3 Mecanismo de Transmissão de Dados

A Interface de Transmissão é responsável pela transmissão dos dados da camada ATM para a camada Física. Para isso ela espera receber o sinal **TxFull*** da camada Física. Quando detectar o sinal **TxFull*** com nível lógico 0 (indicando que mais 4 octetos poderão ser aceitos pela camada Física), a interface ativa o sinal **TxEnb*** e envia os dados pelo barramento, na borda positiva do sinal Tx*Clock*.

A Interface de Recepção é responsável pela transmissão dos dados da camada Física para a camada ATM; quando o sinal **RxEnb*** é ativado pela camada ATM, indicando que serão amostrados os sinais **RxSoc** e **RxData** no

final do próximo ciclo. A camada Física em resposta a essa requisição, envia o sinal **RxEmpty*** com valor 0 para indicar que há dados a serem enviados à camada ATM. O sincronismo de transferência é feito na borda de subida.

1.1.4 Interface de Transmissão

1.1.4.1 Sinais

O protocolo de comunicação entre a camada ATM e a camada Física é feito através de sinais definidos pela UTOPIA, divididos em dois tipos: básicos (**TxSoc**, **TxEnb***, **TxFull** e **TxClk**) e opcionais (**TxPrty** e **TxRef***). A Tabela 1 ilustra os sinais definidos nesse padrão e seus respectivos significados.

Sinal	Significado	Observação
TxData[7..0]	Barramento unidirecional da camada ATM para a camada Física; contém os dados a serem transferidos. TxData[7] é o MSB	Para o modo 16 bits, o barramento é definido como TxData[15..0]
TxSoc	*Start Of Cell.* Sinal ativo alto enviado pela camada ATM, indicando a presença do primeiro *octeto* válido a ser transportada	Para o modo 16 bits, indica a primeira palavra de 16 bits
TxEnb*	*Enable.* Sinal ativo baixo enviado pela camada ATM para indicar a presença de dados válidos no barramento TxData	
TxFull*	*Full Available.* Sinal ativo baixo enviado pela camada Física para indicar que um máximo de 4 octetos serão aceitos pela camada Física.	Para o modo 16 bits, indica que uma célula completa será aceita pela camada Física (TxClav - Fluxo de células disponível)
TxClk	*Clock* de Transferência. Sinal de *clock* gerado pela camada ATM para sincronismo de transferência dos dados presentes no barramento TxData[7..0]	Transferência efetuada na borda de subida do relógio
TxPrty[0]	*Parity*. sinal enviado pela camada ATM para indicar a paridade (ímpar) do barramento TxData[7..0]	Sinal opcional. Para o modo 16 bits, é definido um segundo sinal TxPrty[1], que indica a paridade do barramento TxData[15..0]
TxRef*	*Transmit reference.* Sinal enviado pela camada ATM para propósitos de sincronismo (e.g. marcador de 8 kHz, indicador de quadro, etc)	Sinal opcional

Tabela 1 Padrão de Sinais da Interface de Transmissão

1.1.5 Operação e Temporização

A Interface de Transmissão é controlada pela camada ATM. A camada ATM fornece um relógio para a camada Física para efeito de sincronização. Os dados são transmitidos da camada ATM para a camada Física da seguinte maneira:

a) a camada Física indica que ela aceita dados usando os sinais **TxFull***/TxClav (**TxFull*** = 1);

b) então a camada ATM coloca dados no **TxData** e ativa o sinal TxEnb;

c) uma vez posto o primeiro octeto no barramento de dados, a camada ATM gera o sinal **TxSoc**;

d) a camada Física controla o fluxo de célula através do sinal **TxFull*** como ilustrado na Figura 5.

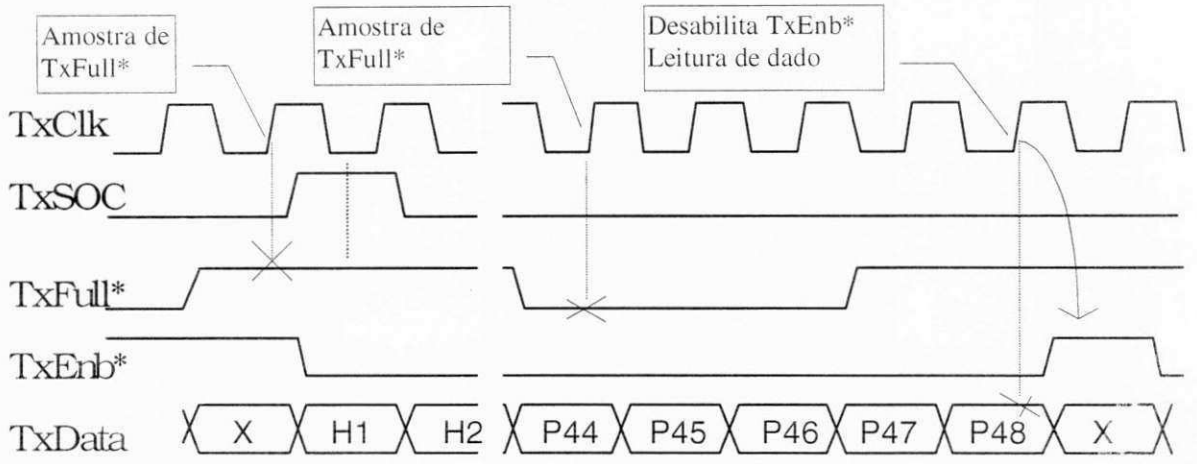


Figura 5: Diagrama de tempo de transmissão (em modo 8 bits).

1.1.5.1 Janela de Transmissão

A janela de transmissão é o intervalo de tempo indicado na transição do sinal **TxFull*** para o nível 1 (indicando a disponibilidade da camada Física em receber dados) até 4 ciclos de *clock* após a transição do sinal **TxFull*** para o nível lógico 0 (que indica a indisponibilidade da camada Física para a recepção de dados). Isto é ilustrado no diagrama de tempo da Figura 6

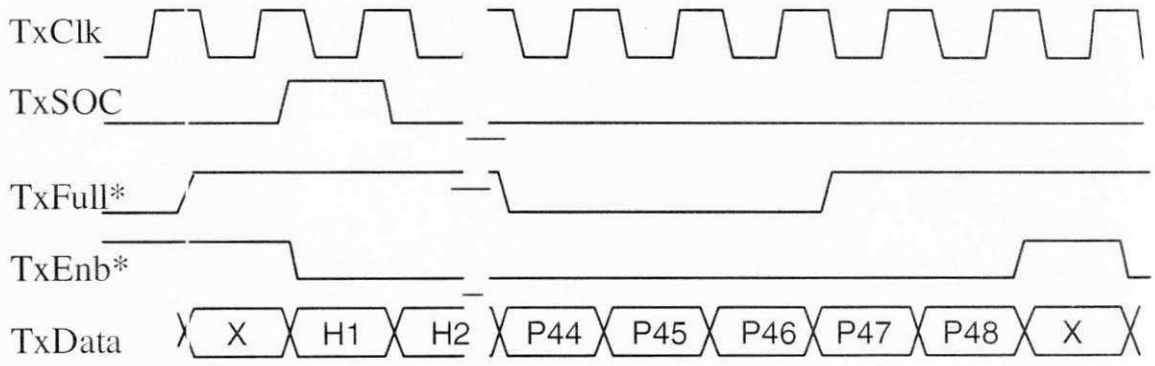


Figura 6: Diagrama de Tempo: Ativação do sinal **TxEnb***.

A ativação do sinal TxEnb pela Interface de Transmissão da ATM, deve obedecer alguns requisitos de tempo, relacionados com a janela de transmissão.

a) a ativação do sinal **TxEnb*** fora da janela de transmissão constitui um erro;

b) a ativação do sinal **TxEnb*** deve ser feita apenas quando for detectado o sinal **TxFull*** desativado (i.e., **TxFull*** = 1) e dentro da janela;

c) a desativação do sinal **TxEnb*** (=1) deve ser feita quatro ciclos de *clock* após a ativação do sinal **TxFull*** (=0).

1.1.5.2 Características da Temporização

Os seguintes requisitos/especificações das camadas ATM e Física dizem respeito à Interface de Transmissão (observar que onde foi referenciado **TxData**, a mesma temporização é aplicada a **TxData**, **TxPrty**, e **TxSoc**).

Item	Descrição	Escopo	Mín	Máx
tT1	Freqüência do TxClk	TxClk	0	25 mH
tT2	Ciclo de Tarefa do TxClk	TxClk	40%	60%
tT3	Atraso de saída de TxClk	TxData, TxEnb*, TxRef*	1 ns	20 ns
tT4	Inicialização de Entrada para o TxClk	TxFull* / TxClav	10 ns	
tT5	Entrada obtida de TxClk	TxFull* / TxClav	1 ns	
tT6	Tamanho do pulso do TxRef* alto ou baixo	TxRef*	**1** TxClk	**1** TxClk

Tabela 2: Características de Temporização da Interface de Transmissão.

1.1.6 Interface de Recepção

1.1.6.1 Sinais

São definidos para a Interface de Recepção, um barramento unidirecional de 8 ou 16 bits, 4 sianis básicos (**RxSoc**, **RxEnb***, **RxEmpty*** e **RxClk**) e 2 dois sinais opcionais (RxPrty e **RxRef***). A Tabela 3 ilustra os sinais definidos nesse padrão e seus respectivos significados.

Sinal	Significado	Observação
RxData[7..0]	Barramento unidirecional da camada Física para a camada ATM; contem os dados a serem transferidos. RxData[7] é o MSB	Para o modo 16 bits, o barramento é definido como RxData[15..0]
RxSoc	*Start of Cell.* Sinal ativo baixo enviado pela camada Física, que indica	Para o modo 16 bits, indica a primeira palavra de 16 bits
RxEnb*	*Enable.* Sinal ativo baixo enviado pela camada ATM indicando que os sinais ExData e RxSoc serão amostrados no próximo ciclo	
RxEmpty*	*Empty.* Sinal ativo baixo enviado pela camada Física, indicando que no ciclo atual, não existe octeto a ser enviado à camada ATM.	Para o modo 16 bits, o sinal RxClav (*Cell Availabe*) substitui o RxEmpty*, ele é ativo alto, indicando a existência de uma célula completa disponível para transferência
RxClk	*Clock* de Transferência. Sinal de *clock* gerado pela camada ATM para sincronismo de transferência dos dados presentes no barramento RxData[7..0]	Transferência efetuada na borda de subida do relógio
RxPrty[0]	*Parity.* sinal enviado pela camada ATM para indicar a paridade (ímpar) do barramento RxData[7..0]	Sinal opcional. Para o modo 16 bits, é definido um segundo sinal RxPrty[1], que indica a paridade do barramento RxData[15..0]
RxRef*	*Receive reference.* Sinal enviado pela camada Física para propósitos de sincronismo (e.g. marcador de 8 Khz, indicador de quadro, etc.).	Sinal opcional

Tabela 3: Padrão de Sinais da Interface de Recepção.

1.1.6.2 Operação e Diagrama de Tempo

A Interface de Recepção é controlada pela camada ATM. A camada ATM fornece um sinal de relógio (**RxClk**) para sincronismo da transferência e desta forma a camada Física deve incluir *buffers para* retenção dos dados (FIFO - *Firts In First Out*).

A interface transfere dados no sentido da camada Física para a camada ATM. A Interface de Recepção gera todos os sinais de saída e amostra todos os sinais de entrada na borda de subida de **RxClk**. A recepção dos dados é efetuada da seguinte maneira:

a) a camada Física indica que existem dados a serem transferidos, através do sinal **RxEmpty*** (=1);

b) a camada ATM ao amostrar o sinal **RxEmpty*** (=1), ativa **RxEnb*** (=1) indicando que será efetuada a leitura do barramento **RxData** [7..0];

c) a camada Física gera o sinal **RxSoc** sempre que o barramento **RxData**[7..0] contiver o primeiro octeto da célula.

d) a camada Física controla o fluxo de dados através de **RxEmpty***.

A figura 8 ilustra o comportamento da Interface de Recepção através de um diagrama de tempo.

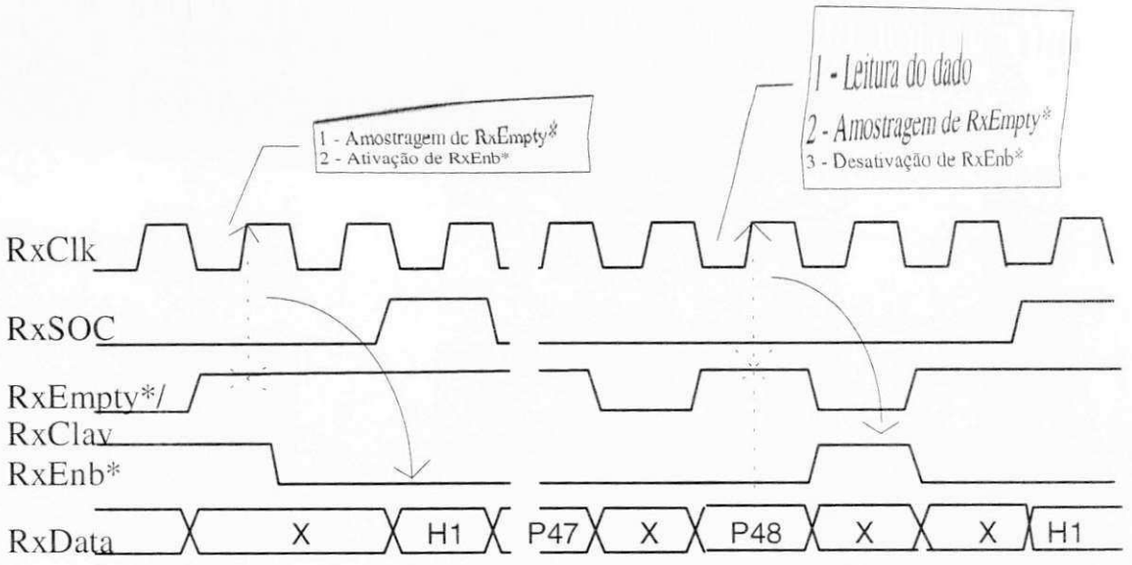


Figura 7: Diagrama de Tempo da Interface de Recepção.

1.1.6.2.1 Janela de Leitura

O intervalo de tempo em que o sinal **RxEnb*** está ativo (=0) representa a janela de leitura.

Durante a janela de leitura, a camada Física coloca os dados no barramento **RxData**, na borda positiva de **RxClk** e indica que há dados, desativando o sinal **RxEmpty*** (=1). A ativação e desativação do sinal **RxEnb*** pode ser feita em qualquer instante como ilustrado na Figura 26.

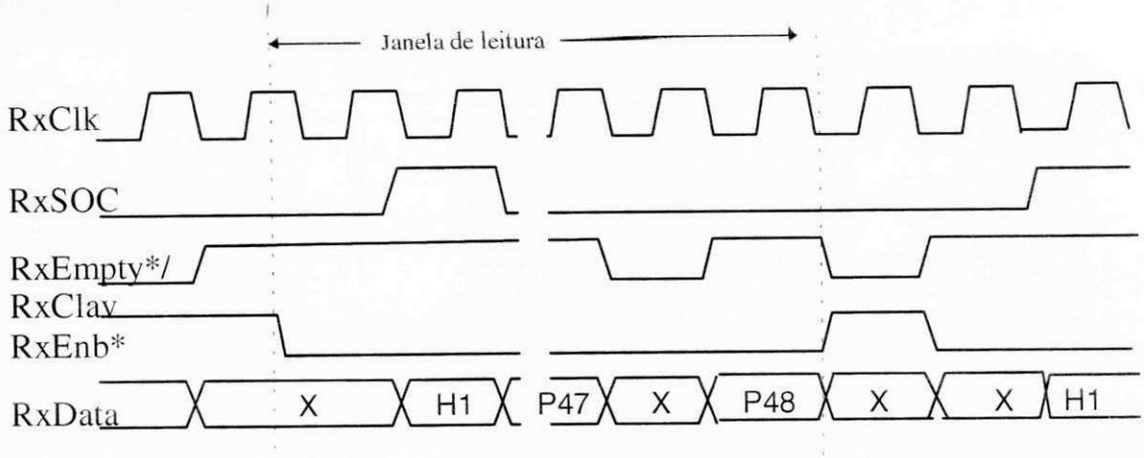


Figura 8: Diagrama de Tempo da Ativação do sinal **RxEnb*.**

O diagrama para **RxData** mostra o valor dos dados em ciclos durante os quais a transferência da camada Física para a camada ATM é efetiva.

1.1.6.2.2 Características de temporização

Os seguintes requisitos/especificações das camadas ATM e Física dizem respeito a Interface de Recepção (observar que onde foi referenciado **RxData**, a mesma temporização é aplicada a **RxData**, **RxPrty**, e **RxSoc**). A Tabela 4 e a Figura 9 ilustram as características de temporização da Interface de Recepção.

Item	Descrição	Escopo	Mín	Máx
tR1	Freqüência do RxClk	RxClk	0	25 MHz
tR2	Ciclo de Tarefa do RxClk	RxClk	40%	60%
tR3	Atraso de saída de RxClk	RxData, RxEnb*, RxRef*	1 ns	20 ns
tR4	Inicialisação de Entrada para o RxClk	RxFull*/ RxClav	10 ns	
tR5	Entrada obtida de RxClk	RxFull*/ RxClav	1 ns	
tR6	Tamanho do pulso do RxRef* alto ou baixo	RxRef*	1 RxClk	1 RxClk

Tabela 4: Características de Temporização da Interface de Recepção.

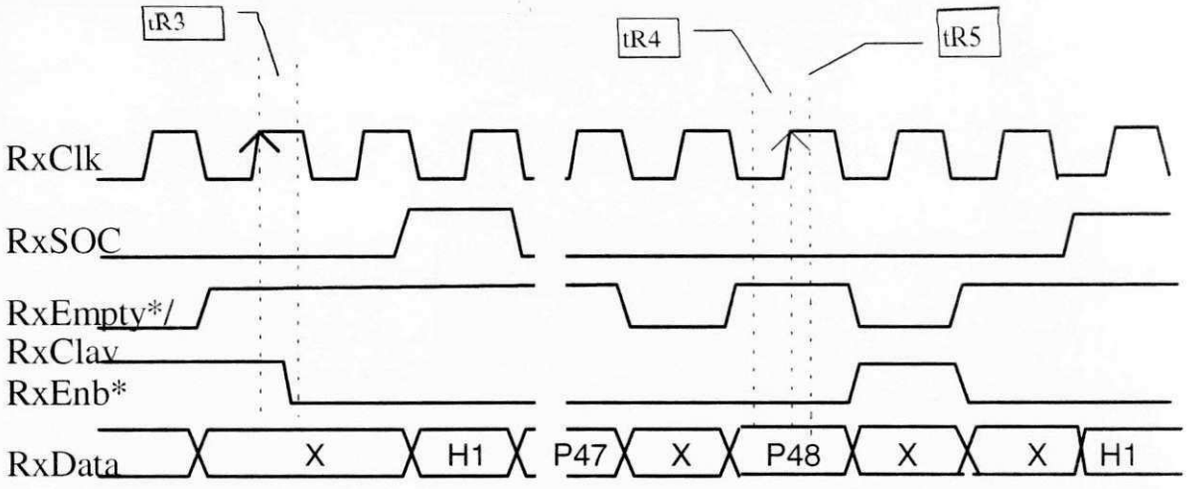


Figura 9 Características de Temporização da Interface de Recepção

Anexo

C

Características Funcionais do *Cyclic Redundancy Check* (CRC)

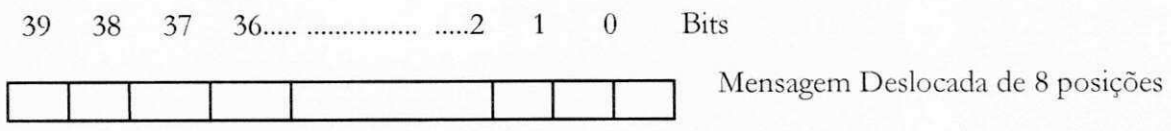
O objetivo principal da utilização do CRC como técnica de detecção de erro é permitir que o receptor de uma mensagem transmitida num meio sujeito a ruído, possa determinar se a mensagem foi ou não corrompida. Para isso, o transmissor, a partir de uma certa lógica, gera um valor que vai ser inserido na mensagem a ser transmitida. Tal valor é chamado de *checksum* pela sigla conhecida como CRC. O receptor poderá portanto utilizar a mesma função que permitiu ao transmissor gerar o CRC, para calcular o CRC da mensagem recebida e compará-lo com o valor inserido na mensagem e verificar se a mensagem recebida está ou não correta.

A idéia atrás da lógica para gerar o CRC (i.e., o algoritmo do CRC) é simplesmente a de tratar a mensagem como sendo um grande número de bits , a ser dividido por um outro número binário prefixado, e fazer do resto da divisão o CRC. Uma vez recebida a mensagem, o receptor vai efetuar a mesma divisão e comparar o resto com o CRC recebido.

1.1 Funcionamento do algoritmo do cálculo do CRC de forma serial

O cálculo do CRC em todos os algoritmos tem um fundamento básico: efetuar uma divisão polinomial da mensagem a transmitir com o polinômio gerador. O resto desta divisão (o CRC) é inserido na mensagem a transmitir.

Para implementar este algoritmo, insere-se a mensagem a transmitir num registrador de divisão. No caso do HEC esta mensagem é de 32 bits. Uma vez isso feito, a mensagem é deslocada de 8 posições (correspondendo ao valor da potência do polinômio gerador $x^8 + x^2 + x + 1$) conforme ilustrado na



x^8 + x^2 + x + 1 = 100000111 = Polinômio gerador

Figura 1 Formato de uma Mensagem a ser Processada

Para efetuar tal divisão, pode ser utilizado um registrador de 8 bits. Como estamos utilizando operações módulo-2, a divisão se resume em uma série de subtrações (ou operações de XORs consecutivas). Portanto, efetua-se uma série de subtrações de várias potências (i.e., fazendo-se deslocamentos laterais) do polinômio, a partir da mensagem, até que fique somente o resto . Veja exemplo na Tabela 1.

Divisão Polinomial Módulo 2	Resultado Final
1101001010100 / 100000111 = =110100101010100 100000111 ---------- 0101000100 100000111 ---------- 0010000111	00101000 + 01010101= HEC 00001010 + 01010101 ---------- 01011111 = HEC

. . . . --- 010000110 1 10000011 1 --- 000001010 = CRC	

Tabela 1 Exemplo de Procedimento de uma Divisão Polinomial Módulo-2

Uma vez tido o CRC será feita uma adição binária (módulo 2) com o octeto padrão 01010101 (i.e., uma operação de XOR); tendo como resultado o HEC. Portanto teriamos o seguinte resultado utilizando o CRC da tabela 1:

00001010 + 01010101 =01011111

O algoritmo da divisão polinomial está descrito na Tabela 2:

Passos	Procedimento
1	*Carregar os registradores com bits zero*
2	*Deslocar a mensagem de 8 posições, (i.e. inserindo 8 zeros no final da mensagem)*
3	*Enquanto tem bits na mensagem* { *Deslocar o registrador de um bit a esquerda, lendo o próximo bit da mensagem deslocada no registrador contendo o bit da posição 0* *Se (1 bit for movido fora do registrador na etapa 3) então* *registrador = registrador XOR polinômio* *fim Se* }

Tabela 2: Algoritmo de Divisão Polinomial.

O registrador agora contem o resto.

O circuito resultante de uma tal operação está ilustrado na Figura 2.

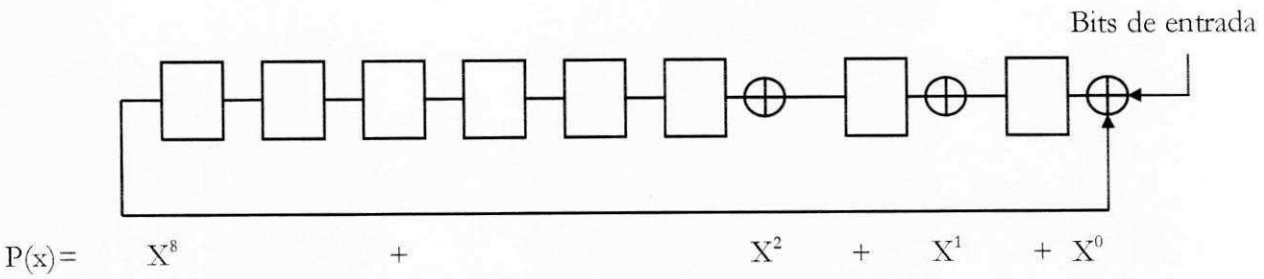


Figura 2 Circuito da Geração do CRC Usando uma entrada Serial

No exemplo apresentado na Tabela 1, a operação de divisão polinomial é feita bit a bit. Portanto, como temos uma mensagem com tamanho de 40 bits (cabeçalho da célula ATM menos o HEC, deslocado de 8 posições) precisaremos de pelo menos 40 pulsos de relógios para poder ter o CRC. Este procedimento é simples, mas na maioria dos casos (como é o nosso), ele fica lento por ter uma limitação de tempo devido à operação serial feita para obter o resultado (tendo que efetuar um *loop* para cada bit). Este atraso vai também criar um certo atraso nos processos subsequentes. Isso nos motivou a procurar um outro meio de calcular o HEC, considerando as restrições de atraso imposto pelo projetos COMATM.

1.1.1 Funcionamento do algoritmo do cálculo do CRC de forma paralela

A operação de geração do HEC, neste caso, funciona da seguinte maneira: a seqüência de bits recebida fica armazenada num conjunto de registradores, depois disso, efetua se um deslocamento de oito posições a esquerda. Uma vez isto feito, começa-se a divisão polinomial do quadro obtido. E com o CRC obtido, efetua-se uma operação de XOR com o octeto padrão 01010101. O resultado disso representa o HEC a ser inserido no quinto octeto do cabeçalho.

Para tornar o algoritmo acima mais rápido (i.e. podendo ser processado em um intervalo de tempo menor), precisamos encontrar um meio para permitir que o algoritmo processe a mensagem em unidades maiores do que um bit; 8 bits por exemplo. Portanto a mensagem deslocado de 8 bits terá a aparência ilustrada na Figura 3.

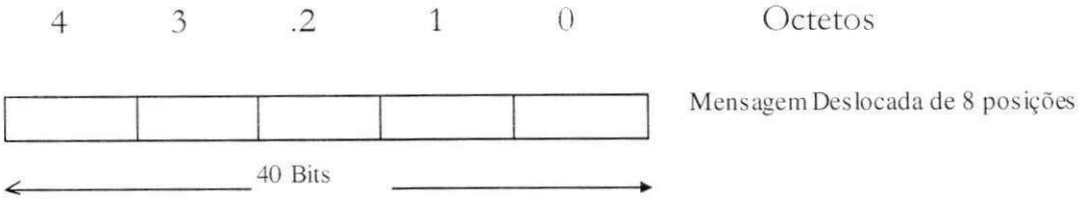


Figura 3: Mensagem de 5 Octetos deslocada de 8 posições.

O algoritmo anterior apresentado na Tabela 2 ainda é válido. Chamamos este algoritmo é chamado de "Serial-Algo" Vamos considerar os 8 primeiros bits do registrador de 40 bits (octeto 4) serem igual a : *t7 t6 t5 t4 t3 t2 t1 t0*

Na próxima iteração do "Serial-Algo", t7 vai determinar se haverá uma operação XOR do polinômio no registrador inteiro. Se t7 for igual a 1, esta operação será efetuada senão não acontecerá tal operação. Vamos agora supor que os 8 primeiros bits do polinômio gerador são:

g7 g6 g5 g4 g3 g2 g1 g0,

então na próxima iteração o octeto de topo será:

t6 t5 t4 t3 t2 t1 t0 ??

+ t7 *(g7 g6 g5 g4 g3 g2 g1 g0)

* lembrando que + é uma operação XOR

O novo bit de topo (que vai controlar o que vai acontecer na próxima iteração) tem agora o valor *t6 + t7*g7*. O que principalmente nos interessa neste

A implementação deste algoritmo em linguagem C requer somente um *shift* (deslocamento)e OR, um XOR, e uma tabela de *lookup* por octeto. Conforme ilustrado na Tabela4.

Algoritmo	Implementação em Linguagem C
While (a mensagem deslocado não estiver vazia) *Begin* *Top = top_*octeto*(registrador);;* *Registrador = (Registrador << 24) | próx_*octeto*_mensagem;* *Registrador = Registrador XOR Tabela_precomputado[topo]* *end*	*r=0;* *while (len--)* { octeto *t=(r >>24) xor 0xFF;* *r = (r<< 8) | *p++;* *r^=table[t];* }

Len: tamanho da mensagem deslocado de 8 bits,
p: aponta a esta mensagem,
r: é o registrador,
t: uma variável temporária,
e: *table* é a tabela computada.

Tabela 4 Algoritmo e Implementação do Cálculo do CRC usando uma tabela pré-computada

Para a implementação deste algoritmo usando ALTERA, podemos usar os LFSR (*Linear Feed-back Shift Register*). Pode-se assim expressar o conteúdo do registrador após 8 *shifts* como uma função do conteúdo inicial do *shift register* e dos 8 bits deslocados dentro do registrador. Esta função pode ser criada utilizando-se somente operações XOR. A Figura 4 mostra a função que utiliza o gerador polinomial X^8 + X^2 +X + 1. O conteúdo do registrador C[7..0] e os próximos 8 bits de dados D[7..0] são usados para calcular o conteúdo geral do registrador após 8 *shifts*. Como o registrador desloca uma vez para cada entrada de dado, o operador XOR8 produz as séries de termos (C0 D0), (C1 D1), etc. A estes termos atribuímos o nome Mn onde Mn= (Cn Dn). A operação XOR é aplicada a todos os termos que se encontram na mesma linha.

O circuito é composto de um registrador de 40 bits que armazena a mensagem (os quatro primeiros octetos do cabeçalho da célula deslocados de 8 posições à esquerda), de nove registradores armazenando o resultado da operação de divisão polinomial da mensagem pelo gerador polinomial, de um registrador que armazena o gerador polinomial e de um bloco funcional que guarda o apontador para o próximo bit, ou para o início do próximo conjunto de bits a ser inserido no resto da divisão anteriormente efetuada, completando assim o dividendo da operação de geração/ verificação do HEC. Este último bloco funcional permite também, através de sucessivas operações de comparações, determinar o número de bits que vão ser emendados ao resto da divisão para a próxima operação.

C

Características Funcionais do *Cyclic Redundancy Check* (CRC)

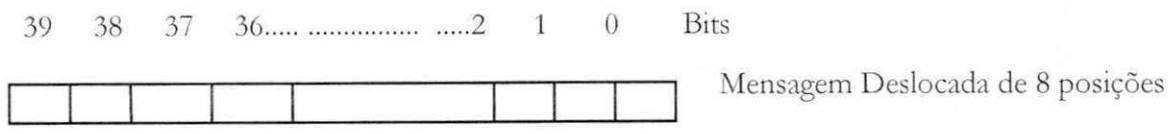
O objetivo principal da utilização do CRC como técnica de detecção de erro é permitir que o receptor de uma mensagem transmitida num meio sujeito a ruído, possa determinar se a mensagem foi ou não corrompida. Para isso, o transmissor, a partir de uma certa lógica, gera um valor que vai ser inserido na mensagem a ser transmitida. Tal valor é chamado de *checksum* pela sigla conhecida como CRC. O receptor poderá portanto utilizar a mesma função que permitiu ao transmissor gerar o CRC, para calcular o CRC da mensagem recebida e compará-lo com o valor inserido na mensagem e verificar se a mensagem recebida está ou não correta.

A idéia atrás da lógica para gerar o CRC (i.e., o algoritmo do CRC) é simplesmente a de tratar a mensagem como sendo um grande número de bits , a ser dividido por um outro número binário prefixado, e fazer do resto da divisão o CRC. Uma vez recebida a mensagem, o receptor vai efetuar a mesma divisão e comparar o resto com o CRC recebido.

1.1 Funcionamento do algoritmo do cálculo do CRC de forma serial

O cálculo do CRC em todos os algoritmos tem um fundamento básico: efetuar uma divisão polinomial da mensagem a transmitir com o polinômio gerador. O resto desta divisão (o CRC) é inserido na mensagem a transmitir.

Para implementar este algoritmo, insere-se a mensagem a transmitir num registrador de divisão. No caso do HEC esta mensagem é de 32 bits. Uma vez isso feito, a mensagem é deslocada de 8 posições (correspondendo ao valor da potência do polinômio gerador $x^8 + x^2 + x + 1$) conforme ilustrado na



x^8 + x^2 + x + 1 = 100000111 = Polinômio gerador

Figura 1 Formato de uma Mensagem a ser Processada

Para efetuar tal divisão, pode ser utilizado um registrador de 8 bits. Como estamos utilizando operações módulo-2, a divisão se resume em uma série de subtrações (ou operações de XORs consecutivas). Portanto, efetua-se uma série de subtrações de várias potências (i.e., fazendo-se deslocamentos laterais) do polinômio, a partir da mensagem, até que fique somente o resto . Veja exemplo na Tabela 1.

Divisão Polinomial Módulo 2	Resultado Final
1101001010100 / 100000111 = =110100101010100 100000111 --- 0101000100 100000111 --- 0010000111	00101000 + 01010101= HEC 00001010 + 01010101 --- 01011111 = HEC

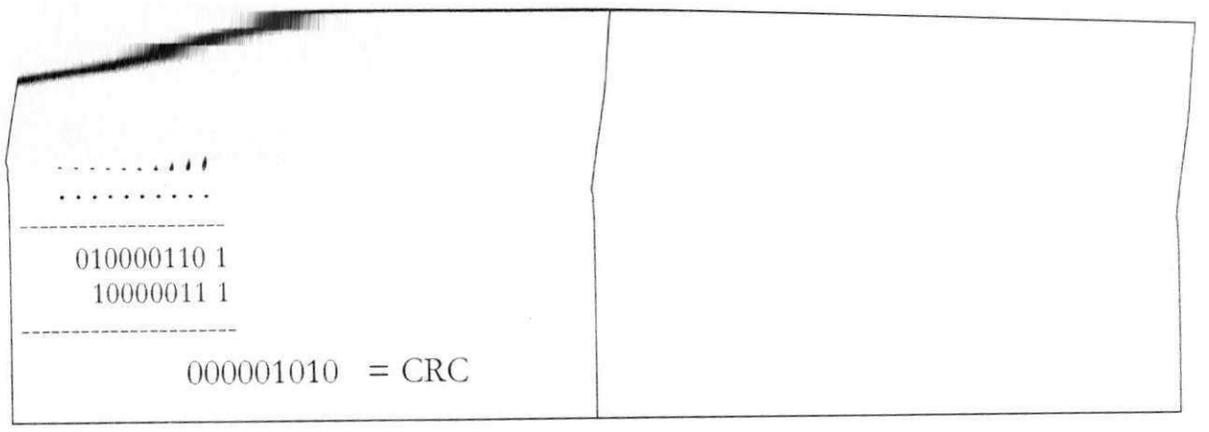


Tabela 1 Exemplo de Procedimento de uma Divisão Polinomial Módulo-2

Uma vez tido o CRC será feita uma adição binária (módulo 2) com o octeto padrão 01010101 (i.e., uma operação de XOR); tendo como resultado o HEC. Portanto teriamos o seguinte resultado utilizando o CRC da tabela 1:

00001010 + 01010101 =01011111

O algoritmo da divisão polinomial está descrito na Tabela 2:

Passos	Procedimento
1	*Carregar os registradores com bits zero*
2	*Deslocar a mensagem de 8 posições, (i.e. inserindo 8 zeros no final da mensagem)*
3	*Enquanto tem bits na mensagem* { *Deslocar o registrador de um bit a esquerda, lendo o próximo bit da mensagem deslocada no registrador contendo o bit da posição 0* *Se (1 bit for movido fora do registrador na etapa 3) então* *registrador = registrador XOR polinômio* *fim Se* }

Tabela 2: Algoritmo de Divisão Polinomial.

O registrador agora contem o resto.

O circuito resultante de uma tal operação está ilustrado na Figura 2.

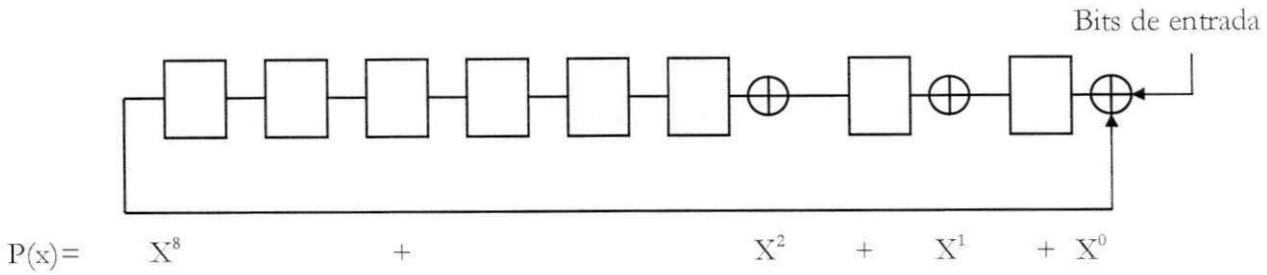


Figura 2 Circuito da Geração do CRC Usando uma entrada Serial

No exemplo apresentado na Tabela 1, a operação de divisão polinomial é feita bit a bit. Portanto, como temos uma mensagem com tamanho de 40 bits (cabeçalho da célula ATM menos o HEC, deslocado de 8 posições) precisaremos de pelo menos 40 pulsos de relógios para poder ter o CRC. Este procedimento é simples, mas na maioria dos casos (como é o nosso), ele fica lento por ter uma limitação de tempo devido à operação serial feita para obter o resultado (tendo que efetuar um *loop* para cada bit). Este atraso vai também criar um certo atraso nos processos subsequentes. Isso nos motivou a procurar um outro meio de calcular o HEC, considerando as restrições de atraso imposto pelo projetos COMATM.

1.1.1 Funcionamento do algoritmo do cálculo do CRC de forma paralela

A operação de geração do HEC, neste caso, funciona da seguinte maneira: a seqüência de bits recebida fica armazenada num conjunto de registradores, depois disso, efetua se um deslocamento de oito posições a esquerda. Uma vez isto feito, começa-se a divisão polinomial do quadro obtido. E com o CRC obtido, efetua-se uma operação de XOR com o octeto padrão 01010101. O resultado disso representa o HEC a ser inserido no quinto octeto do cabeçalho.

Para tornar o algoritmo acima mais rápido (i.e. podendo ser processado em um intervalo de tempo menor), precisamos encontrar um meio para permitir que o algoritmo processe a mensagem em unidades maiores do que um bit; 8 bits por exemplo. Portanto a mensagem deslocado de 8 bits terá a aparência ilustrada na Figura 3.

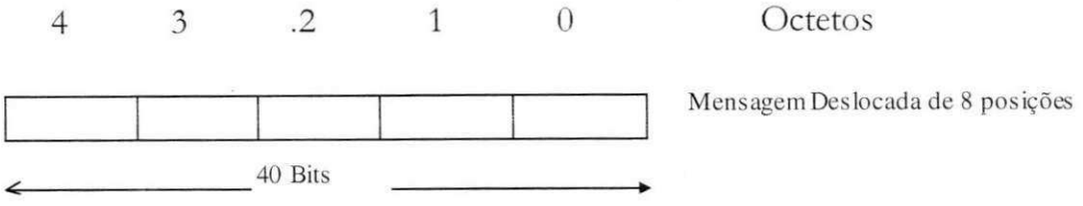


Figura 3: Mensagem de 5 Octetos deslocada de 8 posições.

O algoritmo anterior apresentado na Tabela 2 ainda é válido. Chamamos este algoritmo é chamado de "Serial-Algo" Vamos considerar os 8 primeiros bits do registrador de 40 bits (octeto 4) serem igual a : *t7 t6 t5 t4 t3 t2 t1 t0*

Na próxima iteração do "Serial-Algo", t7 vai determinar se haverá uma operação XOR do polinômio no registrador inteiro. Se t7 for igual a 1, esta operação será efetuada senão não acontecerá tal operação. Vamos agora supor que os 8 primeiros bits do polinômio gerador são:

g7 g6 g5 g4 g3 g2 g1 g0,

então na próxima iteração o octeto de topo será:

t6 t5 t4 t3 t2 t1 t0 ??

+ t7 *(g7 g6 g5 g4 g3 g2 g1 g0)

* lembrando que + é uma operação XOR

O novo bit de topo (que vai controlar o que vai acontecer na próxima iteração) tem agora o valor *t6 + t7*g7*. O que principalmente nos interessa neste

procedimento é que todas as informações necessárias para calcular o novo bit de topo estavam presentes nos dois bits de topo do octeto de topo original. De forma similar, o próximo bit de topo pode ser calculado antecipadamente a partir dos 3 bits de topo *t7*, *t6*,e *t5*. De fato, geralmente, o valor do bit de topo no registrador em k iterações pode ser calculado a partir dos k bits de topo do registrador.

Vamos considerar agora que nós vamos utilizar os 8 primeiros bits do registrador para calcular o bit de topo durante as 8 próximas iterações. Supomos também que vamos obter as 8 próximas iterações usando os valores calculados (que poderemos eventualmente armazenar em um único registrador de 8 bits). Com isso podemos observar o seguinte:

a) o octeto de topo não importa mais. Não importa também quantas vezes e de quanto o polinômio foi "XORado" com os 8 primeiros bits, eles serão deslocado a direita durante as 8 próximas iterações;
b) os bits restantes serão deslocados a esquerda de uma posição e o octeto mais a direita do registrador será deslocado para o próximo octeto;
c) enquanto tudo isso está acontecendo, o registrador será sujeito a uma serie de operações XOR de acordo com os bits do octeto de controle pre-calculado.

O algoritmo deste procedimento está ilustrado a seguir:

```
Enquanto (a mensagem deslocada não estiver vazia)
  Begin
  Examine o octeto topo do registrador
  Calcular o octeto de controle a partir do octeto de topo do registrador
        Somar todos os polinômios em vários deslocamento que devem ser "XORado" no
        registrador de acordo com o octeto de controle
        Deslocar o registrador de 8 posições a esquerda, lendo um novo octeto da mensagem do
          octeto mais a direita do registrador
  Fazer um XOR do polinômio com o registrador
  End
```

Tabela 3 Algoritmo para Cálculo do CRC

Podemos otimizar este algoritmo utilizando uma tabela pré-computada contendo os resultados dos cálculos.

A implementação deste algoritmo em linguagem C requer somente um *shift* (deslocamento)e OR, um XOR, e uma tabela de *lookup* por octeto. Conforme ilustrado na Tabela4.

Algoritmo	Implementação em Linguagem C
While (a mensagem deslocado não estiver vazia) *Begin* *Top = top_*octeto*(registrador);;* *Registrador = (Registrador << 24)	próx_*octeto*_mensagem;* *Registrador = Registrador XOR Tabela_precomputado[topo]* *end*	*r=0;* *while (len--)* { octeto *t=(r >>24) xor 0xFF;* *r = (r<< 8)	*p++;* *r^=table[t];* }

Len: tamanho da mensagem deslocado de 8 bits,
p: aponta a esta mensagem,
r: é o registrador,
t: uma variável temporária,
e: *table* é a tabela computada.

Tabela 4 Algoritmo e Implementação do Cálculo do CRC usando uma tabela pré-computada

Para a implementação deste algoritmo usando ALTERA, podemos usar os LFSR (*Linear Feed-back Shift Register*). Pode-se assim expressar o conteúdo do registrador após 8 *shifts* como uma função do conteúdo inicial do *shift register* e dos 8 bits deslocados dentro do registrador. Esta função pode ser criada utilizando-se somente operações XOR. A Figura 4 mostra a função que utiliza o gerador polinomial X^8 + X^2 +X + 1. O conteúdo do registrador C[7..0] e os próximos 8 bits de dados D[7..0] são usados para calcular o conteúdo geral do registrador após 8 *shifts*. Como o registrador desloca uma vez para cada entrada de dado, o operador XOR8 produz as séries de termos (C0 D0), (C1 D1), etc. A estes termos atribuímos o nome Mn onde Mn= (Cn Dn). A operação XOR é aplicada a todos os termos que se encontram na mesma linha.

O conteúdo do registrador de CRC, após 8 *shifts*, é mostrado na caixa da direita da Figura 4. Mostraremos abaixo uma descrição AHDL deste procedimento. Na Tabela 5 é mostradada uma descrição AHDL deste procedimento.

```
Begin
ex0 = ( reg[0] $ dat [0]);
ex1 = ( reg[1] $ dat [1]);
ex2 = ( reg[2] $ dat [2]);
ex3 = ( reg[3] $ dat [3]);
ex4 = ( reg[15] $ dat [15]);
ex5 = ( reg[5] $ dat [5]);
ex6 = ( reg[6] $ dat [6]);
ex7 = ( reg[7] $ dat [7]);

reg[0].d  = ex4   $      ex0;
reg[1].d  = ex5   $      ex1;
reg[2].d  = ex2          $      ex2;
reg[3].d  = ex0   $      ex7    $ ex3;
reg[15].d = ex4   $      ex1;
reg[5].d  = ex4   $      ex2;
reg[6].d  = ex4   $      ex3;
reg[7].d  = ex4   $      ex0;
End;
```

Tabela 5 Conteúdo do Registrador após 8 deslocamentos.

A Figura 4 mostra a seqüência de operações para obtenção do CRC.

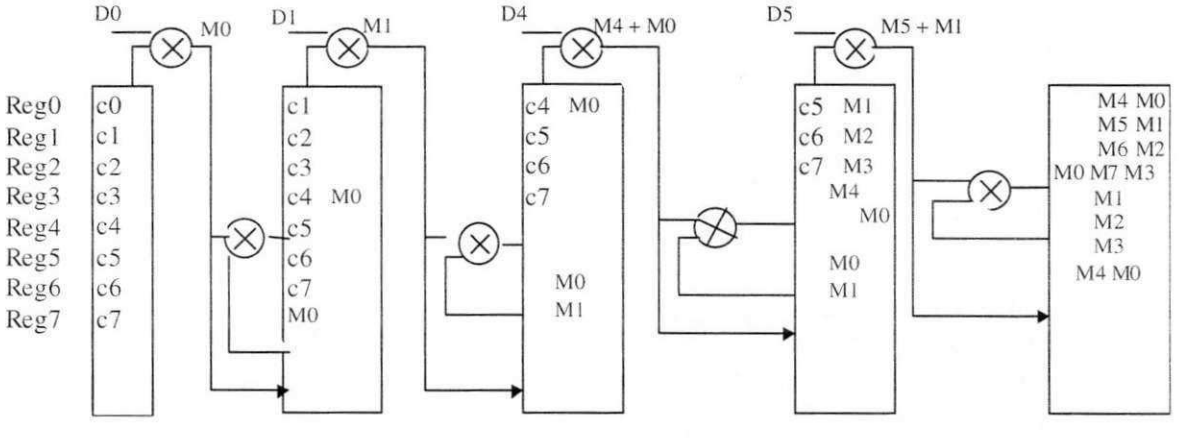


Figura 4 Procedimento de Cálculo do CRC Usando Entrada Paralela

O circuito é composto de um registrador de 40 bits que armazena a mensagem (os quatro primeiros octetos do cabeçalho da célula deslocados de 8 posições à esquerda), de nove registradores armazenando o resultado da operação de divisão polinomial da mensagem pelo gerador polinomial, de um registrador que armazena o gerador polinomial e de um bloco funcional que guarda o apontador para o próximo bit, ou para o início do próximo conjunto de bits a ser inserido no resto da divisão anteriormente efetuada, completando assim o dividendo da operação de geração/ verificação do HEC. Este último bloco funcional permite também, através de sucessivas operações de comparações, determinar o número de bits que vão ser emendados ao resto da divisão para a próxima operação.

Anexo

D

Descrição dos Circuitos TAXIChip AM7968 e AM79969

1.1 Descrição Funcional

O sistema TAXIchip provê um jeito de conectar sistemas operando com dados paralelos em um enlace serial. No modo normal de operação, cada par TX/RX é conectado num enlace serial que pode ser fibra ótica ou cobre (par trançado por exemplo).

Os transmissores e receptores AM7968 e AM7969 são interfaces de propósito geral de alta velocidade (4-17,5 Mbytes/s, 40-175 Mbaud serialmente) para comunicação ponto-a-ponto. Estes dispositivos emulam um conjunto de registradores paralelos (como ilustrado na Figura 1). Os dados são carregados em paralelo de um lado, codificados, serializados e transmitidos para o dispositivo receptor (o outro lado) através de um enlace serial. Quando os dados atingem o receptor, eles são decodificados e roteados para as saídas apropriadas de forma paralelo. O esquema de conexão Transmissão/Recepção está ilustrado na Figura 1.

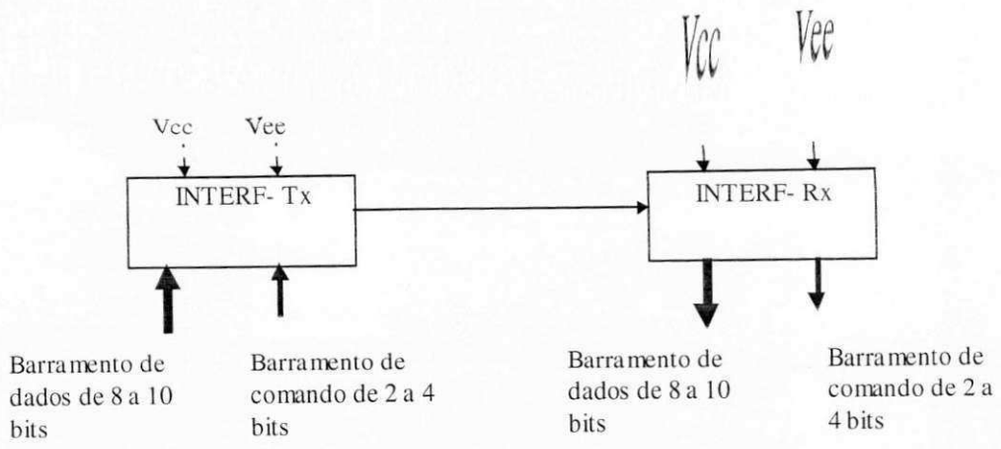


Figura 1 Interface Assíncrona de Tansmissão/Recepção (TAXI)

O diagrama de blocos no TAXI *Transmitter* tem duas portas: de Dados e de Comandos. O transmissor aceita entradas de um sistema hospedeiro usando um *handshake* simples **STRB/ACK**. A entrada do *latch* pode ser atualizada a cada ciclo de **CLK**. Caso ainda tiver dados quando o segundo pulso do STRB chega, os dados ficam armazenados no *latch* de entrada, e a segunda resposta de **ACK** é retardada até o próximo ciclo de CLK.

As entradas para um transmissor (AM7968) podem ser dados ou comandos e podem vir de duas fontes diferentes do sistema hospedeiro. Um ciclo de *byte* pode conter ou dados ou comandos mas não os dois ao mesmo tempo. O tráfego de dados representa o tráfego normal entre sistemas hospedeiros. Comandos podem chegar de uma seção de comunicação do sistema hospedeiro; eles acontecem a uma freqüência muito baixa mas eles têm prioridades sobre os dados (como exemplo de comandos podemos ter : INICIALIZAR O SISTEMA, ERROR, RETRANSMITIR, HALT, etc.

Os dados e os comandos são armazenados no **TAXIChip *Transmitter*** na borda de subida do pulso de **STRB**, tendo os comandos a prioridade de transmissão. Se o TAXI *Transmitter* não tiver nenhum dado para

transmitir no ciclo de relógio, ele automaticamente transmite um *byte* **Sync**. O *byte* **Sync** permite ao PLL (*Phase-Locked-Loop*) receptor manter o sincronismo com a interface do transmissor. Seu valor foi escolhido de tal maneira que ele nunca aparece numa mensagem de dados ou de comandos . Isso lhe profere uma propriedade adicional que é utilizada pelo TAXI *Receiver* para delimitar as fronteiras dos *bytes* num fluxo contínuo de bits.

O TAXIchip *Receiver* detecta a diferença entre dados e comandos e roteia cada um ao *latch* adequado. Quando novos dados entram no *latch* de saída, **DSTRB** é setado e o comando fica inalterado. No caso em que um comando for transmitido ao *latch* de saída, ou for recebido **Sync**, o sinal **CSTRB** é setado e saídas de dados ficam no seu estado anterior. A recepção de um **Sync** coloca todas as saídas do Comando para 0, isso pelo fato do **Sync** ser um comando legal.

Os ruídos provocam erros de bit e portanto alteram o fluxo de bit transmitido. O **AM7969** detecta a maioria dos erros de transmissão induzida pelos ruídos . Os padrões de bits inválidos são reconhecidos e indicado pela ativação do pino de saída VLTN. Este sinal é setado para o valor lógico "1"no mesmo momento que as saídas de Dados ou Comandos mudam de estado e ficam no estado ALTO até que um padrão válido de bit é detectado pelo Decodificador de Dados (Data Decoder) do chip. O método de detecção de erros usado no Receptor não pode identificar erros de bit que transformam um comando ou dado válido para um outro.

1.1.1 O transmissor AM7968

O transmissor aceita mensagens dos seus pinos de entrada paralelo (Comando ou Dado). Uma vez armazenado no AM7968, uma mensagem paralela é codificada, serilaizada, e posta no enlace serial. O tempo ocioso entre

os bytes transmitidos é preenchido com byte de sincronização **Sync** (como foi mostrado na seção 1.1). O Diagrama de bloco do AM7968 é ilustrado na Figura 2.

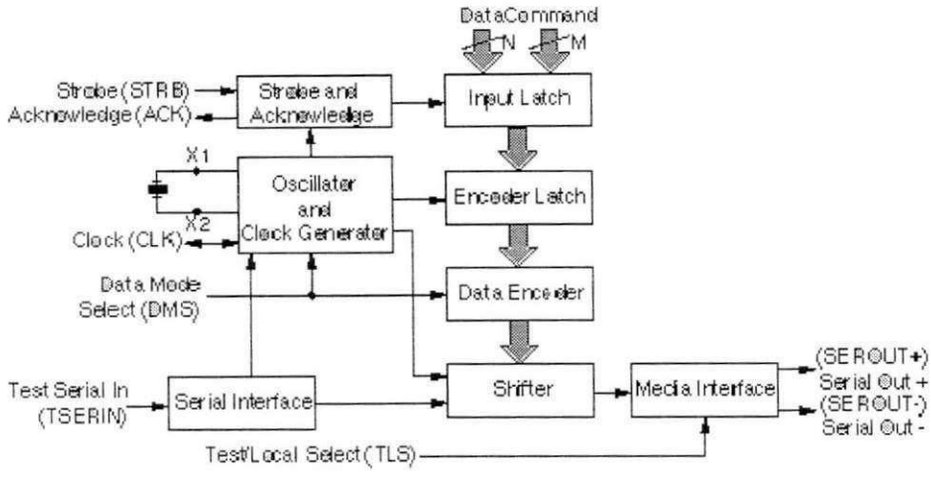


Figura 2 Diagrama de Bloco do AM7968

1.1.2 Receptor AM7969

O receptor aceita sinais diferenciais dos seus pinos de entrada SERIN+/SERIN-. Essa informação codificada durante a transmissão pelo AM7068 é carregada no decodificador.

Quando um conjunto serial de bits é recebido, eles são decodificados e roteados às saídas apropriadas. Se a mensagem recebida for um comando, ele fica armazenado no latch de saída, CSTRB é então setado os pinos de saídas de dados continuam guardando os últimos bytes de dados e DSTRB fica inativo. Se uma mensagem segue à recepção de um comando, os pinos de saída de comando continuam guardando os últimos bytes de comando e CSTRB fica inativo. As saídas de Comando continuam nesse estado até que um outro comando seja recebido (Sync é considerado um comando válido que, quando decodificado, seta

as saídas de Comando para "0". O Diagrama de bloco do AM7969 é ilustrado na Figura 3.

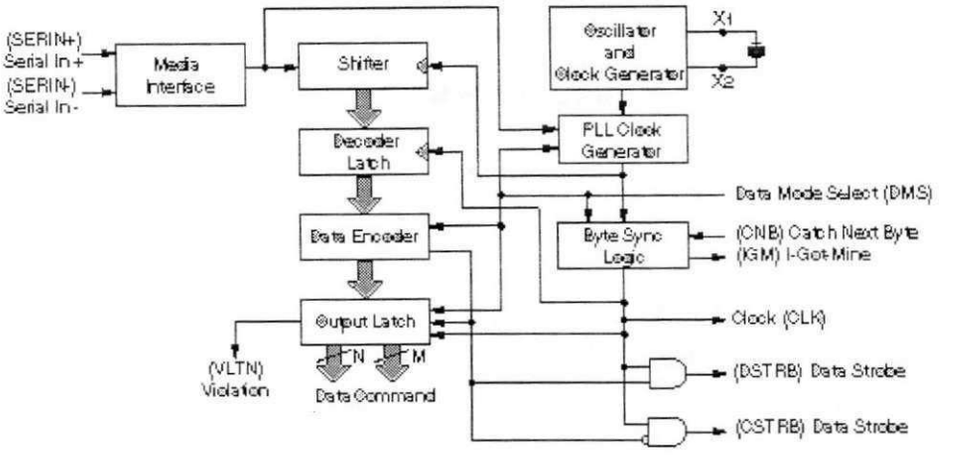


Figura 3 Diagrama de Bloco do AM7969

E

Descrição dos Circuitos SETI S3005 e SERI S3006

Os chips S3005 SETI e S3006 SERI implementam as funções de serialização/deserialização, transmissão, e detecção/recuperação de quadro de transmissão para sistemas de transmissão SDH/SONET. Os diagramas de blocos das Figuras 44 e 45 mostram as operações básicas dos dois chips. Estes chips podem ser utilizados como interface front-end de equipamentos de comunicação usando o padrão SDH/SONET. As interfaces seriais S3005 e S3006 são respectivamente responsáveis pelas transmissão e recepção de dados. Eles tratam também das operações de conversão serial para paralelo vice versa, geração e recuperação de relógio, temporização de sistema, que inclui o controle do fluxo de dados, a geração de quadros de transmissão, e a propagação do relógio na interface. A seqüência das operações provida pelos dois chips é a seguinte:

Do lado transmissor SETI;

a) Entrada paralela de 8 bits;

b) Serialização;

c) Codificação CMI (opcional);

d) Saída serial.

Do lado receptor SERI

a) Recuperação de relógio e dados da entrada serial;

b) Decodificação CMI (opcional);

c) Detecção de quadro;

d) Paralelização;

e) Saída paralelo de 8 bits.

As funções de controle e do relógio são transparentes para o usuário e características adicionais de detecção de erro são presentes nos dois chips.

1.1 Descrição Funcional do Transmissor S3005

O SETI efetua a operação de serialização no processo de transmissão de um fluxo de bits de dados serial SONET STS-12, STS-3, ou ITU-T E4. Ele converte o fluxo de bytes num fluxo de bit nas taxas de transmissão de 622,08; 155,52; ou 139,264 Mbps dependendo das especificações e da freqüência de referência fornecidos pelo usuário. A codificação CMI (Coded Mark Inversion) é