Pontifícia Universidade Católica do Rio Grande do Sul
Faculdade de Informática
Pós-Graduação em Ciência da Computação

Camadas de Adaptação ATM para Transferência de Dados e Voz

Sheila Moreira Souza

Dissertação apresentada como requisito parcial à obtenção do grau de mestre em Ciência da Computação

Orientador: Prof. Ney Laert Vilar Calazans

Porto Alegre, janeiro de 2003

Para minha mãe... que é a grande responsável pelas minhas vitórias...

Agradecimentos

Ao Prof. Ney Calazans, pela orientação, tão importante durante os dois anos do curso.

Aos membros da banca, por compreenderem os imprevistos e estarem presentes.

À Prof. Lúcia Maria Giraffa, pelos momentos de descontração, pelos conselhos, e pela sua campanha: "Alimente os mestrandos", imprescindível nas tardes de sexta-feira.

Ao Prof. João Batista de Oliveira, pela paciência e incansável atenção, pelas inúmeras tardes solucionando meus problemas no LaTeX.

Ao bolsista Luís Henrique Ries, pelo seu esforço e sua dedicação, que foram fundamentais na etapa final do trabalho.

Aos meus colegas do Mestrado, pela união, nos momentos de alegria e nos de indignação, pelas festas e churrascos... pela amizade que agora eu tenho por cada um, e que eu espero conservar pra sempre...

À minha amiga Carol, por entender a falta de e-mails ou telefonemas naqueles períodos em que a gente parece sumir do mundo... e por estar sempre presente...

Aos meus avós, Antão, Linda e Maria, por perdoarem a falta de visitas e as vezes até mesmo de notícias... e por rezarem sempre por mim...

À minha mãe, pelo seu amor incondicional, pelo seu total apoio, e pelas palavras de incentivo e de carinho que me acompanham sempre e que me fazem lembrar que os obstáculos não existem para nos derrubar, e sim para serem superados...

... e especialmente ao Leonardo, por todo seu amor... por me levantar quando eu estava caindo, me fazer sorrir quando eu começava a chorar, me dizer palavras de conforto quando em meu pensamento só haviam mágoas... por me fazer feliz e por ser esse homem maravilhoso, que eu quero ter comigo por toda a minha vida...

Abstract

Multimedia services are increasingly used in electronic products such as cellular phones and handhelds. These services introduce the need to combine distinct information flow types keeping a quality of service adequate to real time applications. Such flows normally include digitalized voice, digital images and data. The advance of both the ATM technology and the techniques for information encoding and compression to offer multimedia services the keep the quality of services for a wide range of applications. For most applications however, maintaining the quality of service is far from being a simple task, at least from the point of view of implementation. Distinct, complex techniques must be used to encode and transmit each information flow, and for each of these to be efficient, the transmission network must be adapted to transmit each information flow type. The ATM technology supports such specificity at the equivalent to the network level of the OSI reference model, the set of ATM adaptation layers or AALs.

This work presents a study about the simultaneous transfer of voice and data in ATM networks at the AAL level. The various AALs are explored in detail and the use of the AAL 5 and AAL 2 to respectively transmit data and voice is justified. This occurs in a given applications context, generated in the scope of an industry cooperation. Selected modules of the chosen AALs were developed and validated. Alternative implementations for the AAL 2 in hardware and software have been conducted and some initial results of comparing these implementations are advanced. For the voice processing, a transmission efficiency study is presented based on the computation of the timer-CU parameter and its effect on the maximum load and associated packaging density. The integration of the AALs with the ATM layer is also discussed.

Resumo

Serviços multimídia estão sendo cada vez mais utilizados em produtos tecnológicos tais como telefones celulares e computadores de mão (*handhelds*). Estes serviços introduzem a necessidade de combinar fluxos de informação de diferentes naturezas mantendo uma qualidade de serviço adequada a aplicações de tempo real. Tais fluxos compreendem dados, voz digitalizada e imagens digitais, estas últimas às vezes sob a forma de vídeo em tempo real.

O avanço da tecnologia ATM e das técnicas de codificação e compressão de informações possibilitam oferecer serviços multimídia com a qualidade esperada. Contudo, para a maioria aplicações, a manutenção da qualidade de serviço não é uma tarefa simples do ponto de vista de implementação. Técnicas distintas e complexas devem ser usadas para codificar e transmitir cada um dos tipos de fluxo de informação. Para que cada uma destas técnicas possa ser eficiente, a rede de transmissão deve ser adequada a cada um dos tipos de fluxos transmitidos. A tecnologia ATM suporta tal especificidade no nível de camadas de adaptação ATM *ATM Adaptation Layers* ou AALs.

Este trabalho apresenta um estudo sobre a transferência simultânea de dados e voz em redes ATM no nível de AALs. Explora-se em detalhe as diferentes camadas de adaptação ATM e justifica-se o uso das camadas de adaptação tipo 5 e 2 para transmitir dados e voz, respectivamente. Isto ocorre em um dado contexto de aplicações, fornecido no escopo de uma cooperação com empresa. Módulos selecionados das AALs escolhidas foram desenvolvidos e validados. Implementações alternativas para a AAL 2 em hardware e software foram conduzidas, fornecendo-se alguns resultados iniciais de comparação destas implementações. Para a transmissão de voz, apresenta-se um estudo de eficiência de transmissão, baseado em cálculos de determinação do parâmetro *timer-CU* e seu efeito sobre a carga máxima da AAL 2 e a densidade de empacotamento associada. A integração das AALs com a camada ATM de protocolos, imediatamente inferior às AALs, é igualmente apresentada.

Sumário

Lista de Tabelas

Lista de Figuras

Lista de Símbolos e Abreviaturas

RDSI-FL	Redes Digitais de Serviços Integrados de Faixa Larga	1
B-ISDN	Broadband Integrated Services Digital Networks	1
ISDN	Integrated Services Digital Network	1
FDDI	Fiber Distributed Data Interface	1
DQDB	Distributed Queue Dual Bus	1
RDSI-FE	Redes Digitais de Serviços Integrados - Faixa Estreita	1
ATM	Asynchronous Transfer Mode	1
QoS	Quality of Service	2
ITU-T	International Telecommunication Unit - Telecommunication Standarization Sector	2
VOD	Video on Demand	2
IDL	Interactive Distance Learning	2
MPEG	Moving Pictures Expert Group	3
VBR	Variable Bit Rate	3
AAL	ATM Adaptation Layer	3
CBR	Constant Bit Rate	3
PCM	Pulse Code Modulation	4
ADPCM	Adaptive Pulse Code Modulation	4
FPGA	Field Programmable Gate Array	5
LAN	Local Area Network	8
MAN	Metropolitan Area Network	8
VCI	Virtual Channel Identifier	8
VPI	Virtual Path Identifier	8
RS-232	Recommended Standard 232	9
SYNC	Syncronization	9
SOM	Start of Message	9
EOM	End of Message	9
STM	Synchronous Transfer Mode	9

CLR	Cell Loss Ratio	10
Max-CTD	Maximum Cell Transfer Delay	10
P2P-CDV	Peak-to-peak Cell Delay Variation	10
CER	Cell Error Ratio	10
SECBR	Severely Errored Cell Block Ratio	10
CMR	Cell Misinsertion Rate	11
HEC	Header Error Check	12
SAR	Segmentation and Reassembly Sublayer	13
CS	Convergence Sublayer	13
CPCS	Commom Part Convergence Sublayer	13
SSCS	Service Specific Convergence Sublayer	13
UNI	User to Network Interface	13
NNI	Network to Network Interface	13
OAM	Organization, Administration and Maintenance	13
PT	Payload Type	14
CLP	Cell Loss Priority	15
GFC	Generic Flow Control	15
VCC	Virtual Channel Connection	16
VCL	Virtual Channel Link	16
VPC	Virtual Path Connection	16
VPL	Virtual Path Link	16
PVC	Permanent ou Provisioned Virtual Circuits	16
SVC	Switched Virtual Circuits	16
VP	Virtual Path	17
VC	Virtual Connection	17
PM	Physical Medium Sublayer	17
TC	Transmission Convergence Sublayer	17
UTOPIA	Universal Test & Operations PHY Interface for ATM	17
PHY	Physical Layer	17
SDH	Synchronous Digital Hierarchy	17
PDH	Plesiochronous Digital Hierarchy	17
HDLC	High-Level Data Link Control	18
SONET	Synchronous Optical NETwork	20
SDU	Service Data Unit	20

GCRA	Generic Cell Rate Algorithm	21
TC	Tabela de Comutação	22
SAP	Service Access Points	23
LP	Loss Priority	23
CI	Congestion Indication	23
UU	User-to-user	23
rt-VBR	Real-Time Variable Bit Rate	25
nrt-VBR	Non Real-Time Variable Bit Rate	25
ABR	Available Bit Rate	25
UBR	Unspecified Bit Rate	26
GFR	Guaranteed Frame Rate	26
IP	Internet Protocol	27
SAAL	Signaling AAL	28
PCI	Protocol Control Information	28
PDU	Protocol Data Unit	28
SN	Sequence Number	29
CSI	Convergence Sublayer Indication	30
SC	Sequence Count	30
SNP	Sequence Number Protection	30
CRC	Cyclic Redundancy Check	30
CDV	Cell Delay Variation	30
SRTS	Synchronous Residual Time Stamp	30
ST	Segment Type	31
MID	Multiplexing Identification	31
LI	Length Indication	31
BOM	Begin Of Message	32
COM	Continuation Of Message	32
EOM	End Of Message	32
SSM	Single Segment Message	32
CPI	Commom Part Indicator	32
BTag	Beginning Tag	32
BASize	Buffer Allocation Size Indication	32
PAD	Padding	33
AL	Alignment	33

ETag	End Tag	33
CPS	Common Part Sublayer	33
AUU	ATM-User-to-User Indication	34
SLP	Submitted Loss Priority	34
RLP	Received Loss Priority	34
UUI	User-to-User Indication	35
MAAL	Management AAL	35
CID	Channel Identifier	35
LI	Length Indicator	35
PAD	Padding	36
STF	Start Field	36
OSF	Offset Field	36
P	Parity	37
SEAL	Simple and Efficient Adaptation Layer	38
ID	Interface Data	41
M	More	41
IDU	Interface Data Unit	42
RS	Reception Status	44
DSL	Digital Subscriber Line	45
DSP	Digital Signal Processor	46
MAC	Medium Access Control	46
CSMA/CD	Carrier Sense Multiple Access with Collision Detection	46
ADSL	Asymmetric Digital Subscriber Line	48
SDSL	Symmetric Digital Subscriber Line	48
HDSL	High-bit-rate Digital Subscriber Line	48
VHDL	VHSIC Hardware Description Language	51
VHSIC	Very High Speed Integrated Circuits	51
HDL	Hardware Description Language	51
WR	Write Enable	54
FSM	Finite State Machine	55
DV	Data Valid	60
PH	Packet Header	75
CPS-PH	CPS Packet Header	75
Timer_CU	Timer Combined Use	101

Capítulo 1

Introdução

A transmissão analógica já não é o mais alto nível de desenvolvimento tecnológico. Há alguns anos a tecnologia digital vem tomando cada vez mais espaço no ramo das telecomunicações, por ser mais confiável, mais eficiente e, principalmente, mais econômica. Juntamente, foram surgindo os chamados *sistemas multimídia*, que integram diferentes tipos de informação (vídeo, voz, áudio, dados, etc.); e as *redes de alta velocidade*, responsáveis pela transmissão dessas informações.

Os sistemas de transmissão tradicionais, tais como os oriundos de serviços de telefonia, haviam sido projetados para o tráfego de informações de tipo específico, o que dificulta, e muito, a transmissão de sistemas multimídia, com voz, vídeo e áudio integrados. A solução proposta para ultrapassar essas dificuldades foi investir no desenvolvimento das *Redes Digitais de Serviços Integrados de Faixa Larga (RDSI-FL)*, do inglês *Broadband Integrated Services Digital Networks (B-ISDN)*. Essas redes permitem o tráfego de diferentes tipos de informação, além dos já suportados pela ISDN (*Integrated Services Digital Network*).

As diferentes características e limitações das redes de alta velocidade como FDDI, DQDB e RDSI-FE (Redes Digitais de Serviços Integrados - Faixa Estreita) levaram à conclusão de que Redes ATM (*Asynchronous Transfer Mode*) são a forma mais promissora para a construção de redes de serviços integrados [SOA95].

1.1 Redes de Serviços Integrados

As redes de serviços integrados surgiram pela necessidade de transmitir mais de um tipo de informação ao mesmo tempo. As redes tradicionais, especializadas em um único serviço, não eram mais satisfatórias, pois haviam sido projetadas para um tipo de tráfego e adaptavam-se mal a outros. A integração de áudio, vídeo, voz e dados impulsionou o desenvolvimento de redes de alta velocidade capazes de transportar todos esses tipos de informação de forma eficiente, dando origem à B-ISDN.

Um dos problemas em transportar vídeo, áudio, voz e dados usando a mesma infraestrutura é que esses serviços apresentam diferentes características e exigem, cada um deles, um conjunto distinto de requisitos do sistema. Por exemplo, quando transporta-se áudio, a perda de alguns bits durante a transmissão pode não significar um problema. Ao pensar-se em termos de uma conversa telefônica, um pequeno ruído não faz com que se perca o entendimento da informação transmitida. No caso de transmitir-se alguma informação de maneira compactada, como pode ocorrer na transmissão de vídeo, a perda de alguns bits pode impossibilitar a descompatação do vídeo no destino, gerando a perda total da informação transmitida. O controle da Qualidade

de Serviço (QoS) nos diferentes tipos de tráfego deve ser considerado em qualquer rede que almeje a integração de serviços. Aspectos relacionados à Qualidade de Serviço em redes ATM são explorados mais detalhadamente em [CAS01].

Ao avaliar Redes ATM, percebe-se que estas suportam de forma adequada a diversidade de serviços definidos para as redes de serviços integrados, satisfazendo os requisitos de tráfego e transmitindo informações a taxas elevadas. Isso fez com que a tecnologia ATM fosse determinada como base para a implementação da RDSI-FL.

1.2 Modelo de Referência da B-ISDN

O ITU-T [UNI91] define o *Modelo de Referência para Protocolos da B-ISDN* conforme a Figura 1.1, baseada em [SOA95].

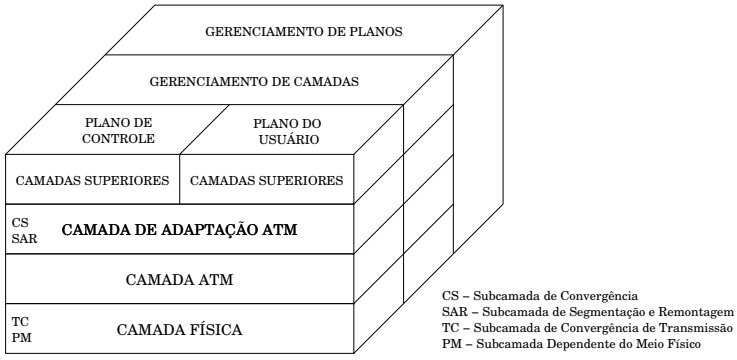


Figura 1.1: Modelo de referência para protocolos da B-ISDN.

O modelo de referência consiste em três planos: um de usuário, um de controle e um de gerenciamento. Segundo [SOA95], o plano de usuário é responsável pela transferência das informações dos usuários, enquanto que o plano de controle desempenha funções de controle, como sinalização para estabelecimento e manutenção de conexões. O plano de gerenciamento tem duas funções: a gerência de planos (de usuário, de controle e do próprio plano de gerenciamento) e a gerência de camadas (que trata dos fluxos de dados e das operações de manutenção de cada camada).

O plano de usuário é dividido em três camadas, além das camadas superiores, que não são objetivo deste trabalho. As três camadas são: camada física, camada ATM e camada de adaptação ATM. Elas constituem a arquitetura das redes ATM e serão oportunamente detalhadas.

1.3 Motivação

Há alguns anos vem surgindo um grande número de serviços multimídia, como por exemplo vídeo sob demanda (VOD - *Video on Demand*), estudo interativo à distância (IDL - *Interactive Distance Learning*), e outros. Hoje, a tecnologia ATM, as técnicas eficientes de compressão e outros desenvolvimentos na área de telecomunicações tornam possível oferecer esses serviços.

Uma das principais características do fluxo de dados multimídia é sua natureza contínua. Dados multimídia embutem um comportamento temporal que naturalmente fixa os limites de atraso dentro dos quais a informação deve ser distribuída. Isso significa dizer que, considerando

vídeo ou áudio em sua forma original, deve-se realizar a transmissão sem falhas e respeitando um certo limite de tempo. Observando o caso particular de um filme, o espectador deseja que uma impressão de movimento natural seja passada durante o filme. Logo, não é desejável uma transmissão onde haja um intervalo significativo entre os quadros.

O maior problema de se transmitir informações multimídia em sua forma original talvez seja a elevada largura de banda necessária para uma transmissão contínua. Assim, várias técnicas de compactação e compressão de dados são largamente utilizadas em transmissões desse tipo. A técnica conhecida como MPEG (*Moving Pictures Expert Group*) é largamente aceita para compressão e empacotamento de dados multimídia [KIM97]. Como esta, existem outras tantas, umas mais adequadas a vídeo, outras a áudio. Essas técnicas, de um modo geral, possibilitam a transferência das informações em conexões de tráfego com taxa variável (VBR), com os dados sendo enviados e recebidos em rajadas. Isso diminui a largura de banda ocupada em transmissões multimídia. Um estudo comparativo entre as diferentes técnicas pode auxiliar futuras implementações de AALs para transmissão multimídia ou na escolha da mais adequada para um determinado tipo de áudio ou vídeo.

As redes ATM, adotadas como solução para a B-ISDN, têm a vantagem de poderem manipular conexões com taxa de bits constante (CBR) tão bem quanto as com taxa de bits variável (VBR). Embora as conexões CBR sejam mais fáceis de gerenciar, as conexões VBR podem utilizar a largura de banda da rede de maneira mais eficiente, através do emprego de multiplexação estatística.

O ITU-T definiu diferentes tipos de AALs, cada uma para um tipo específico de tráfego (CBR, VBR, etc.), conforme já foi estudado e apresentado em [SOU01]. A AAL é responsável por receber os pacotes de dados da aplicação e particioná-los em blocos de 48 bytes, para que a camada ATM os transmita à camada física após inserir um cabeçalho (5 bytes), gerando células ATM. Conseqüentemente, no sentido contrário, cabe às AALs receber esses blocos de bytes e remontar os pacotes da aplicação. Segundo o ITU-T, a camada de adaptação definida para tráfego VBR é a AAL 2 [UNI97]. Mas alguns pesquisadores, em particular [ADA97], alegam que existe uma tendência de que a AAL 2 direcione-se apenas a comunicações *wireless*. Esse, entre outros fatores, está motivando estudos e implementações de novas AALs para transmissão multimídia, a maioria baseada na AAL 5, como pode ser visto em [ADA97], [LUN93], [MEH97] e [THA95]. Tais implementações estão sendo feitas apenas em *hardware* ou combinando *hardware* com *software*, o que caracteriza o emprego de técnicas de projeto integrado de software e hardware, ou *hw/sw codesign*.

Devido ao crescimento da complexidade de sistemas digitais, muitos desses sistemas estão hoje empregando soluções de *hardware* e *software* operando conjuntamente. Segundo [ADA96], *hw/sw codesign* é uma tentativa de integrar técnicas de projeto de *hardware* e *software* com o objetivo de incorporar mais de um processo de projeto de sistema em uma única metodologia de projeto.

1.4 Objetivos

Como objetivos estratégicos do trabalho destacam-se os seguintes:

- Adquirir conhecimentos sobre as técnicas de transmissão de voz e dados, usando Redes ATM.
- Compreender as diversas classes de implementações existentes de camadas de adaptação ATM para transmissão de voz e dados, identificando seus prós e contras.

- Compreender os compromissos de *hardware* e *software* envolvidos em implementações de camadas de adaptação ATM para voz e dados.

Para o presente trabalho, os objetivos específicos são:

- Proport implementações homogêneas e heterogêneas de *hardware* e *software* de camadas de adaptação ATM que suportam voz e dados.
- Projetar camadas de adaptação ATM responsáveis pela transmissão de voz e dados, usando arquiteturas homogêneas e heterogêneas de *hardware* e *software*.
- Avaliar a qualidade dos projetos para condições de contorno específicas.

1.5 Revisão Bibliográfica

Segundo [COM01], é preciso dispor de uma largura de banda de 4000 Hz para ser possível transportar voz em algum meio de forma inteligível. De acordo com o Teorema de Nyquist, são necessárias amostras com o dobro da freqüência mais alta de um sinal para que seja possível reconstituir o sinal original, após este ser transmitido em algum meio. Assim, voz digitalizada não compactada requer que sejam obtidas amostras 8000 vezes por segundo, ou uma amostra a cada $125\mu s$. Além da taxa de amostragem, outro fator importante do projeto de sistemas que transmitem voz é a escolha da faixa de valores inteiros a ser utilizada na representação digital do sinal analógico. É preciso estabelecer um compromisso entre a exatidão e o tamanho dos dados a ser transmitido. O padrão PCM (*Pulse Code Modulation*) [UNI93] utiliza valores entre 0 e 255, com o qual se transmite um byte a cada $125\mu s$, gerando uma taxa de transmissão digital de 64 kbps. Uma variante, conhecida como ADPCM (*Adaptative Pulse Code Modulation*) [UNI90] envia um seqüência de diferenças em lugar de valores absolutos. Isso diminui o tamanho dos dados enviados, permitindo taxas como 40, 32, 24 ou 16 kbps para transmissão de voz. Considerações gerais sobre transmissão de voz utilizando a AAL 2 são discutidas em [LIU99]. Os autores estabelecem um modelo de Cadeias de Markov para analisar o processo de empacotamento da AAL 2. Nesse artigo também é proposto um algoritmo para o cálculo do valor do `Timer_CU`, um parâmetro importante na AAL 2 para atingir uma determinada eficiência na transmissão, a ser definido na Seção 7.3.

Por outro lado, os pacotes de dados gerados pela camada superior são, em geral maiores do que uma PDU. No contexto deste trabalho, que será melhor discutido no Capítulo 4, a camada superior é um adaptador Ethernet, que gera pacotes Ethernet variando entre 64 e 1518 bytes. Isso faz com que mesmo o menor pacote de dados precise ser particionado em mais de uma AAL5_PDU. Além disso, dados raramente possuem restrições de atraso e variação do atraso, como ocorre com os pacotes de voz. Isso tudo faz com que não seja necessário o uso de parâmentros de controle complexos, deixando a implementação da AAL 5 mais simples.

A escolha da AAL 2 para este projeto também foi apoiada por artigos como [MCL97a] e [MCL97]. McLoughlin e O'Neil apresentam em [MCL97a] uma visão geral sobre a camada de adaptação ATM tipo 2. São discutidas características da AAL 2 e algumas vantagens em relação à AAL 1. Também são feitas considerações sobre a eficiência do protocolo e da largura de banda utilizada para transmissão de voz. Em [MCL97], McLoughlin e Mumford fazem uma comparação entre a AAL 1 e a AAL 2 para voz. O artigo mostra que, na maioria dos casos, a AAL 2 com tráfego VBR é mais eficiente do que a AAL 1 com tráfego CBR. Para algumas aplicações, contudo, a AAL 1 pode mostrar-se mais adequada, como por exemplo

para interconexão a um provedor de serviços da Internet. A referência [PET95] apresenta uma avaliação da eficiência da AAL 2 para transporte de voz em redes ATM. Nesse artigo, os autores determinam o número máximo de fontes de voz a uma determinada taxa de bits que podem ser suportadas por uma certa largura de banda sem violar os requisitos de qualidade de serviço para as AALs 1, 2 e 5. Eles concluem que a AAL 2 pode suportar até cinco vezes mais usuários de voz do que as AALs 1 e 5. São propostos modelos para as AALs e para as fontes de voz, e a avaliação é feita através de simulação. Em [SRI99] são observadas a variação no atraso e a probabilidade de perda de pacotes AAL 2 em aplicações de voz. Esse artigo traz uma base teórica breve sobre sistemas de voz que utilizam pacotes AAL 2, e defende o uso de números de seqüência nos cabeçalhos desses pacotes com o objetivo de minimizar os atrasos e as perdas. Em [NAN02] é avaliada a eficiência da largura de banda no transporte conjunto de voz e dados através da AAL 2, quando utiliza-se serviços rt-VBR, comparando-se com a utilização de serviços CBR.

O uso da AAL 5 na transmissão de dados é uma abordagem clássica e está baseada principalmente na literatura. Autores como [SOA95], [DUT95], [HAN95], [PRY95], [RAH98] e [TAN94] exploraram o assunto e fornecem base teórica suficiente para justificar a escolha da AAL 5 neste trabalho.

Nas referências [QIG01] e [EIL00] são apresentadas algumas implementações de camadas de adaptação ATM, ou funções da mesma, utilizando FPGAs (*Field Programmable Gate Arrays*). Em [EIL00] é discutida a implementação de uma AAL para transmissão de dados em tempo real com uma alta taxa de bits, em um FPGA Altera. Em [QIG01], é mostrada a implementação do processo de segmentação e remontagem da AAL 5 em um FPGA Xilinx.

1.6 Estrutura do Trabalho

O restante deste trabalho está dividido em capítulos, com a seguinte estrutura:

Capítulo 2 - Redes ATM: São descritos conceitos básicos, características e arquitetura de redes ATM, abordando em mais detalhe as camadas física e ATM.

Capítulo 3 - Camada de Adaptação ATM: São apresentados aspectos gerais a respeito da camada de adaptação ATM. Também são abordadas as AALs tipo 1 e 3/4 de maneira breve e as AALs 2 e 5 em detalhes.

Capítulo 4 - Contexto da Aplicação: Alguns conceitos relevantes ao trabalho realizado são apresentados neste capítulo, contextualizando o estudo de caso utilizado.

Capítulo 5 - Implementação da AAL 5: Neste capítulo são apresentados os módulos de *hardware* desenvolvidos para a transferência de dados através da AAL 5. Apenas os módulos da recepção são detalhados.

Capítulo 6 - Implementação em *Hardware* da AAL 2: São descritos os módulos de *hardware* implementados para a transferência de pacotes de voz através da AAL 2. Assim como a AAL 5, apenas os módulos da recepção são detalhados.

Capítulo 7 - Implementação em *Software* da AAL 2: Neste Capítulo são descritos os módulos de *software* implementados para transferência de voz. Os comentários aqui referem-se tanto à transmissão quanto à recepção de pacotes de voz através da AAL 2.

Capítulo 8 - Integração dos Módulos com a Camada ATM: Os módulos de *hardware*, tanto da AAL 2 como da AAL 5, foram integrados com o módulo, também de *hardware*, da camada ATM, e os resultados são comentados neste capítulo.

Capítulo 9 - Conclusões e Trabalhos Futuros: São apresentadas as conclusões obtidas do presente trabalho propõem-se trabalhos futuros relacionados ao tema.

Capítulo 2

Redes ATM

2.1 Características das Redes ATM

As características das redes ATM mais relevantes para este trabalho são as seguintes:

- são orientadas à conexão
- permitem operações não-orientadas à conexão
- unem facilidades do chaveamento de pacotes com chaveamento de circuitos
- transmitem de maneira assíncrona
- garantem transmissão em seqüência
- possuem condições de controlar a qualidade do serviço prestado pela rede
- permitem controlar a perda ou o descarte de células
- possuem controle de congestionamento
- possuem capacidade de policiamento da taxa de entrada
- não garantem a integridade dos dados
- devem ser implementadas sobretudo em *hardware*
- são fortemente padronizadas

As seções a seguir apresentam um breve comentário sobre cada uma dessas características.

2.1.1 Rede Orientada à Conexão

Redes ATM são orientadas à conexão. Isto significa dizer que deve-se estabelecer uma conexão virtual entre dois ou mais equipamentos terminais antes de se transmitir as informações desejadas.

Uma conexão só pode ser aceita se existirem recursos suficientes na rede disponíveis para estabelecer a conexão fim-a-fim, de acordo com os requisitos de qualidade de serviço. Nenhuma das conexões já existentes pode ser afetada em termos de qualidade pelo estabelecimento de uma nova conexão.

Existem mecanismos de controle de admissão de conexão disponíveis em redes ATM que determinam se a conexão que está sendo solicitada pode ou não ser estabelecida. Mais detalhes podem ser encontrados em [CAS01, HAN95, RAH98].

2.1.2 Operação Não-Orientada à Conexão

Assim como a comunicação orientada à conexão, a B-ISDN também suporta comunicação sem conexão, ou *connectionless*. A disponibilidade desse tipo de serviço é importante porque uma das aplicações na B-ISDN é interconectar LANs ou MANs, e muitas tecnologias de redes locais utilizam, hoje em dia, serviços não-orientados à conexão. Dessa forma, mesmo sendo orientadas à conexão, as redes ATM disponibilizam operações sem conexão.

Nesse tipo de operação, a primeira célula transmitida carrega o endereço de rede do destinatário em seu campo de carga útil (*payload*). As células seguintes que pertençam ao mesmo usuário não trazem esse endereço, mas são ligadas à primeira pelo mesmo VPI/VCI.

2.1.3 Facilidades de Chaveamento (Pacotes *versus* Circuitos)

O *chaveamento de circuitos* é a tecnologia dominante nos dias atuais para comunicação de voz e dados [RAH98]. Esse tipo de chaveamento implica em existir um caminho de comunicação dedicado entre os dois terminais. Uma comunicação através de chaveamento de circuito requer três etapas:

1. estabelecimento do circuito;
2. transferência dos dados;
3. desconexão do circuito.

Por outro lado, o *chaveamento de pacotes* consiste em primeiro armazenar a informação, para só depois repassá-la ao próximo elemento da rede. O estabelecimento de um circuito não é necessário para esse tipo de chaveamento.

As redes ATM transmitem informações em pequenos pacotes de tamanho fixo, as células (detalhadas na Seção 2.3.2). Os pacotes que chegam a um elemento da rede ATM, tipicamente uma chave, detalhada na Seção 2.3.3, são totalmente recebidos para só então prosseguirem seu caminho na rede. Essa chave decide o caminho a ser seguido de acordo com os valores de VCI e VPI, ou somente VPI, dependendo do tipo de chave. Isso é uma característica do chaveamento de pacotes. Por outro lado, as redes ATM são orientadas à conexão e, por essa razão, é necessário o estabelecimento de um caminho por onde as células deverão ser transmitidas. Essa característica pertence ao chaveamento de circuitos. Assim, pode-se dizer que a tecnologia ATM integra facilidades de ambos os tipos de chaveamento, constituindo redes de características mistas de chaveamento de circuitos e de pacotes [RAH98].

2.1.4 Transmissão Assíncrona

Em sistemas de comunicação digital, existem dois tipos de *métodos de transmissão de dados*: o síncrono e o assíncrono [RAH98]. A diferença básica entre eles é quanto a transmissão ou não do *clock* junto com a informação útil (dados).

O método de transmissão *assíncrono* não transmite o *clock*. Nesse método os bytes podem ser delimitados, por exemplo, por dois bits chamados *start bit* e *stop bit*, para marcar o início

e o fim do byte, respectivamente. A Figura 2.1, baseada em [RAH98], ilustra o método de transmissão digital, usando como exemplo o padrão RS-232. As figuras seguintes desta seção também foram baseadas na mesma referência.

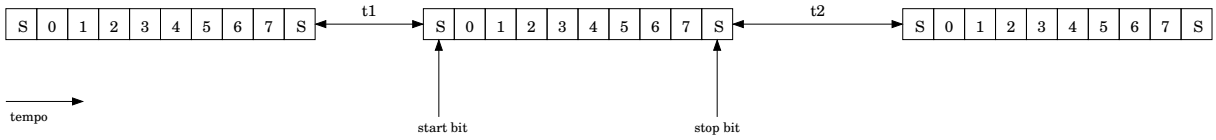


Figura 2.1: Transmissão no método assíncrono. As transmissões são delimitadas por bits, chamados de *start bit* e *stop bit*. Os tempos entre essas transmissões, indicados por `t1` e `t2` não são especificados e não precisam ser iguais.

Por outro lado, o método de transmissão *síncrono* inclui informações de temporização (*clock*) associada à transmissão dos dados. O receptor e o transmissor usam o mesmo *clock* para a transmissão, ou derivam o sinal de *clock* com uma certa tolerância de frequência a partir dos dados transmitidos. As mensagens são delimitadas por alguns caracteres de controle responsáveis pela sincronização, indicação de início e final de mensagem e controle de erros. Os métodos de transmissão síncrona, ainda segundo [RAH98], são mais adequados a operações em alta velocidade. A Figura 2.2 mostra uma transmissão pelo método síncrono.

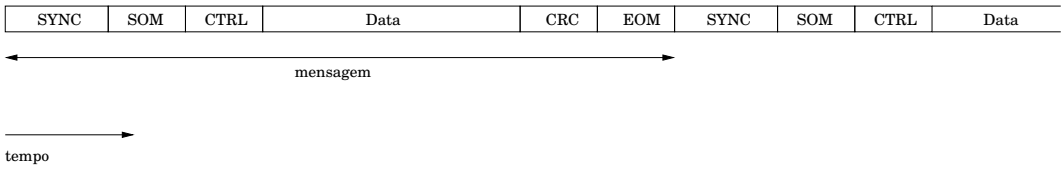


Figura 2.2: Transmissão no método síncrono. A mensagem inicia com um código de sincronização (`SYNC`), uma indicação de início de mensagem (`SOM` - *Start of Message*) e outros caracteres de controle, e termina com um código de verificação de erros (`CRC`) e uma indicação de fim de mensagem(`EOM`).

Quando utiliza-se o método de transmissão síncrono, pode-se trabalhar em dois *modos de transmissão* distintos: o modo assíncrono (ATM) e o modo síncrono (STM). Em uma rede ATM, o método síncrono está associado ao modo assíncrono, com a informação sendo transmitida em células. No modo de transmissão *síncrono* (STM), os dados são transferidos em quadros (*frames*), sobre os quais é imposto um mecanismo de sincronização. O modo ATM tenta eliminar as limitações do STM, obtendo vantagem nos serviços de taxa variável e mantendo um desempenho satisfatório com serviços de taxa constante.

Ao contrário do STM, um canal de comunicação não é dividido em *slots* de tempo para a transmissão da informação. A principal desvantagem do STM é que, se o canal não estiver em uso, sua capacidade é desperdiçada. A Figura 2.3 mostra uma transmissão no modo síncrono. No ATM, esse tempo pode ser destinado a outro canal, porque as células ATM carregam no cabeçalho uma identificação do canal relacionado à conexão virtual que foi estabelecida para a transmissão, e podem ser enviadas sem respeitar qualquer posição fixa no tempo, como mostrado na Figura 2.4.

Mas, considerando um tipo de tráfego contínuo, em que sempre há um canal transmitindo, observa-se que o STM apresenta um *overhead* menor em relação ao modo ATM. Isso verifica-se porque o modo ATM coloca um cabeçalho de 5 bytes a cada 48 bytes de informação

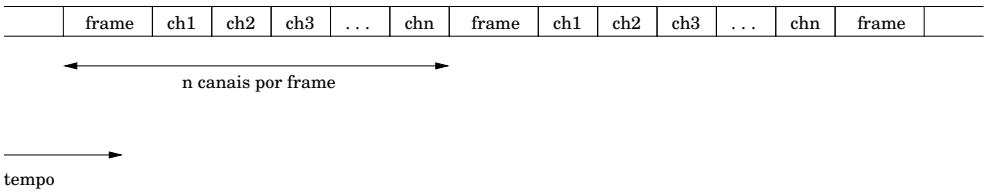


Figura 2.3: Transmissão no modo síncrono.

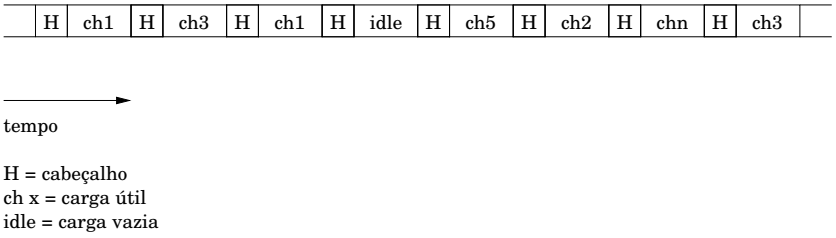


Figura 2.4: Transmissão no modo assíncrono.

(totalizando uma célula de 53 bytes), enquanto que, por exemplo, o STM-1 [UNI93a] utiliza um cabeçalho de 9 bytes para cada 261 bytes de informação (formando um *frame* de 270 bytes).

2.1.5 Garantia de Transmissão em Seqüência

As células transmitidas por um equipamento terminal em uma rede ATM através de uma conexão virtual serão recebidas no outro extremo da comunicação na mesma sequência em que foram enviadas. Isso significa que o usuário não precisa se preocupar em reorganizar células que chegam fora de ordem, por que isso não ocorre. O problema disso é que as células deverão utilizar sempre o mesmo caminho para trafegarem. No caso de falha em alguma chave pertencente ao caminho, a transmissão será interrompida.

2.1.6 Qualidade de Serviço (QoS)

Cada conexão virtual tem associada a si uma qualidade de serviço. São seis os parâmetros utilizados para determinar a qualidade de serviço em uma dada conexão, mostrados a seguir. Os três primeiros podem fazer parte da negociação entre a rede e os sistemas terminais (a definição de *sistema terminal* aparece na Seção 2.3.3) como parte do contrato de tráfego, os outros três, não.

1. Proporção de células perdidas (CLR - *Cell Loss Ratio*)
2. Atraso máximo na transferência de células (Max-CTD - *Maximum Cell Transfer Delay*)
3. Variação no atraso da célula (P2P-CDV - *Peak-to-peak Cell Delay Variation*)
4. Proporção de células com erro (CER - *Cell Error Ratio*)
5. Proporção de blocos de células severamente erradas (SECBR - *Severely Errored Cell Block Ratio*)

6. Taxa de células inseridas erradas (CMR - *Cell Misinsertion Rate*)

Esses parâmetros deverão ser respeitados durante a conexão. Se a rede não puder garantir os requisitos mínimos solicitados para qualidade de serviço, então o estabelecimento da conexão deve ser negado.

2.1.7 Perda ou Descarte de Células

Eventualmente, algumas células podem ser perdidas durante o tráfego na rede ou descartadas propositalmente se, por exemplo, algum ponto da rede estiver sofrendo congestionamento. Mas, em momento algum, a rede ATM é capaz de informar ao usuário final que uma célula sua foi perdida.

O usuário pode definir prioridades para suas células, decidindo quais serão descartadas primeiro, caso seja necessário. Se o descarte de células de baixa prioridade ainda for insuficiente para descongestionar a rede, então os parâmetros de qualidade de serviço são utilizados para escolher as próximas células a serem descartadas.

2.1.8 Controle de Congestionamento

Congestionamento é um estado dos elementos da rede que a torna incapaz de garantir a qualidade de serviço das conexões já estabelecidas, e conseqüentemente de estabelecer novas conexões. Ele pode ser causado, por exemplo, pela falha de algum elemento na rede.

O controle de congestionamento em redes ATM implica em estabelecer políticas para minimizar os efeitos de congestionamentos que já tenham ocorrido e prevenir a ocorrência de outros. Os algoritmos de controle de fluxo utilizados em redes mais antigas de comutação por pacotes não podem ser aplicados em redes ATM, nem em redes de alta velocidade de uma maneira geral. A eficiência dos algoritmos não é satisfatória nesses ambientes devido a sua natureza reativa. Mecanismos de controle de tráfego reativos tentam controlar o congestionamento da rede através de pacotes de realimentação enviados ao nodo anterior. Em redes de alta velocidade, quando um nodo recebe uma mensagem de realimentação, ele já pode ter enviado uma grande quantidade de informação, que será perdida em função do congestionamento. Mais detalhes sobre isso são encontrados em [CAS01, SUD98].

2.1.9 Policiamento da Taxa de Entrada

Chaves ATM monitoram seus pontos de entrada para verificar se a taxa dos dados que chegam por uma conexão virtual está de acordo com os parâmetros de qualidade de serviço estabelecidos no momento da criação da conexão. Se a taxa estiver acima do limite, poderão ser tomadas três atitudes: descartar as células que excedem o limite; marcá-las como sendo de baixa prioridade, para que, se o descarte for inevitável, elas sejam as escolhidas; ou não tomar nenhuma atitude no momento, deixando que outro elemento da rede resolva o problema.

2.1.10 Falta de Integridade dos Dados Fim-a-fim

Redes ATM não garantem a integridade dos dados durante uma transmissão. Essa tarefa deverá ser de responsabilidade das camadas superiores à AAL. O que há é uma verificação de erros nos dados, feita pela camada de adaptação (que será tratada no Capítulo 3), onde a célula com erro é descartada, mas não recuperada.

2.1.11 Implementação em *Hardware*

As funções da rede são tipicamente implementadas em *hardware*, o que diminui o tempo perdido com o processamento de células. Com a utilização de meios físicos cada vez mais rápidos, como fibra ópticas, o gargalo na comunicação deixou de ser a propagação no meio físico e passou a ser o processamento realizado nas chaves e equipamentos terminais [HUN96]. Reduzindo esse tempo de processamento, aumenta-se o desempenho da rede.

2.1.12 Forte Padronização

Há algum tempo que órgãos como o ITU-T e o ATM Forum preocupam-se com a padronização de redes de alta velocidade, e mais especificamente de redes ATM, no caso do ATM Forum. Na realidade, o que esse órgãos fornecem são recomendações para o desenvolvimento de redes, aceitas como normas nesse assunto. Várias dessas recomendações foram consultadas para a realização deste estudo, algumas das quais encontram-se nas referências bibliográficas.

2.2 Arquitetura ATM

A arquitetura de uma rede ATM é dividida em camadas. A Figura 2.5 ilustra essa divisão.

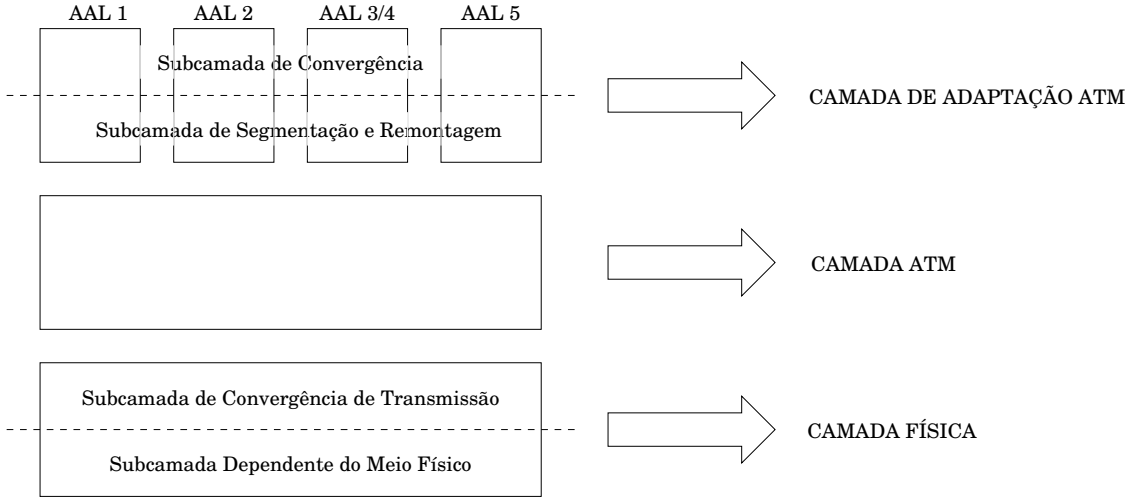


Figura 2.5: Arquitetura ATM em camadas.

O nível mais baixo é a camada física (a ser comentado na Seção 2.4), que é dividida em duas subcamadas:

- Subcamada Dependente do Meio Físico (*Physical Medium Sublayer*): é responsável pela transmissão no meio físico, entre outras funções de baixo nível de abstração.
- Subcamada de Convergência de Transmissão (*Transmission Convergence Sublayer*): é responsável por funções como o desacoplamento da taxa de células, a geração e verificação do HEC e o delineamento de células.

Acima da camada física encontra-se a camada ATM. É essa camada a responsável pelas funções de chaveamento e controle de fluxo das células ATM. A estrutura da camada ATM

é mostrada na Seção 2.5. Outros detalhes sobre gerenciamento e qualidade de serviço estão detalhados em [CAS01].

A camada seguinte é a *ATM Adaptation Layer - AAL*, ou Camada de Adaptação ATM. A AAL está presente apenas em equipamentos terminais. Existem quatro tipos de camadas de adaptação. São eles: AAL 1, AAL 2, AAL 3/4 e AAL 5. Cada AAL é responsável por um determinado conjunto de funções. Elas foram assim divididas basicamente em relação ao tipo de serviço que oferecem: se taxa constante ou variável de transmissão, se oferece operações sem conexão, e outros.

Assim como a camada física, as AALs também são divididas em subcamadas: a SAR - Subcamada de Segmentação e Remontagem e a CS - Subcamada de Convergência. Essa última ainda pode ser particionada novamente em CPCS - Parte Comum da Subcamada de Convergência e SSCS - Subcamada de Convergência Específica do Serviço, mas somente na AAL 3/4 e na AAL 5. A camada de adaptação ATM terá um destaque maior neste trabalho, principalmente a AAL 2 e a AAL 5.

2.3 Princípios Básicos das Redes ATM

2.3.1 Interfaces UNI e NNI

Existem dois tipos de interfaces definidas para redes ATM, que são:

- UNI - *User to Network Interface*
- NNI - *Network to Network Interface* (ou *Network to Node Interface*)

A *UNI* é a interface entre um equipamento terminal e uma chave ATM. Pode-se fazer uma comparação com a interface entre um telefone e a central. Por outro lado, a *NNI* é uma interface entre chaves ATM. Tanto a UNI como a NNI podem ser classificadas em públicas ou privadas, de acordo com o tipo de chave que compõe a interface.

A Figura 2.6 mostra a estrutura de uma rede ATM. Neste caso, UNI refere-se à interface entre um equipamento terminal e uma chave, sem analisar se ela é pública ou privada. Da mesma forma, a NNI também não depende da natureza das chaves. Uma explicação mais aprofundada a respeito de UNI e NNI privadas e públicas pode ser encontrada em [CAS01, RAH98].

2.3.2 Célula ATM

A transmissão de informação em redes ATM é baseada em pacotes de tamanho fixo, denominados *células*. Historicamente, a definição do tamanho de uma célula foi cercada de algumas discussões entre companhias norte-americanas e européias [HUN96]. De um lado, os americanos queriam um tamanho de 64 bytes, com mais 6 bytes adicionais de cabeçalho, mais apropriado para transmissão de dados. De outro, os europeus, defendendo um tamanho de 32 bytes, com outros 4 de cabeçalho, mais adequado à transmissão de voz. A solução foi um equilíbrio entre as duas preferências: 48 bytes para informação útil e 5 bytes para cabeçalho.

As células podem ser provenientes do usuário do serviço, quando contêm informações das camadas superiores a serem transmitidas, ou de manutenção, quando forem geradas pelo nível físico. As células de manutenção contêm um padrão de bits específico no cabeçalho, indicando tratar-se de células OAM (*Organization, Administration and Maintenance*).

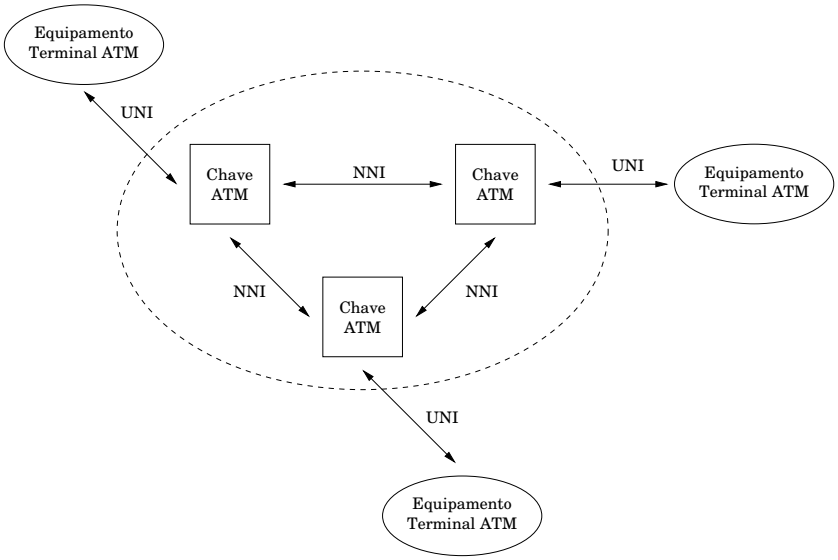


Figura 2.6: Interfaces UNI e NNI em uma rede ATM.

O formato da célula ATM é mostrado nas Figuras 2.7 e 2.8, de acordo com o padrão de interface, UNI ou NNI.

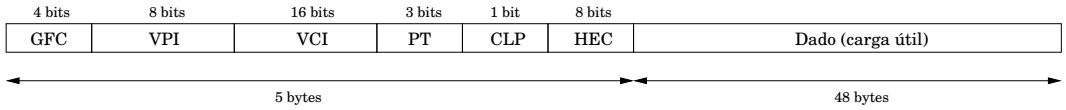


Figura 2.7: Formato de célula UNI.

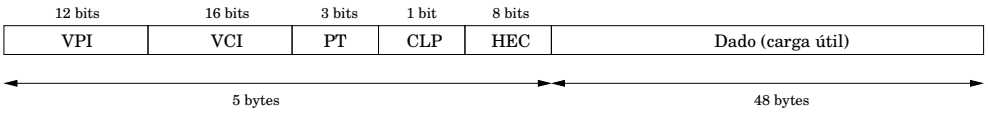


Figura 2.8: Formato de célula NNI.

Os campos do cabeçalho das células ATM são especificados a seguir:

- VPI/VCI - *Virtual Path Identifier/Virtual Channel Identifier* - Esses campos contêm informações que serão utilizadas durante o chaveamento das células.
- PT - *Payload Type* - Informa o tipo de carga que está sendo transportada pela célula. O bit mais significativo (PT2) deste campo indica quando se trata de uma célula de usuário (PT2=0) ou de manutenção (PT2=1, células OAM). No primeiro caso (se for uma célula de usuário), o bit central (PT1) informa se ela passou por algum ponto da rede que enfrentava congestionamento (PT1=1). O bit menos significativo (PT0) é utilizado pelas camadas mais altas. Por exemplo, na AAL 5 que será discutida mais adiante, utiliza-se esse bit para indicar se a célula é a última de um bloco de células do usuário (PT0=1) ou não (PT0=1).

- CLP - *Cell Loss Priority* - Indica se a célula pode ser descartada em caso de necessidade ou não. Isso é útil para, por exemplo, prevenir congestionamentos. No caso de ser necessário descartar alguma célula, as de menor prioridade serão escolhidas primeiro. Baixa prioridade é indicada por CLP=1.
- HEC - *Header Error Check* - Esse campo é utilizado pela camada física para detecção de erros múltiplos no cabeçalho ou para correção de erros simples, além de ser usado para o delineamento de células em alguns tipos de meios físicos [DUT95, SOA95].
- GFC - *Generic Flow Control* - Esse campo é definido apenas para a UNI e é utilizado para controle de tráfego [CAS01]. Na interface NNI, os bits destinados ao GFC são realocados para expandir o campo VPI.
- Dado (carga útil ou payload) - É a carga útil da célula, com as informações do usuário (no caso de células de usuário).

A Figura 2.9 mostra as camadas da rede ATM e os campos do cabeçalho que são de responsabilidade de cada uma. As camadas não foram colocadas na hierarquia correta para facilitar a compreensão da figura.

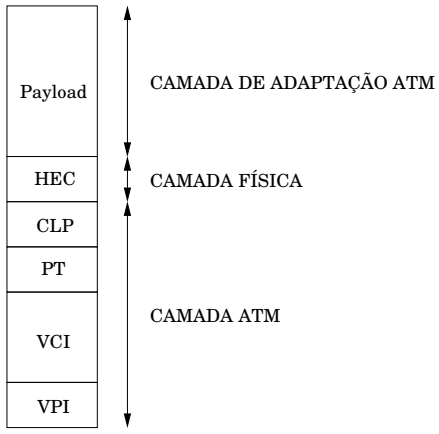


Figura 2.9: Mapeamento dos campos da célula ATM no modelo de camadas da rede ATM, estabelecendo a responsabilidade de geração dos campos.

2.3.3 Equipamentos Terminais e Chaves (*Switches*)

As chaves, (*switches*, ou comutadores) compõem a base de redes ATM, executando o transporte dos dados dentro delas. Elas são normalmente classificados em "públicas" ou "privadas". Chaves podem diferir em alguns pontos:

1. nos tipos de enlaces (*links*) suportados.
2. nos procedimentos de controle e contabilidade.
3. nos modos de endereçamento suportados.
4. no desempenho - equipamentos públicos geralmente necessitam de um desempenho maior.

Uma *chave* consiste em várias portas de entrada e de saída associadas a linhas físicas da rede. Quando uma célula é recebida através de uma das portas de entrada, a chave deverá retransmiti-la por uma de suas portas de saída. Para determinar qual delas deverá ser a escolhida, a chave necessita de informações sobre o caminho a ser seguido pela célula. Essas informações encontram-se no cabeçalho da célula (VCI/VPI).

Equipamentos terminais (*ATM Endpoints*) são as partes do equipamento do usuário que possuem interface com uma Rede ATM. Um *endpoint* envia e recebe células através de conexões estabelecidas pelos padrões ATM. Eles conectam-se à rede ATM por uma UNI (Interface Usuário-Rede).

Existem ainda os *sistemas terminais*, que consistem em um ou mais equipamentes ligados a uma rede ATM. Pode ser, por exemplo, uma LAN Ethernet ligada a uma rede ATM através de um *gateway*. Toda a estrutura da LAN Ethernet e o *gateway* compõem um sistema terminal. O sistema terminal também pode ser apenas um equipamento ATM, quando os conceitos de equipamento e sistema terminal se equivalem.

2.3.4 Conexões ATM

Em uma rede ATM, as células são transportadas através de conexões fim-a-fim denominadas *Conexões de Canal Virtual* (*VCC - Virtual Channel Connection*). Uma VCC é formada pela concatenação de *Enlaces de Canal Virtual* (*VCL - Virtual Channel Link*), que são definidos entre dois nós da rede. Em cada enlace físico da rede poderão existir vários VCLs pertencentes a diferentes VCCs.

Pode ocorrer também que várias VCCs sejam comutadas através de uma mesma saída na chave, por questões de desempenho. Neste caso, tem-se definida uma *Conexão de Caminho Virtual* (*VPC - Virtual Path Connection*), que é o conjunto de VCCs que utilizam a mesma saída. De maneira análoga, *Enlaces de Caminho Virtual* (*VPL - Virtual Path Link*) são os diferentes enlaces que juntos formam o caminho entre dois pontos. A Figura 2.10 ilustra esses conceitos.

Os valores de VCI e VPI são utilizados pelas chaves ATM para determinar por qual saída a célula deve ser enviada. De acordo com os valores de VCI/VPI, ou somente VPI, a chave escolhe um novo VCL e/ou VPL por onde a célula deve transitar.

O estabelecimento de uma conexão virtual resulta de um processo de sinalização, onde um caminho é selecionado de acordo com três fatores: a qualidade de serviço desejada, os identificadores de conexão em cada chave por onde o caminho irá passar, e os recursos da rede como, por exemplo, largura de banda. Essas conexões virtuais podem ser de dois tipos [RAH98]:

1. Circuitos Virtuais Permanentes ou Pré-disponíveis (*PVC - Permanent ou Provisioned Virtual Circuits*): são circuitos estabelecidos manualmente ou através de algum outro fator externo, por exemplo, por um protocolo de gerenciamento.

2. Circuitos Virtuais Chaveados (*SVC - Switched Virtual Circuits*): são conexões estabelecidas pelos elementos da rede, de acordo com a necessidade.

Informações mais detalhadas a respeito de conexões ATM podem ser obtidas em [CAS01, SOA95].

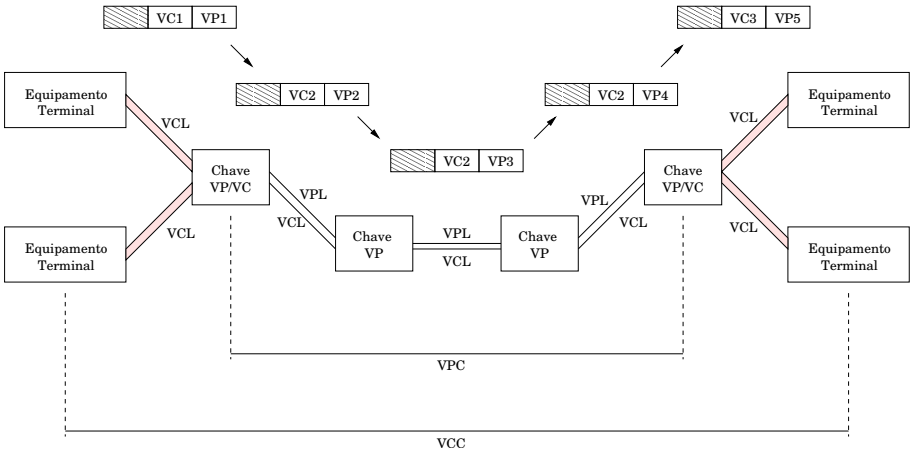


Figura 2.10: Conexões virtuais. Quando uma conexão é estabelecida, os valores de VCI e VPI são escolhidos em cada chave ATM para que as células desta conexão sejam roteadas corretamente até o destino. Esses valores são armazenados em tabelas de comutação nas chaves ATM. A célula que chega a uma chave VP/VC tem seus valores de VCI e VPI avaliados para determinar o caminho que ela deverá seguir. Ambos valores são modificados para que a próxima chave também possa tomar essa decisão. Se for uma chave VP, então apenas o valor de VPI é utilizado para determinar o caminho e alterado para a próxima chave.

2.4 Camada Física *(Physical Layer)*

A camada física está presente em todos os equipamentos da rede. Ela é dividida em duas subcamadas, que são: Subcamada Dependente do Meio Físico (*Physical Medium Sublayer - PM*) e Subcamada de Convergência de Transmissão (*Transmission Convergence Sublayer - TC*).

A Recomendação I.432.1 do ITU-T [UNI99] trata de aspectos da camada física ATM, e as seções seguintes irão discutir suas subcamadas e a interface UTOPIA, que serve para conectar a camada física à camada ATM.

2.4.1 Subcamada de Convergência de Transmissão

A Subcamada de Convergência de Transmissão realiza a transformação do fluxo de células em um fluxo contínuo de bits ou bytes para a transmissão sobre o meio físico. Para isso é feito o mapeamento das células ATM na estrutura de transmissão empregada pelo meio, que pode ser, por exemplo, SDH, PDH ou baseada em células. No lado receptor, essa subcamada é responsável pela extração das células individuais do fluxo de bits ou bytes.

As principais funções desempenhadas pela TC são as seguintes:

Geração e Inserção do HEC: Uma função importante da Subcamada de Convergência de Transmissão é a geração do campo HEC, e sua inserção no cabeçalho da célula ATM.

Correção e Detecção de Erros: No lado receptor, o campo HEC é utilizado para detectar e corrigir um erro simples (ou seja, a inversão de um bit) e detectar erros múltiplos, que podem ocorrer no cabeçalho da célula. Se algum erro for detectado, a célula recebida é descartada pela própria camada física, e não é repassada à camada ATM de destino. Como o cabeçalho da célula informa à camada ATM o que ela deve fazer com a célula que

está sendo recebida, é importante que ele seja livre de erros. Caso contrário, esses erros poderiam, por exemplo, fazer com que uma célula fosse roteada para um destinatário incorreto.

Delineamento de Células: Essa função permite determinar onde, dentro de um fluxo de bits ou bytes, começa e termina uma célula. Existem três formas para isso:

1. O transmissor inclui uma seqüência de bits delimitadores, para marcar o início e o final da célula. Por exemplo, o protocolo HDLC utiliza a seqüência "01111110" como delimitador. Como existe a chance dessa seqüência fazer parte dos dados, ainda precisam ser consideradas as técnicas de transparência de dados, como aquela chamada de *bit stuffing* [SOA95].
2. O delineamento baseado no fato de alguns sistemas de transmissão utilizarem quadros cíclicos, como é o caso dos sistemas que utilizam hierarquias digitais (PDH ou SDH) [SOA95]. Esses quadros podem ser utilizados de duas formas para delinearem células. A primeira é através da inclusão de um bit de *overhead* na estrutura do quadro, que aponte para os limites da célula. A segunda é através do posicionamento fixo das células dentro de um quadro, o que causa desperdício de capacidade, caso o tamanho das células não se ajuste ao do quadro.
3. O campo HEC é utilizado para o delineamento. O funcionamento deste procedimento é ilustrado pelo diagrama de transição de estados da Figura 2.11, baseada em [SOA95]. O receptor inicia no estado `HUNT`, monitorando uma janela de 5 bytes (tamanho de um cabeçalho). Para cada conjunto de 5 bytes, o campo HEC é verificado para constatar se corresponde a um cabeçalho ou não. Quando a seqüência correta é detectada, a máquina passa para o estado de `PRESYNC`, onde o receptor procura uma confirmação de que o sincronismo foi atingindo. A partir daí, a cada 53 bytes são verificados os últimos 5, em busca do próximo cabeçalho correto. Após receber um determinado número de cabeçalhos corretos, o receptor assume que o sincronismo foi atingido e passa para o estado `SYNC`. Se, estando ainda no estado `PRESYNC`, for recebido um conjunto de 5 bytes que não correspondam a um cabeçalho válido, o receptor assume que o sincronismo não foi atingido e retorna ao estado `HUNT`. No estado `SYNC`, se for recebido um certo número de cabeçalhos com erro, o sincronismo foi perdido e a máquina retorna ao estado `HUNT`.

Embaralhamento (*Scrambling*): Essa função embaralha os bits do campo de carga útil, para evitar longas seqüências de uns e zeros. Essas seqüências poderiam ser confundidas com o cabeçalho, quando o receptor estiver no estado `HUNT`, apresentado no item anterior.

Desacoplamento da Taxa de Células: O ITU-T e o ATM Forum divergem sobre quem seria responsável por essa função: a camada física ou a camada ATM, respectivamente. Essa função encarrega-se de enviar células ociosas, no caso do ITU-T, e não assinaladas, no caso do ATM Forum, quando a camada ATM não as estiver fornecendo. A abordagem do ITU-T deixa a camada ATM independente do meio físico, preocupando-se apenas com as operações sobre as células.

2.4.2 Subcamada Dependente do Meio Físico

A Subcamada Dependente do Meio Físico tem sua função básica associada ao meio de transmissão usado. Ela realiza funções como a sincronização de bits (ou *bit timing*) e a

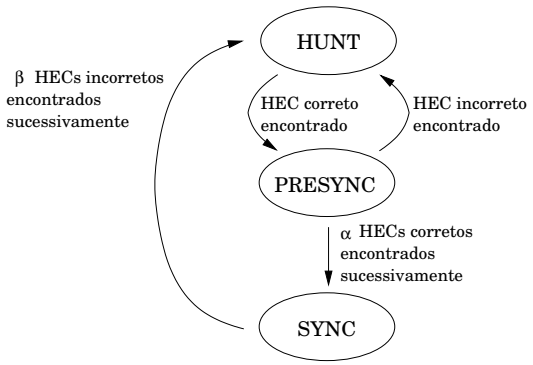


Figura 2.11: Diagrama de transição de estados para o delineamento de células baseado no campo HEC. A máquina de estados procura por um cabeçalho correto para passar do estado `HUNT` para o estado `PRESYNC`. Se logo após for recebido um cabeçalho incorreto, a máquina retorna ao `HUNT`. Em `PRESYNC`, α cabeçalhos corretos promovem a troca para o estado `SYNC`, indicando que o sincronismo foi alcançado. Em `SYNC`, β cabeçalhos incorretos promovem o retorno ao estado `HUNT`, indicando que o sincronismo foi perdido.

codificação da informação para adequar o fluxo de dados à linha de transmissão. No caso de fibras ópticas, a PM também é responsável pela conversão eletro/óptica.

2.4.3 UTOPIA - *Universal Test & Operations PHY Interface for ATM*

O ATM Forum define uma interface padrão entre a camada ATM e a camada física, chamada *UTOPIA*. Isso permite uma interface física comum em subsistemas ATM através de uma larga faixa de velocidades e tipos de dados. Essa padronização atualmente define quatro níveis para a interface UTOPIA, chamados de 1, 2, 3 e 4, cada qual com sua respectiva especificação [COM94, COM95, COM99, COM00]. Este trabalho aborda apenas os níveis 1 e 2.

A interface UTOPIA (níveis 1 e 2) consiste em dois caminhos de dados unidirecionais de 8 ou 16 bits, um para transmissão e outro para recepção. Ambos caminhos possuem sinais de *clock* independentes, assim como *handshake* a nível de célula ou a nível de bytes. A transmissão e a recepção são sincronizadas através de seus respectivos sinais de *clock*. Com um caminho de dados de 8 bits e uma taxa de *clock* máxima de 25 MHz, a interface dá suporte a taxas de até 200 Mbps. A Figura 2.12 mostra como a interface UTOPIA está inserida no contexto das redes ATM.

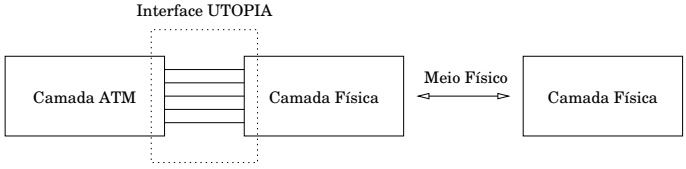


Figura 2.12: Interface UTOPIA.

A transmissão da UTOPIA a uma taxa de 200 Mbps garante que os dados serão enviados

pelo camada física respeitando as diferentes taxas de transmissão dos vários tipos de meios físicos. Alguns exemplos dessas taxas são:

- 155.52 Mbps (SONET/OC-3c)
- 155.52 Mbps (8B/10B block coded)
- 100 Mbps (4B/5B TAXI)
- 44.736 Mbps (DS-3)
- 51.84 Mbps (OC-1)

2.5 Camada ATM *(ATM Layer)*

A segunda camada do Modelo de Referência de Protocolos da B-ISDN é a camada ATM. Assim como a camada física, ela está presente em todos os elementos da rede, sejam eles equipamentos terminais ou chaves. A principal função desta camada está relacionada à geração e extração dos campos do cabeçalho da célula ATM (vista na Seção 2.3.2) Essa função, juntamente com as demais implementadas pela camada ATM serão discutidas a seguir.

A camada ATM recebe da camada de adaptação blocos de 48 bytes, chamados de ATM-SDUs. A essas SDUs são acrescentados os 5 bytes de cabeçalho, para formar a célula ATM, que será repassada à camada física. À exceção do campo *HEC*, os demais são preenchidos pela camada ATM. A comunicação também ocorre no sentido contrário, onde as células são recebidas a partir da camada física e o cabeçalho é consumido pela camada ATM antes de enviar os dados à AAL.

2.5.1 Funções da Camada ATM

A camada ATM é responsável pela transferência de qualquer tipo de informação, seja ela do usuário, de controle ou de gerenciamento. As informações do usuário são aquelas provenientes das aplicações das camadas superiores. Exemplos de controle e gerenciamento são pacotes de sinalização e de estabelecimento de conexões.

As funções da camada ATM, comentadas a seguir segundo as referências [KWO98] e [SOA95], são:

Multiplexação e Demultiplexação de Conexões: Diferentes conexões ATM com diferentes requisitos de qualidade de serviço (VPCs ou VCCs) são multiplexadas/demultiplexadas através de uma interface na camada ATM, que pode ser UNI ou NNI. Essas conexões podem estar transportando diferentes tipos de serviços. Dessa forma, a camada ATM repassa à camada física apenas um fluxo de células.

Desacoplamento da Taxa de Células (segundo o ITU-T): No lado do transmissor, essa função adiciona células não assinaladas (ou seja, células com o campo de carga útil preenchido com zeros) ao fluxo de células assinaladas (células com o campo de carga útil válido), para serem transmitidas. Isso transforma um fluxo não contínuo de células assinaladas em um fluxo contínuo de células assinaladas ou não. No receptor, as células não assinaladas são consideradas inválidas.

Discriminação de Células: O cabeçalho da célula ATM pode ser usado para indicar diferentes tipos de células, usando valores pré-definidos de VCI, VPI e PT. Por exemplo, quando trata-se de uma célula não assinalada, VCI = 0 e VPI = 0.

Geração, Preenchimento e Extração do Cabeçalho das Células: O cabeçalho de 8 bytes de cada célula é gerado e extraído pela camada ATM. Na mesma camada são preenchidos todos os campos deste cabeçalho, à exceção do campo HEC, que é preenchido e verificado pela camada física.

Indicação de Prioridade de Perda e Descarte Seletivo de Células: Essa indicação é feita através do campo CLP (*Cell Loss Priority*) no cabeçalho de célula.

Conformação de Tráfego: Qualquer elemento da rede pode realizar uma conformação de tráfego em uma VC ou em um VP. Essa função modifica as características do tráfego para que este fique de acordo com os seus descritores (usando a terminologia oficial, para que fique *conforming*). Embora a conformação de tráfego seja uma capacidade de gerenciamento de tráfego opcional, ela é fundamental para o uso eficiente dos recursos da rede [GIR98]. A rede pode atuar drasticamente sobre o tráfego que não estiver *conforming*, marcando ou descartando as células que causam esse problema, o que leva à retransmissão da informação pelas camadas superiores e, portanto, a um uso ineficiente dos recursos da rede. A conformação de tráfego utiliza algoritmos GCRA (*Generic Cell Rate Algorithm*) para verificar a conformidade das células. Outras informações sobre conformação de tráfego podem ser encontradas em [CAS01].

Controle Genérico de Fluxo na UNI: O mecanismo de GFC ajuda a controlar o fluxo de tráfego de conexões ATM na UNI da B-ISDN. Ele é usado para aliviar as condições de sobrecarga a curto prazo [HAN95].

Chaveamento de Células Baseado na Informação do Cabeçalho: Essa função é realizada nas chaves ATM e utiliza as informações contidas nos campos VCI e VPI do cabeçalho. Quando uma célula chega à chave ATM, os campos VCI e VPI em conjunto (ou somente o campo VPI, dependendo do tipo de chave) identificam o VCL utilizado pela chave anterior. Essa informação, juntamente com a porta de entrada, são utilizadas pela chave para consultar uma tabela que relaciona cada VCL e porta de entrada a um próximo VCL e uma porta de saída. Assim, a chave ATM atualiza o cabeçalho da célula e transmite a mesma pela porta de saída. As Figuras 2.13 e 2.14, baseadas em [SOA95], ilustram esse procedimento para chaves VP e VP/VC, respectivamente.

Maiores detalhes sobre chaveamento e controle de tráfego são encontrados em [CAS01].

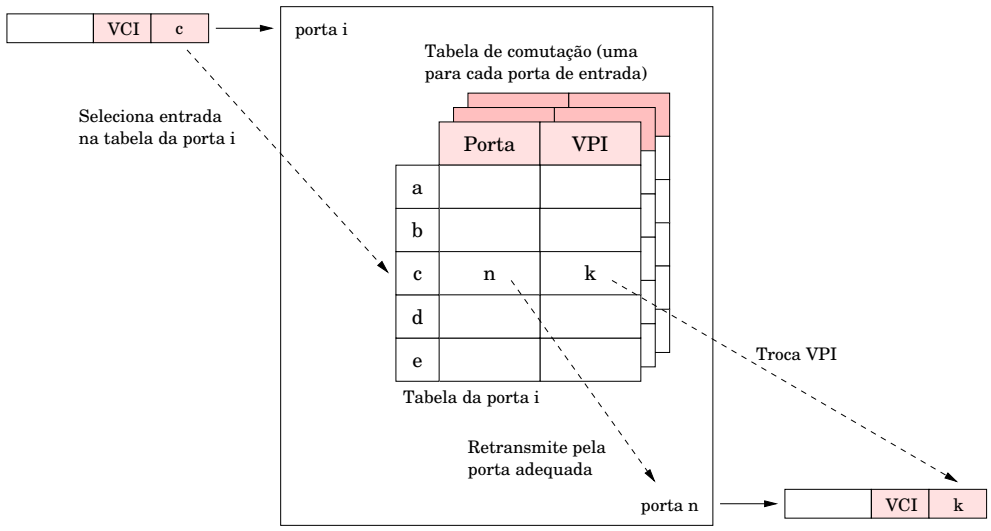


Figura 2.13: Chaveamento de células em chaves VP. A chave VP utiliza o campo *VPI* da célula que chega através da *porta i* para indexar a tabela referente à porta e determinar por qual a porta de saída a célula deverá ser retransmitida e qual o novo valor de VPI que a célula deve receber.

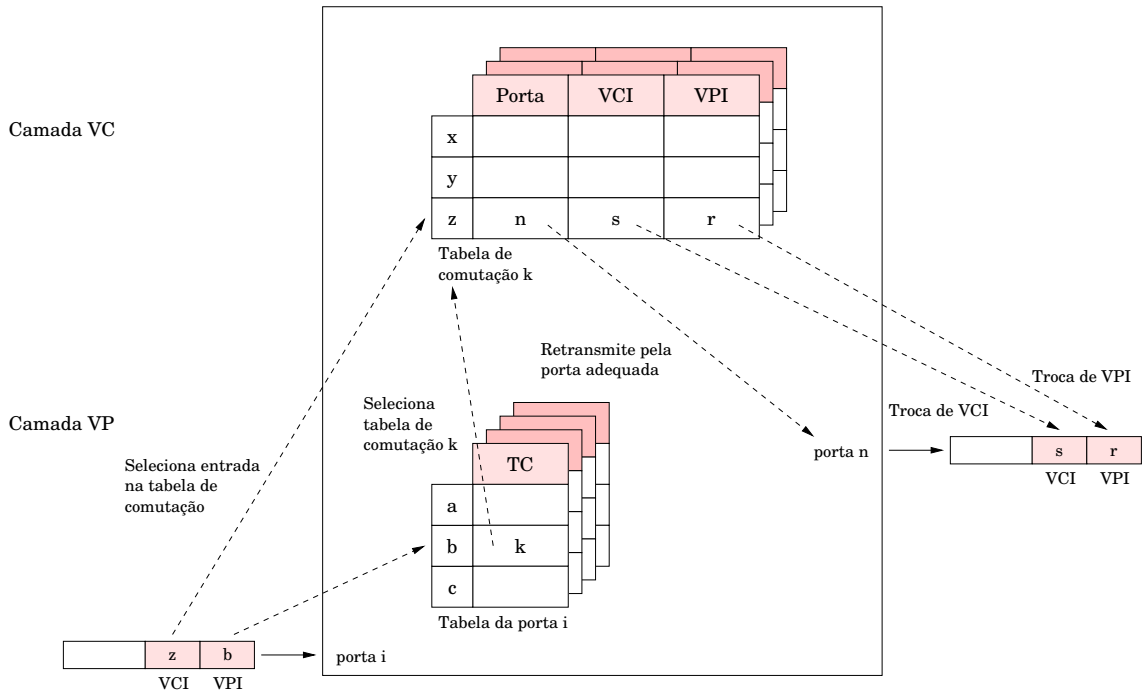


Figura 2.14: Chaveamento de células em chaves VP/VC. O primeiro passo é utilizar o valor de VPI para indexar a tabela referente à porta de entrada da chave VP/VC para determinar qual das tabelas de comutação deverá ser utilizada. O segundo passo é indexar essa tabela de comutação pelo valor de VCI para determinar por qual porta de saída a célula deverá ser retransmitida e quais os novos valores de VCI e VPI que a célula deverá receber.

2.5.2 Primitivas de Serviço da Camada ATM

A camada ATM troca informações com as camadas física e AAL através de SAPs (*Service Access Points*), que são pontos de acesso aos serviços que camadas adjacentes oferecem uma a outra. As seções seguintes tratam das primitivas de serviço entre as camadas ATM e AALs e entre as camadas ATM e física.

2.5.2.1 Primitivas de Serviço entre a Camada ATM e as AALs

Para comunicação com as AALs, existem duas primitivas definidas, que são:

- *ATM-DATA request*
- *ATM-DATA indication*

Essas primitivas utilizam os seguintes parâmetros:

ATM-SDU: Contém a unidade de informação que está sendo transmitida.

ATM-LP (*ATM Loss Priority*): Indica a prioridade de perda da informação a ser transmitida. Assume apenas dois valores, "0" para prioridade alta e "1" para prioridade baixa.

ATM-CI (*ATM Congestion Indication*): É utilizado para indicar se a célula passou por algum ponto da rede que estava congestionado.

ATM-UU (*ATM User-to-user Indication*): Esse parâmetro é passado de forma transparente pela camada ATM, podendo ser usado pelas camadas superiores à ATM para controle da informação.

A primitiva *ATM-DATA request* é invocada pela AAL, solicitando o envio de uma ATM-SDU para um outro ponto da rede, na camada correspondente, através de uma conexão existente. A primitiva *ATM-DATA indication* é utilizada para o sentido contrário da comunicação. Ou seja, ela é invocada pela própria camada ATM para indicar a chegada de uma ATM-SDU através de uma conexão existente.

2.5.2.2 Primitivas de Serviço entre a Camada ATM e a Camada Física

As primitivas trocadas entre a camada ATM e a camada física são as seguintes:

- *PHY-DATA request*
- *PHY-DATA indication*

O único parâmetro utilizado por ambas as primitivas é o chamado *PHY-SDU*, que contém a unidade de informação que está sendo transmitida.

A primitiva *PHY-DATA request* é invocada pela camada ATM para solicitar à camada física o envio de uma PHY-SDU atráves de uma conexão existente. Uma PHY-SDU (ou célula) inteira é transportada pela camada física, tendo apenas o campo *HEC* do cabeçalho alterado (como visto na Seção 2.4). A primitiva *PHY-DATA indication* é chamada pela camada física para avisar a camada ATM da chegada de uma PHY-SDU.

Capítulo 3

Camada de Adaptação ATM *(ATM Adaptation Layer)*

A camada de adaptação ATM (ou simplesmente AAL), está localizada entre a camada ATM e as camadas superiores. As funções desempenhadas nesta camada dependem dos requisitos das camadas acima dela.

A AAL é a primeira camada de protocolo fim-a-fim no modelo de referência da B-ISDN, conforme a Figura 3.1.

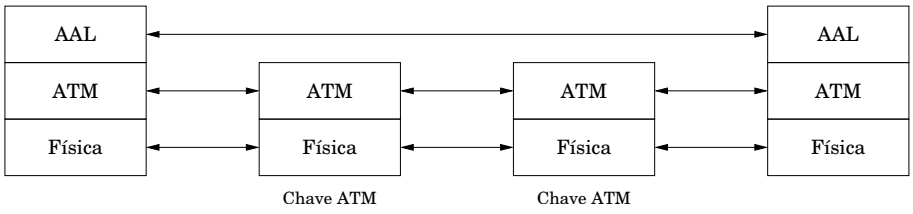


Figura 3.1: Camada de adaptação ATM fim-a-fim.

3.1 Tipos de Tráfego e Classes de Serviço

Os seis tipos de tráfego definidos pelo ATM Forum em [COM99a] são os seguintes:

- CBR (*Constant Bit Rate* - Taxa de Bits Constante): Utilizado por serviços que necessitam de quantidade estática de largura de banda, que esteja continuamente disponível durante a conexão.
- rt-VBR (*Real-Time Variable Bit Rate* - Taxa de Bits Variável - Tempo Real): É utilizado em aplicações de tempo real, isto é, aquelas que necessitam de um rigoroso compromisso de atraso e variação de atraso. É apropriado para aplicações de voz e vídeo.
- nrt-VBR (*Non-Real-Time Variable Bit Rate - Taxa de Bits Variável - Não-Tempo Real*): É utilizado em aplicações que tenham características de tráfego em rajadas.
- ABR (*Available Bit Rate* - Taxa de Bits Disponível): Esse tipo de tráfego permite que a rede modifique as características da conexão estabelecida. As larguras de banda mínima

Tabela 3.1: Classes de serviço.

Parâmetro	**Classe A**	**Classe B**	**Classe C**	**Classe D**
Sincronização entre fonte e destino	Necessária		Não necessária	
Taxa de transmissão	Constante	Variável		
Orientado à conexão	Sim			Não
Tipo de AAL	1	2	3/4 ou 5	3/4 ou 5

e máxima são informadas no estabelecimento da conexão e a banda pode variar dentro desses limites.

- UBR (*Unspecified Bit Rate* - Taxa de Bits Não Especificada): É utilizado para aplicação *non-real-time*, que não requerem compromissos obrigatórios de atraso e variação de atraso.

- GFR (*Guaranteed Frame Rate* - Taxa de Quadros Garantida): Projetado para aplicações que requerem uma taxa mínima garantida e podem aproveitar a largura de banda disponível dinamicamente na rede.

Outras informações a respeito de tipos de tráfego podem ser encontradas em [CAS01] e [COM97].

O ITU-T [UNI99a] especifica quatro classes de serviço, baseado em três critérios distintos: sincronização requerida entre fonte e destino, taxa de transmissão e modo de conexão (orientado à conexão ou não).

As quatro classes definidas foram chamadas de "A", "B", "C" e "D", conforme a Tabela 3.1 e serão apresentadas nas seções seguintes, de acordo com [DUT95].

3.1.1 Classe A

Essa clase de serviço é utilizada para aplicações de voz e vídeo com taxa de bits constante. Essas aplicações têm as seguintes características:

- Existe uma taxa de bits constante na origem e no destino.

- Existe uma relação de tempo entre a origem e o destino.

- Existe uma conexão estabelecida entre origem e destino.

A camada de adaptação que suporta esses tipos de serviços deverá disponibilizar as seguintes funções:

- Segmentação e remontagem de quadros em células.

- Manipulação da variação do atraso de células.

- Detecção e manipulação de perda, descarte, duplicação ou roteamento errado de células.

- Recuperação da freqüência de clock da origem.

- Detecção de bits errados no campo de informação.

3.1.2 Classe B

Esses serviços têm um fluxo de informações variável, necessitam de alguma relação de tempo entre origem e destino e são orientados à conexão.

A AAL que disponibilizar este tipo de serviço deverá realizar as seguintes funções:

- Transferir informações entre origem e destino a uma taxa de bits variável.
- Transferir uma relação de tempo entre origem e destino.
- Indicar a ocorrência de informações perdidas ou corrompidas não recuperadas pela própria AAL.

Manter um serviço da Classe B é complicado. Altas larguras de banda e cargas variáveis exigem estruturas de controle de taxa de transmissão complexas, devido a sua natureza imprevisível e as suas taxas de transferência de dados que podem, freqüentemente, atingir um pico percentualmente significativo em relação aos recursos de rede que estão sendo utilizados. Isso significa dizer que, a menos que a rede tenha sido corretamente projetada para suportar tráfego de serviços da Classe B, as operações poderão ser interrompidas com grande facilidade.

A camada de adaptação ATM associada a essa classe de serviço é a AAL 2, e devido a essa complexidade foi a padronização que demorou mais tempo a surgir [UNI97].

3.1.3 Classe C

Esses serviços são orientados à conexão e têm um fluxo de informações variável. A camada de adaptação para a Classe C deve disponibilizar as seguintes funções:

- Segmentação e remontagem de quadros em células.
- Detecção e sinalização de erros nos dados.

Além disso, ela também pode disponibilizar outros serviços, como multiplexação e demultiplexação de conexões entre equipamentos terminais em uma única conexão de rede ATM. Contudo, a necessidade dessa função é fortemente discutida [DUT95].

3.1.4 Classe D

Esses serviços não são orientados à conexão e têm um fluxo de informações variável. Isso é importante para suportar protocolos de rede sem conexão, como o IP, e serviços de transferência de dados caracter-a-caracter.

A AAL que disponibiliza serviços da Classe D deve executar as seguintes funções:

- Segmentação e remontagem de quadros em células.
- Detecção de erros em dados (mas não a retransmissão).

Além desses, a Classe D também engloba multiplexação e demultiplexação de múltiplos fluxos de dados em um único fluxo através da rede ATM, e roteamento e endereçamento das camadas da rede.

3.2 Tipos de AALs

O ITU-T [UNI93b] define quatro tipos de AAL, conforme já ilustrado na Tabela 3.1:

- AAL 1: disponibiliza serviços da classe A.
- AAL 2: disponibiliza serviços da classe B.
- AAL 3/4: anteriormente haviam sido definidas separadamente, mas devido à semelhança nas funções que executavam, foram unidas em uma única AAL, que disponibiliza serviços das classes C e D.
- AAL 5: também disponibiliza serviços das classes C e D, mas de forma mais simples do que a AAL 3/4

Alguns autores [DUT95, HAN95, SOA95] definem um outro tipo de camada de adaptação: a AAL 0, que não executa funções da camada AAL, apenas fornece serviços da camada ATM.

Outro tipo de AAL, padronizado, é a SAAL (*Signaling AAL - AAL de Sinalização*), utilizada para suporte a conexões de sinalização entre chaves ATM ou entre equipamentos terminais e chaves ATM.

As camadas de adaptação dos tipos 1 e 3/4 serão abordados em seções deste capítulo, enquanto que os tipos 2 e 5 terão mais ênfase e serão vistos em capítulos específicos.

3.3 Estrutura da Camada de Adaptação ATM

Quase todas as camadas de adaptação ATM (a exceção é a AAL 2) são divididas nas seguintes subcamadas:

- Subcamada de Convergência (*Convergence Sublayer - CS*)
- Subcamada de Segmentação e Remontagem (*Segmentation and Reassembly Sublayer - SAR*)

A SAR tem a finalidade de quebrar as informações que chegam das camadas superiores em segmentos que caibam no campo de informação de uma célula, para então transmitir para a camada ATM; ou receber esses segmentos e agrupá-los para repassar à camada superior.

A tarefa da CS depende do tipo de serviço disponível. Dentre as funções, pode-se ter multiplexação, detecção de perda de células e recuperação da relação de tempo entre a origem e o destino.

A informação que chega à AAL proveniente das camadas superiores recebe o nome de *SDU* (*Service Data Unit*). Neste caso é uma AAL-SDU. A camada de adaptação ATM atua sobre essa SDU juntando um cabeçalho denominado de *PCI* (*Protocol Control Information*). O resultado dessa junção é a chamada *PDU* (*Protocol Data Unit*). Esta nomenclatura segue o padrão proposto pelo ITU-T [SOA95]. A PDU é a unidade transmitida entre camadas adjacentes pertencentes a um mesmo elemento da rede. A AAL envia uma AAL-PDU para a camada ATM, que a recebe e dá o nome de ATM-SDU. No sentido oposto de comunicação, a AAL recebe uma AAL-PDU (que é uma ATM-SDU enviada pela camada ATM) e retira o cabeçalho inserido na outra extremidade da comunicação, gerando novamente uma AAL-SDU. Essa unidade é então passada às camadas superiores. A Figura 3.2 mostra o que ocorre quando as informações são enviadas de um extremo da rede a outro. As transformações de SDU em PDU, ou o contrário, serão mostradas oportunamente, nas seções ou capítulos específicos para cada tipo de AAL.

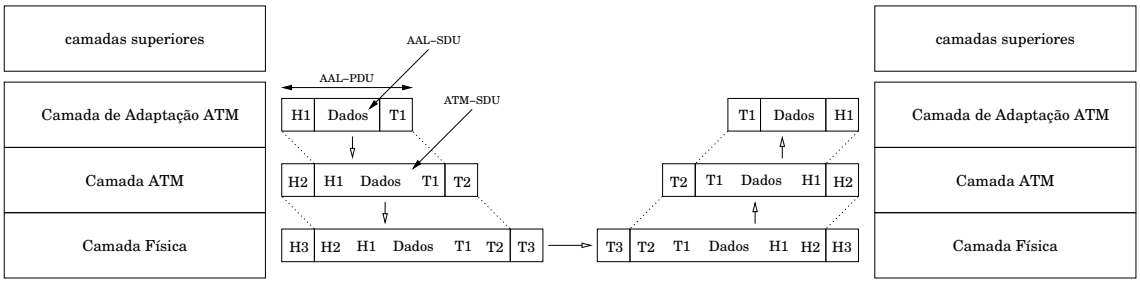


Figura 3.2: SDUs e PDUs em uma rede ATM.

3.4 Camadas de Adaptação ATM tipo 1 e 3/4

As camadas de adaptação ATM dos tipos 1 e 3/4 não estão no escopo deste trabalho. Assim, serão comentadas brevemente na seção seguinte. Por outro lado, as AALs 2 e 5, por serem mais relevantes a este trabalho, encontram-se descritas separadamente, e de forma bem mais detalhada.

3.4.1 Camada de Adaptação ATM Tipo 1

A AAL 1 é orientada à conexão e disponibiliza as funções necessárias para adaptar os serviços de tráfego CBR da Classe A para a camada ATM. Dentre esses tráfegos, tem-se sobretudo vídeo a taxa constante e voz compactada.

Os serviços executados pela AAL 1 são os seguintes:

- Transferência de SDUs com uma taxa de bits constante.
- Transferência da relação de tempo entre origem e destino.
- Transferência da estrutura de informação.
- Indicação de perda de informações, ou erros nas mesmas, que não são recuperados pela AAL.

3.4.1.1 Subcamada de Segmentação e Remontagem da AAL 1

A SAR-AAL1 recebe uma SAR-SDU com 47 bytes da Subcamada de Convergência e adiciona um cabeçalho de um byte para formar a SAR-PDU, que será repassada à camada ATM. Quando a informação vem no sentido contrário, é a camada ATM que passa à SAR um bloco com 48 bytes (SAR-PDU). A SAR-AAL1 então retira o byte de cabeçalho e transmite à Subcamada de Convergência os 47 bytes de carga útil.

Associado aos 47 bytes da SAR-SDU, a Subcamada de Segmentação e Remontagem recebe da Subcamada de Convergência um número de sequência. Quando a informação está sendo entregue à CS, esse valor é repassado a ela para que possa ser usado para detectar perda ou erros na carga útil.

Uma SAR-PDU é mostrada na Figura 3.3. Os campos do cabeçalho têm o seguinte significado:

SN - Sequence Number: Consiste nos primeiros quatro bits do cabeçalho e é subdividido em outros dois campos:

- *CSI (Convergence Sublayer Indication)* - É um campo de um bit que indica a existência da Subcamada de Convergência.
- *SC (Sequence Count)* - É um campo de três bits que contém o valor de um contador de sequência codificado em binário para ajudar na detecção de células perdidas ou com erros.

SNP - Sequence Number Protection: Consiste nos quatro últimos bits do cabeçalho e também é subdividido em outros dois campos:

- *CRC* - É um campo de três bits, calculado sobre o *Sequence Number*, que, junto com os quatro bits do SN forma uma palavra de código de sete bits.
- *Parity* - É um campo de um bit que verifica a paridade par sobre a palavra de código de sete bits que foi gerada pelo processo de CRC.

SAR-PDU Payload: São os 47 bytes de carga útil recebidos da CS. Em outras palavras, é a SAR-SDU.

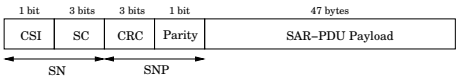


Figura 3.3: Estrutura de uma SAR-PDU para a AAL 1.

3.4.1.2 Subcamada de Convergência da AAL 1

Dependendo do serviço, a CS pode conter diferentes funções, tais como:

- Lidar com a Variação no Atraso de Células (*CDV - Cell Delay Variation*).
- Lidar com o atraso na montagem da carga útil das células (A carga útil da SAR-PDU, ou seja, a SAR-SDU, pode vir preenchida apenas parcialmente com dados recebidos da camada superior, para reduzir o atraso na montagem da carga útil).
- Recuperar o *clock* da origem no destino (Existem diversos métodos para a recuperação do *clock* da origem, mas o ITU-T recomenda o uso do método chamado *Synchronous Residual Time Stamp - SRTS* [RAH98]).
- Recuperar a estrutura de dados da origem no destino (Um ponteiro é utilizado para delinear os limites da estrutura).
- Monitorar a perda ou inserção errada de células e possivelmente corrigir esses erros.
- Monitorar bits errados no campo de carga útil e possivelmente corrigi-los.

3.4.2 Camada de Adaptação ATM Tipo 3/4

A AAL 3/4 foi projetada para transportar quadros de tamanho variável (com o limite de 65.535 bytes) de forma não orientada à conexão, sobre uma conexão previamente estabelecida (PVC) e com capacidade de detectar erros. Não existe relação de tempo entre e a origem e o destino, nem restrição quanto ao tipo de informação que irá ser transportada pela AAL 3/4.

3.4.2.1 Subcamada de Segmentação e Remontagem da AAL 3/4

A SAR da AAL 3/4 recebe da Subcamada de Convergência uma CS-PDU de tamanho variável, múltiplo de 4 bytes, e divide-a em SAR-SDUs de 44 bytes, formando a carga útil da SAR-PDU. A essa SAR-SDU são acrescentados dois bytes de cabeçalho e outros dois de *trailer* (cauda), totalizando uma SAR-PDU de 48 bytes, que será transmitida à camada ATM. A transferência também pode ocorrer no outro sentido, onde a SAR recebe da camada ATM SAR-PDUs de 48 bytes, retira os cabeçalho e os *trailers*, remonta e transmite à CS.

As funções disponibilizadas pela Subcamada de Segmentação e Remontagem da AAL 3/4 são as seguintes:

- Segmentação de CS-PDUs de tamanho variável e remontagem de SAR-SDUs.
- Detecção de erros.
- Multiplexação de várias CS-PDUs na camada ATM.

A estrutura de uma SAR-PDU é mostrada na Figura 3.4. Os campos do cabeçalho e do *trailer* são discutidos a seguir.

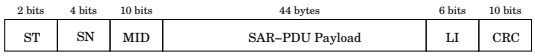


Figura 3.4: Estrutura de uma SAR-PDU para a AAL 3/4.

ST - Segment Type: É um campo de dois bits que indica qual parte da CS-PDU está sendo transportada no campo de carga útil da SAR-PDU: o início, o meio, o final ou uma CS-PDU inteira. A Tabela 3.2 mostra a codificação para esses bits.

SN - Sequence Number: São quatro bits utilizados para identificar a posição seqüencial dos blocos transmitidos. Ele ajuda na detecção de células perdidas ou fora de ordem. O valor desse campo é incrementado para sucessivas SAR-PDUs de uma CS-PDU.

MID - Multiplexing Identification: É um campo de dez bits utilizado para diferenciar entre múltiplas CS-PDUs transmitidas ao mesmo tempo. Uma única CS-PDU utiliza sempre o mesmo valor para todas as SAR-PDUs que contêm seus segmentos.

SAR-PDU Payload: É a carga útil da SAR-PDU, com 44 bytes. Nesse campo estão segmentos da CS-PDU passada à SAR.

LI - Length Indication: São seis bits que informam qual a quantidade de bytes válidos presentes na carga útil da SAR-PDU. No caso de se transmitir a última parte da CS-PDU ou uma única CS-PDU, o campo de carga útil pode conter menos do que 44 bytes, o que faz necessária a existência de algum campo que indique o número correto de bytes válidos transmitidos.

CRC - Cyclic Redundancy Check Code: É um campo de dez bits, calculado sobre toda a SAR-PDU, utilizado para detecção de erros. O cálculo do CRC é feito com base no polinômio $G(x) = 1 + x + x^4 + x^5 + x^9 + x^{10}$.

Tabela 3.2: Codificação do campo ST do cabeçalho da SAR-PDU.

ST	**Significado**	**Codificação**
BOM	*Begin of Message*	10
COM	*Continuation of Message*	00
EOM	*End of Message*	01
SSM	*Single Segment Message*	11

3.4.2.2 Subcamada de Convergência da AAL 3/4

A CS da AAL 3/4 é subdividida em duas parte: a CPCS (*Commom Part Convergence Sublayer* - Parte Comum da Subcamada de Convergência), e a SSCS (*Service Specific Convergence Sublayer* - Subcamada de Convergência Específica do Serviço). A CPCS disponibiliza uma transferência de quadros de tamanho variável (até 65.535 bytes) de forma não segura, ou seja, sem garantir que os dados chegarão livres de erros.

A CS executa as seguintes funções:

- Preservação da CS-SDU.
- Detecção de erros, descartando CS-SDUs corrompidas.
- Alocação do *buffer* máximo requerido para receber a CS-PDU.
- Abortar a transmissão de uma CS-PDU que tenha sido enviada parcialmente.

A CS recebe da camada superior um bloco de até 65.535 bytes, acrescenta um cabeçalho de quatro bytes, um *trailer* também de quatro bytes e um campo de preenchimento, para complementar o tamanho da CS-PDU, se necessário, e torná-lo múltiplo de 48 bytes. A estrutura da CS-PDU da AAL 3/4 é mostrada na Figura 3.5 e os campos do cabeçalho e do *trailer* serão explicados a seguir.

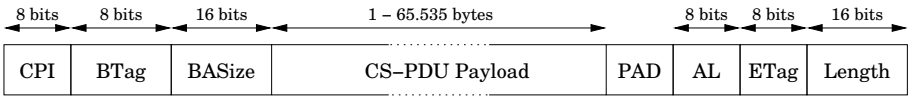


Figura 3.5: Estrutura de uma CS-PDU para a AAL 3/4.

CPI - Commom Part Indicator: Esse campo tem tamanho de um byte e indica se a contagem nos campos de *Length* e *BASize* está sendo feita em bits ou em bytes.

BTag - Beginning Tag: É um campo de um byte e está associado aos cabeçalhos e *trailers* de uma CS-PDU. Ele assegura que todas as SAR-PDUs foram recebidas corretamente, utilizado em conjunto com o campo ETag.

BASize - Buffer Allocation Size Indication: São dois bytes que contêm o tamanho máximo do *buffer* necessário para remontar a CS-PDU. Esse tamanho é expresso em bytes.

CS-PDU Payload: É a carga útil da CS-PDU e tem tamanho variável entre 1 e 65.535 bytes.

PAD - Padding: Esse campo é utilizado para preencher a CS-PDU até que ela alcance um tamanho múltiplo de 4 bytes. Ele é tipicamente preenchido com zeros e seu tamanho varia entre 0 e 3 bytes. O tamanho do campo de carga útil deve ser múltiplo de 4 para facilitar e agilizar o processamento do cabeçalho e do *trailer*, uma vez que ambos tem tamanho igual a 4 bytes.

AL - Alignment: É um campo de um byte que apenas completa o *trailer* para ele ele totalize 4 bytes.

ETag - End Tag: Esse campo consiste em um byte usado para detecção de erros através da correlação com o campo BTag. Seu valor deve ser igual ao de BTag.

Length: São dois bytes que indicam a quantidade de bytes válidos que estão sendo transmitidos.

3.5 Camada de Adaptação ATM Tipo 2 *(AAL 2)*

A camada de adaptação ATM Tipo 2 (AAL 2) é uma das padronizações mais recentes do ITU-T, através da Recomendação I.363.2 [UNI97] de setembro de 1997. A AAL 2 provê serviços da Classe B, disponibilizando uma transmissão com largura de banda eficiente, para pacotes pequenos mas de tamanho variável, em aplicações sensíveis a atrasos, a uma baixa taxa de transmissão.

Durante a recepção, a AAL 2 recebe da camada ATM as informações na forma de *unidades de serviço de dados ATM*, ou ATM-SDUs (*ATM Service Data Units*) com um tamanho de 48 byes. Na transmissão, a AAL 2 passa à camada ATM as informações na mesma forma recebida.

3.5.1 Estrutura da AAL 2

A Figura 3.6 mostra a estrutura da camada de adaptação ATM Tipo 2. Ela está dividida em duas subcamadas, chamadas *Common Part Sublayer* (CPS - Subcamada Comum a Todos os Serviços) e *Service Specific Convergence Sublayer* (SSCS - Subcamada de Convergência Específica do Serviço). Esta última pode ser nula. Ao contrário das demais AALs, não há uma definição explícita de uma subcamada de segmentação e remontagem de células. Essa função é desempenhada pela CPS, como será visto mais adiante.

A comunicação entre a AAL 2 e a camada superior ocorre através de um SAP (Service Access Point), assim como a comunicação entre a AAL 2 e a camada ATM. Entre suas subcamadas, como já ocorria nas demais AALs vistas neste trabalho, não estão definidos SAPs e a comunicação ocorre através de primitivas de serviço, abordadas a seguir.

3.5.2 Primitivas da AAL 2 para a Camada ATM

Existem duas primitivas para a comunicação com a camada ATM através de um SAP, chamadas de *ATM-UNITDATA indication* e *ATM-UNITDATA request*.

As informações passadas como parâmetros nessa comunicação estão definidas na Recomendação I.361 do ITU-T [UNI99a] e são apresentadas a seguir:

ATM-SDU: É a informação do usuário, passada em grupos de 48 bytes. Esse parâmetro é obrigatório em ambas as primitivas.

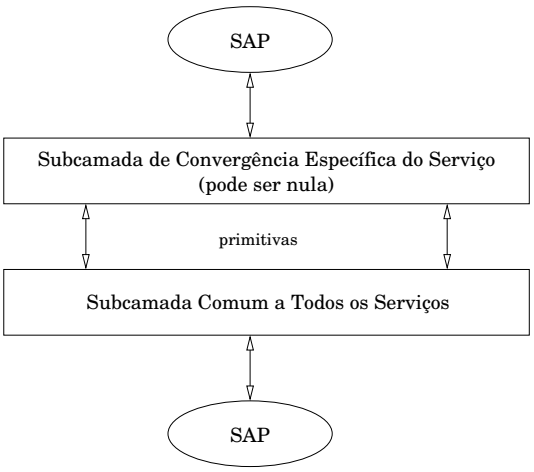


Figura 3.6: Estrutura da AAL 2.

AUU - ATM-User-to-ATM-User Indication: É um parâmetro de 1 bit que contém informações do usuário. Também obrigatório em ambas as primitivas.

SLP - Submitted Loss Priority: Se o bit de CLP (*Cell Loss Priority*) do cabeçalho da célula estiver em "1", significa que a probabilidade de que a rede ATM possa descartar uma célula é incrementada. Essa parâmetro é obrigatório apenas na primitiva *ATM-UNITDATA request*.

RLP - Received Loss Priority: Apenas na primitiva *ATM-UNITDATA indication* tem esse parâmetro como obrigatório e ele significa que o campo CLP pode ser modificado pela rede ATM de "0" para "1".

CI - Congestion Indication: Quando CI = 1, algum congestionamento foi encontrado, antes ou durante a transmissão. O parâmetro é opcional na primitiva de *ATM-UNITDATA request* e obrigatório em *ATM-UNITDATA indication*.

3.5.3 Subcamada Comum a Todos os Serviços - CPS

3.5.3.1 Serviços da CPS da AAL 2

A CPS da AAL 2 tem capacidade de transmitir CPS-SDUs de um usuário da CPS a outro, através da rede ATM. Os usuários CPS podem ser de dois tipos: entidades SSCS ou o Plano de Gerenciamento (ver Figura 1.1). Os serviços ditos *peer-to-peer* oferecidos pela CPS são os seguintes:

- Transferir CPS-SDUs de tamanho variável com valor máximo de 45 ou 64 bytes;
- Multiplexar e demultiplexar um dado número de canais AAL 2;
- Manter a integridade de seqüências de CPS-SDUs em cada canal AAL 2.

Esses serviços não são assegurados. Isso significa dizer que uma CPS-SDU inteira pode ser entregue ou perdida, e que CPS-SDUs perdidas não são corrigidas por retransmissões neste nível de protocolo.

3.5.3.2 Primitivas de Serviço entre a CPS e a SSCS

As primitivas que estão disponíveis para a comunicação entre a CPS e a SSCS da AAL 2 são: *CPS-UNITDATA request* e *CPS-UNITDATA indication*. Os parâmetros definidos para essas primitivas são os seguintes:

CPS-INFO (CPS Interface Data): Esse parâmetro especifica a unidade de interface de dados a ser transferida entre a CPS e a SSCS. Seu tamanho deve ser em bytes. Uma unidade de interface de dados contém uma CPS-SDU completa.

CPS-UUI (CPS User-to-User Indication): Esse parâmetro tem tamanho de 5 bits e possui valores apenas entre 0 e 27. Ele é transportado de forma transparente pela CPS.

Todos os parâmetros são obrigatórios em ambas as primitivas.

3.5.3.3 Primitivas de Serviço entre a CPS e o Plano de Gerenciamento

O Plano de Gerenciamento disponibiliza a primitiva *MAAL-SEND request* para avisar à CPS que é possível transmitir uma CPS-PDU. Essa primitiva não tem parâmetros.

Para permitir uma comunicação de gerenciamento chamada *peer-to-peer*, estão definidas duas primitivas para a comunicação entre a CPS e o Plano de Gerenciamento: *MAAL-UNITDATA indication* e *MAAL-UNITDATA request*. A primeira é utilizada para distribuir os dados do receptor CPS para o Plano de Gerenciamento. A segunda, para a operação contrária. São três os parâmetros, todos obrigatórios em ambas as primitivas:

CPS-INFO (CPS Interface Data): Esse parâmetro especifica a unidade de interface de dados a ser transferida entre a CPS e o Plano de Gerenciamento. Seu tamanho deve ser em bytes.

CPS-UUI (CPS User-to-User Indication): Esse parâmetro tem tamanho de 5 bits e é transportado de forma transparente pela CPS.

CPS-CID (CPS Channel Identifier): São 8 bits que contêm o CID (Channel Identifier), que identifica a conexão CPS pela qual a informação de gerenciamento está sendo transmitida.

3.5.3.4 Formato dos Dados na CPS

Os dados que chegam à CPS da AAL 2 são primeiramente empacotados nos chamados *CPS-Packets*, para só então serem colocados em CPS-PDUs. Esses últimos é que são efetivamente transmitidos pela AAL 2. Nesta Seção serão mostrados os formatos dos CPS-Packets e CPS-PDUs.

Um CPS-Packet é mostrado na Figura 3.7. Os campos do cabeçalho são comentados a seguir.

CID - Channel Identifier: Esse campo tem tamanho de 8 bits e seu valor identifica o usuário CPS do canal AAL 2. Esse canal é bidirecional e o mesmo valor de CID pode ser usado em ambas as direções. A Tabela 3.3 mostra os possíveis valores para esse campo.

LI - Length Indicator: Esse campo possui 6 bits de tamanho e é codificado com um valor binário representando o número de bytes do campo de carga útil menos 1. O tamanho máximo pré-definido é 45 bytes, mas pode ser alterado para 64.

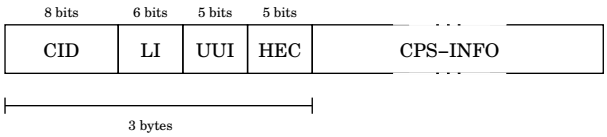


Figura 3.7: CPS-Packet da AAL 2.

UUI - User-to-User Indication: Esse campo tem um tamanho de 5 bits e dois propósitos - levar informações específicas de maneira transparente entre usuários CPS e fazer distinção entre esses usuários.

HEC - Header Error Control: O transmissor deve calcular o resto da divisão pelo polinômio x^5+x^2+1 (módulo 2), do produto de x^5 pelo conteúdo dos 19 primeiros bits do cabeçalho do CPS-Packet. Os coeficientes do polinômio resto deverão ser inseridos no campo HEC, com o coeficiente de x^4 no bit mais significativo. O receptor usa esse campo para a detecção de erros no cabeçalho.

Tabela 3.3: Valores permitidos para o campo CID.

Valor	Significado
0	Não utilizado[a]
1 .. 7	Reservados para procedimentos de gerencimento
8 .. 255	Identificação do usuário CPS

[a] Bytes preenchidos com zeros são utilizados nos campos de PAD, quando forem necessários.

Uma CPS-PDU é mostrada na Figura 3.8. Ela consiste em um byte chamado de STF (*Start Field*) e 47 bytes de carga útil (ou *CPS-Payload*). Os 48 bytes da CPS-PDU formam uma ATM-SDU.

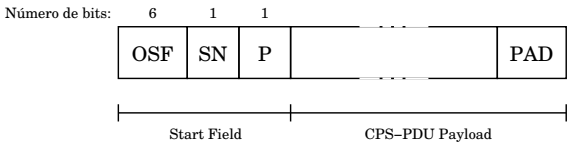


Figura 3.8: CPS-PDU da AAL 2.

O *Start Field* é subdividido em:

OSF - Offset Field: Esse campo contém um valor, codificado em binário, do deslocamento em bytes entre o final do *Start Field* e o primeiro início de um CPS-Packet (que está contido no campo de carga útil da CPS-PDU), ou na ausência de um, entre o final do *Start Field* e o início do próximo campo de PAD. O valor 47 nesse campo indica que não há início de CPS-PDU Payload nesse CPS-PDU. Isso pode ser melhor observado nas Figuras 3.9, 3.10 e 3.11.

SN - Sequence Number: Esse campo de um bit é usado para numerar o fluxo de CPS-PDUs. A norma não esclarece como se dá esta numeração, mas pode-se supor que este bit alterne valores 0 e 1 em células consecutivas, permitindo uma forma limitada de detecção de perda de células.

P - Parity: Esse campo de um bit é usado pelo receptor para detectar erros no *Start Field*. O transmissor preenche esse bit com o valor da paridade ímpar calculada sobre os bits do *Start Field*.

O campo CPS-PDU Payload pode ter um, mais de um ou nenhum CPS-Packet, completo ou parcial. A última CPS-PDU poderá conter bytes de enchimento (PAD) para completar o tamanho de 48 bytes.

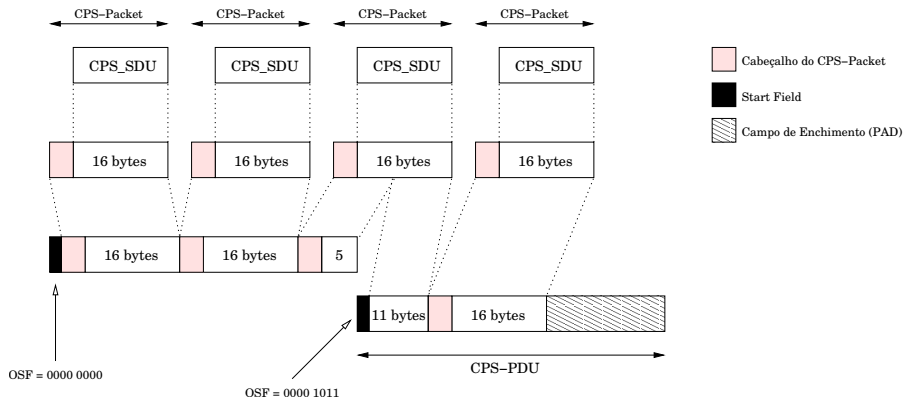


Figura 3.9: Transmissão de CPS-SDUs em CPS-PDUs (Exemplo 1). Neste exemplo, as CPS-SDUs possuem tamanho fixo de 16 bytes e os CPS-Packets, 19. Assim, é possível montar uma CPS-PDU com vários CPS-Packets, uns completos, outros não. Neste caso, não houve espaço para acomodar o terceiro CPS-Packet na primeira CPS-PDU. Apenas o cabeçalho e os cinco primeiros bytes do CPS-Packet foram colocados, e os 11 bytes restantes, remanejados para a CPS-PDU seguinte. Dessa forma, o campo OSF da segunda CPS-PDU deve ser preenchido com o valor decimal 11, indicando que, além do primeiro byte (que é o cabeçalho da CPS-PDU), é preciso saltar outros 11 bytes até encontrar o início de um CPS-Packet nesta CPS-PDU.

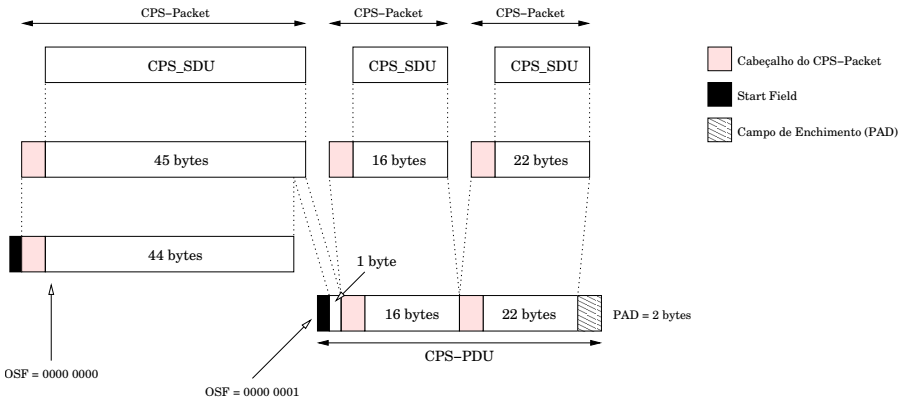


Figura 3.10: Transmissão de CPS-SDUs em CPS-PDUs (Exemplo 2). Neste exemplo os CPS-Packets possuem tamanhos variados até um máximo de 45 bytes. Alguns CPS-Packets inteiros poderão ser maiores do que uma CPS-PDU, como é o caso do primeiro CPS-Packet da figura. O procedimento para acomodar os pacotes em uma CPS-PDU e para o preenchimento do campo OSF são os mesmos descritos acima.

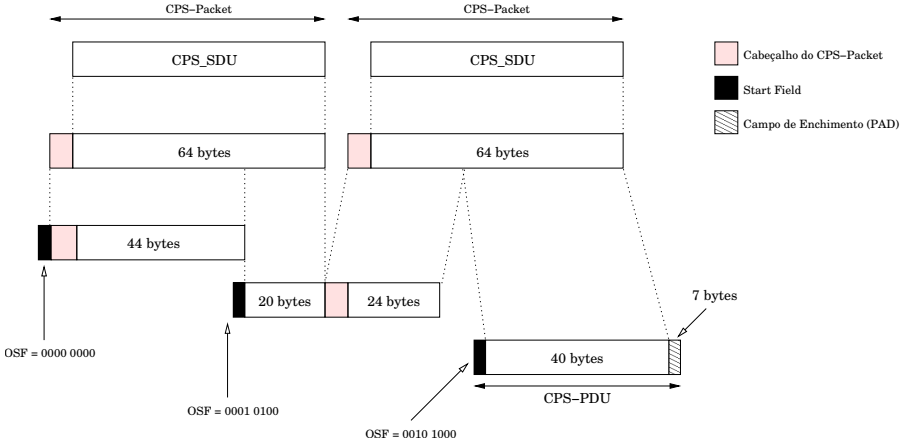


Figura 3.11: Transmissão de CPS-SDUs em CPS-PDUs (Exemplo 3). Aqui todos os CPS-Packets possuem tamanho fixo de 64 bytes. Todos são maiores do que uma CPS-PDU e, portanto, todos deverão ser particionados. O procedimento para acomodar os pacotes em uma CPS-PDU permanece o mesmo - o que não pode ser acomodado em uma CPS-PDU é remanejado para a CPS-PDU seguinte. O preenchimento do campo OSF também se dá de maneira igual à apresentada anteriormente.

3.6 Camada de Adaptação ATM Tipo 5 *(AAL 5)*

A camada de adaptação ATM Tipo 5 (AAL 5) foi definida pelo ITU-T na Recomendação I.363.5 de agosto de 1996 [UNI96]. A AAL 5 disponibiliza serviços das Classes C e D, assim como a AAL 3/4, mas de forma mais simples, com um conjunto de funções reduzido. Por essa razão, a AAL 5 também é conhecida como SEAL - *Simple and Efficient Adaptation Layer* (Camada de Adaptação Simples e Eficiente).

A AAL 5 foi projetada para transportar quadros de tamanho variável, até um limite de

65.535 bytes, em um serviço sem conexão, com capacidade de detecção de erros. Também não há relação de tempo entre a origem e o destino, nem restrição do tipo de tráfego que pode ser transmitido.

3.6.1 Estrutura da AAL 5

A Figura 3.12 mostra a estrutura de subcamadas da AAL 5. A comunicação entre a AAL 5 e a camada superior, bem como da AAL 5 com a camada ATM, ocorre através de um *SAP - Service Access Point*, que é o ponto de interface entre as camadas.

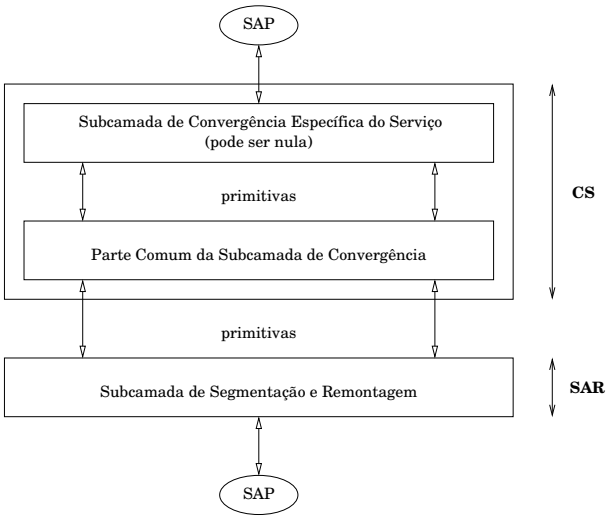


Figura 3.12: Divisão em subcamadas da AAL 5.

A Subcamada de Convergência é dividida em outros dois subníveis:

- CPCS - *Commom Part Convergence Sublayer* (Parte Comum da Subcamada de Convergência)
- SSCS - *Service Specific Convergence Sublayer* (Subcamada de Convergência Específica do Serviço)

Não é obrigatória a existência da SSCS. Segundo [UNI96], a SAR e a CPCS juntas são conhecidas como *Commom Part of the AAL Type 5*.

O esquema de operação da AAL 5 no lado do transmissor é o seguinte [DUT95].

1. Uma AAL-SDU é recebida através do SAP.
2. Ela pode ou não ser processada pela SSCS.
3. O dado é passado à CPCS e torna-se uma CPCS-SDU. A CPCS adiciona um *trailer* de 8 bytes e um outro campo de preenchimento para que o tamanho total da CPCS-PDU seja múltiplo de 48 bytes.
4. A CPCS-PDU é então passada à SAR e torna-se uma SAR-SDU.
5. A SAR segmenta essas informações em diversas SAR-PDUs de 48 bytes, sem que haja inserção de cabeçalho ou *trailer*.

6. A SAR repassa essas SAR-PDUs à camada ATM e indica a última PDU do grupo através de um parâmetro na invocação da camada ATM.

7. A camada ATM coloca um cabeçalho de 5 bytes e inclui na última SAR-PDU um *flag*, chamado de *flag EOM - End of Message*, que indica final de mensagem.

No receptor, ocorre o processo inverso. A Figura 3.13, baseada em [DUT95], ilustra esse processo.

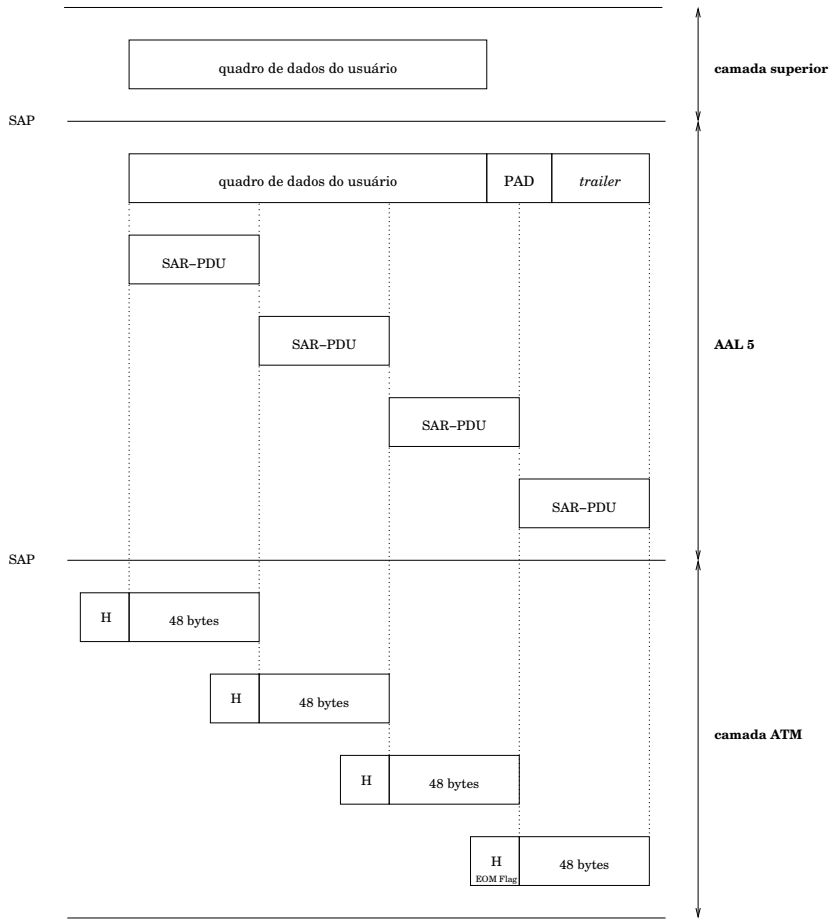


Figura 3.13: Esquema de operação da AAL 5.

Baseado na divisão da AAL 5 em subcamadas, [UNI96] define os *modelos funcionais da AAL 5*, para o lado transmissor e para o lado receptor, mostrados na Figura 3.14. Nesses modelos são apresentadas as primitivas de comunicação entre as subcamadas, bem como as primitivas utilizadas para a comunicação com a camada ATM através do SAP.

3.6.2 Subcamada de Segmentação e Remontagem da AAL 5

A SAR recebe da CPCS uma SAR-SDU de tamanho variável múltiplo de 48, e gera SAR-PDUs com segmentos da SAR-SDU de 48 bytes, sem adicionar nenhum byte de cabeçalho ou *trailer*. A Figura 3.15 mostra esse procedimento.

As funções desempenhadas pela SAR são as seguintes [UNI96]:

- Preservação da SAR-SDU: Isso é feito através de uma indicação de final da SAR-SDU;

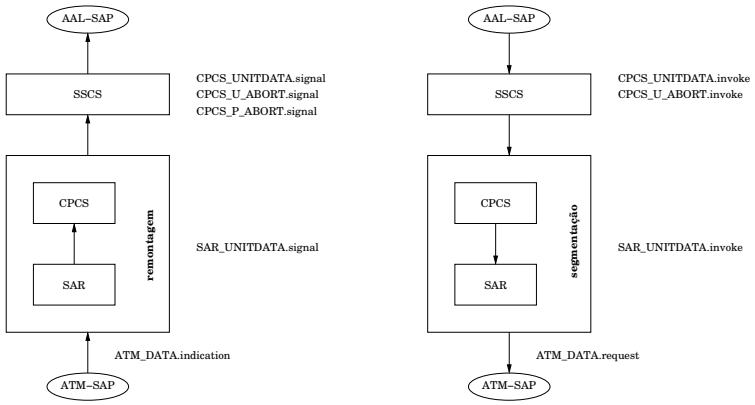


Figura 3.14: Modelo funcional da AAL 5 para (a) o receptor e (b) o transmissor.

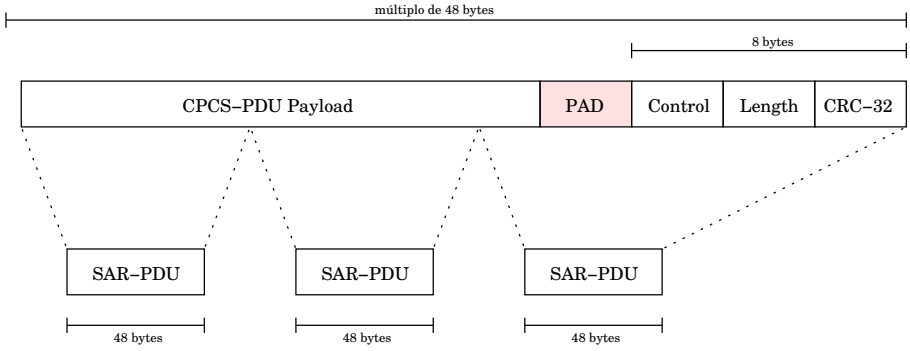


Figura 3.15: Segmentação de uma SDU em PDUs na AAL 5.

- Manipulação da informação de congestionamento: A SAR simplesmente repassa as informações de congestionamento recebidas para a camada seguinte (observado o sentido da comunicação);
- Manipulação da informação de prioridade de perda: A SAR repassa as informações de prioridade de perda de células para a camada seguinte (observado o sentido da comunicação).

3.6.2.1 Primitivas da Subcamada de Segmentação e Remontagem da AAL 5

Estas primitivas são utilizadas na comunicação da SAR com a CPCS, onde não existe um SAP para essa finalidade. As primitivas são chamadas de *invoke* e *signal* para acentuar a ausência do SAP.

Para transferência de dados, o ITU-T define na Recomendação I.363.5 [UNI96] duas primitivas, que são: *SAR-UNITDATA invoke* e *SAR-UNITDATA signal*. Elas utilizam os seguintes parâmetros:

Interface Data (ID): Esse parâmetro especifica a unidade de dados transferida entre a SAR e a CPCS. Seu tamanho é um múltiplo de 48 bytes e ele não representa, necessariamente, uma SAR-SDU completa.

More (M): Indica quando a *Interface Data* contém a última parte da SAR-SDU.

SAR-Loss Priority (SAR-LP): Indica a prioridade de perda associada à ID. Pode conter apenas dois valores, prioridade alta (1) e prioridade baixa (0).

SAR-Congestion Indication (SAR-CI): Esse parâmetro indica se a ID passou por algum ponto da rede congestionado.

3.6.3 Subcamada de Convergência da AAL 5

Como mostrado anteriormente, a Subcamada de Convergência divide-se em SSCS e CPCS. Segundo o ITU-T, podem ser definidos diferentes protocolos para a SSCS, que dão suporte diferentes tipos de serviços, ou ainda grupos de serviços. Por outro lado, a SSCS também pode ser nula. Todas essas definições são tratadas em Recomendações separadas, e não serão objetivo do presente estudo. Assim, a subcamada com maior ênfase nesta seção será a CPCS.

A CPCS recebe da camada superior uma CPCS-SDU e acrescenta a ela um *trailer* de 8 bytes, formando então uma CPCS-PDU. Para a transmissão dessas informações, dois modos de serviços são definidos em [UNI96]:

Message Mode: Onde uma CPCS-SDU é transmitida em exatamente uma CPCS-IDU (*Interface Data Unit*). Esse serviço possibilita a transferência de uma única CPCS-SDU em uma CPCS-PDU.

Streaming Mode: Onde uma CPCS-SDU é transmitida em uma ou mais CPCS-IDUs. Essas IDUs podem ser transmitidas em tempos diferentes. Esse serviço possibilita a transferência de todas as CPCS-IDUs que pertencem a uma mesma CPCS-SDU em uma única CPCS-PDU. O que ocorre é que, à medida que as IDUs chegam elas são repassadas à SAR, com um bit que indica o final da CPCS-PDU.

A estrutura de uma CPCS-PDU é mostrada na Figura 3.16. Os campos desta estrutura são comentados a seguir.

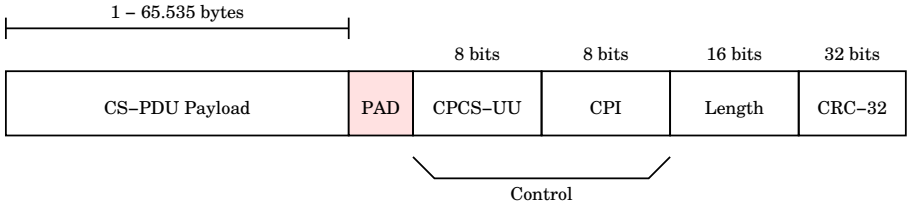


Figura 3.16: Estrutura da CPCS-PDU para a AAL 5.

CPCS-PDU Payload: É o campo de carga útil que transporta uma CPCS-SDU. Pode ter um tamanho que varia de 1 a 65.535 bytes.

PAD - Padding: Esse campo é utilizado apenas para tornar o tamanho total da CPCS-PDU múltiplo de 48 bytes. Pode ter uma tamanho variando de 0 a 47 bytes.

CPCS-UU - CPCS User-to-user Indication: Esse campo tem um tamanho de um byte e é utilizado para a transferência transparente das informações entre usuários da CPCS.

CPI - Commom Part Indicator: É um campo de um byte que é utilizado, entre outras funções, para totalizar 8 bytes de tamanho para o *trailer*.

Length: Esse campo, com dois bytes de tamanho, é utilizado para indicar a quantidade de bytes válidos que está sendo transmitida no campo *CPCS-PDU Payload*. Ele contém uma codificação em binário para o tamanho em bytes.

CRC-32: É um campo de 4 bytes utilizado para detecção de erros. O valor do CRC é calculado sobre toda a CPCS-PDU (incluindo o *trailer*, PAD e *CPCS-PDU Payload*). O CRC é gerado com base no polinômio: $G(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$.

As funções desempenhadas pela CPCS dependem do modo de serviço que se está utilizando. Elas incluem [UNI96]:

- Preservação da CPCS-SDU: Essa função é disponibilizada para delineação e transparência das CPCS-SDUs.
- Preservação da informação transmitida entre usuários da CPCS: Os usuários da CPCS seriam as camadas superiores que utilizam a CPCS da AAL 5 para transmissão de seus dados. Essa função é disponibilizada para transmitir de forma transparente essas informações.
- Manipulação e detecção de erros: Disponibiliza a detecção e manipulação de CPCS-PDUs corrompidas. As CPCS-SDUs corrompidas são descartadas ou entregues à SSCS. Quando elas são entregues ao usuário, uma indicação de erro é associada à entrega.
- Cancelamento (*Abort*): Permite o cancelamento da transmissão parcial de uma CPCS-SDU. Essa função é indicada no campo *Length* do *trailer*, que conterá "zeros" neste caso.
- Preenchimento (*Padding*): Essa função permite tornar o tamanho da CPCS-PDU múltiplo de 48 bytes, para que a SAR possa proceder a divisão em SAR-PDUs completas.
- Manipulação da informação de congestionamento: A CPCS repassa as informações de congestionamento recebidas para a camada seguinte (observado o sentido da comunicação).
- Manipulação da informação de prioridade de perda: A CPCS simplesmente repassa as informações de prioridade de perda de células para a camada seguinte (observado o sentido da comunicação).

3.6.3.1 Primitivas da CPCS da AAL 5

Estas primitivas são utilizadas para a comunicação entre as subcamadas da AAL 5. Estão definidas primitivas para a transferência de dados e para a função de cancelamento. Assim como na SAR, as primitivas são chamadas de *invoke* e *signal* para acentuar a ausência do SAP entre as subcamadas da AAL 5.

As primitivas para transferência de dados são: *CPCS-UNITDATA invoke* e *CPCS-UNITDATA signal*. Para elas, são definidos os seguintes parâmetros:

Interface Data (ID): Esse parâmetro especifica a unidade de dados que será transferida entre a CPCS e a SSCS. No *Message Mode*, essa unidade representa uma CPCS-SDU completa, o que não é necessariamente verdade no *Streaming Mode*.

More (M): Não é utilizado no *Message Mode*. No *Streaming Mode*, indica se a *Interface Data* contém o final de uma CPCS-SDU ou não.

CPCS-Loss Priority (CPCS-LP): Indica a prioridade de perda associada a uma CPCS-SDU. Só assume dois valores (alta e baixa prioridade). No *Streaming Mode*, é obrigatório na primeira primitiva *invoke* relacionada a uma CPCS-SDU, e não está presente nos demais casos. No receptor, só está presente na última primitiva *signal* relacionada a uma CPCS-SDU.

CPCS-Congestion Indication (CPCS-CI): Esse parâmetro indica se a CPCS-SDU associada a ela passou por algum congestionamento. Assim como o parâmetro anterior, no *Streaming Mode*, é obrigatório na primeira primitiva *invoke* relacionada a uma CPCS-SDU, e não está presente nos demais casos; no receptor, só está presente na última primitiva *signal* relacionada a uma CPCS-SDU.

CPCS User-to-user Indication (CPCS-UU): É um parâmetro transportado de forma transparente pela CPCS. No *Streaming Mode*, é obrigatório na última primitiva *invoke* relacionada a uma CPCS-SDU, e não está presente nos demais casos. No receptor, só está presente na última primitiva *signal* relacionada a uma CPCS-SDU.

Reception Status (RS): Indica que a CPCS-SDU associada a ele pode estar corrompida. No *Streaming Mode*, só está presente na última primitiva *signal* relacionada a uma CPCS-SDU.

Capítulo 4

Contexto da Aplicação

As implementações das camadas de adaptação ATM fazem parte de um projeto de cooperação com uma empresa. Esse projeto envolve redes ATM para transmissão e recepção de dados através da AAL 5, bem como transmissão e recepção de voz compactada através da AAL 2.

Está em desenvolvimento um módulo, denominado *driver* ATM, que implementa as funções da camada ATM e das AALs 2 e 5, com suporte para os tráfegos UBR (*unspecified bit rate*), VBR-nrt e VBR-rt (*variable bit rate non real-time* e *variable bit rate real-time*). A Figura 4.1 mostra um diagrama de blocos da estrutura do *driver* ATM, bem como suas interfaces com o mundo externo. Uma possível aplicação para este *driver* é utilizá-lo como parte de um modem xDSL (*Digital Subscriber Line*) para conexões de alta velocidade à Internet e à rede telefônica via par trançado.

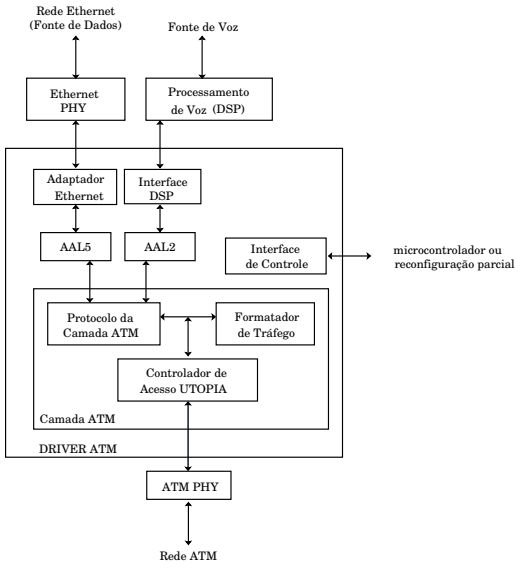


Figura 4.1: Diagrama de blocos de um *driver* ATM, mostrando as interfaces com as fontes de dados e voz e com a rede ATM.

Os módulos das camadas de adaptação ATM, AAL 5 e AAL 2, recebem os sinais de dados e voz (AAL_SDUs - *service data units*) e geram AAL_PDUs (*protocol data units*), que são transmitidas para a camada ATM. As AAL_PDUs são então transformadas em células ATM e multiplexadas. Se, ou quando atenderem aos critérios de conformidade de tráfego estabele-

cidos pelo módulo Formatador de Tráfego, as células são repassadas ao Controlador de Acesso UTOPIA. A Interface UTOPIA, padronizada pelo ATM Fórum, garante a portabilidade do *driver* ATM, podendo este operar em diversos meios físicos, tais como cobre ou fibra ótica, bastando para isto utilizar diferentes implementações do módulo externo ATM-PHY. A implementação dessa interface utiliza o padrão UTOPIA nível 2, apresentado em [COM95]. O módulo ATM PHY, externo ao *driver*, é utilizado para transformar o sinal digital recebido da interface UTOPIA em um sinal adequado ao meio físico. A Interface de Controle é proposta para permitir a configuração dos módulos internos do *driver*. A idéia inicial era realizar essa configuração através de um microcontrolador. Atualmente, considera-se a possibilidade de utilizar reconfiguração parcial, uma vez que a implementação do mesmo visa a utilização de FPGAs com esta capacidade.

No caso da AAL 5, a camada superior responsável pelos dados é um adaptador Ethernet. Esse adaptador transfere dados entre o *driver* ATM e uma rede Ethernet. A implementação desse adaptador Ethernet será uma simplificação do projeto cuja implementação é descrita em [TOR01], onde os pacotes de dados Ethernet possuem tamanho que varia entre 64 e 1518 bytes. Por outro lado, a AAL 2 tem como camada superior a Interface DSP. Essa interface é específica para o tipo de DSP a ser utilizado no processamento da voz. A responsabilidade da interface é retirar qualquer tipo de cabeçalho ou *trailer* inserido pelo DSP e entregar à AAL 2 a informação de voz e alguns parâmetros necessários a esta, como o identificador do canal de voz (`CID` - *Channel Identifier*) e o tamanho do pacote. Além disto, no sentido contrário de transferência, esta interface formata os pacotes recebidos pela AAL 2 para compatibilizá-los com o formato esperado pelo DSP.

As seções seguintes tratam em mais detalhe os módulos internos ao *driver* que não são parte desta dissertação, mas possuem importância para o entendimento da mesma. São eles: o adaptador Ethernet, a interface com o DSP e modems xDSL.

4.1 Adaptador Ethernet e Rede Ethernet

O adaptador Ethernet deve ser desenvolvido utilizando como base o trabalho apresentado em [TOR01], que mostra a implementação funcional, visando prototipação, de um MAC Ethernet. Esse MAC é composto por dois módulos de *hardware*, um para transmissão e outro para recepção de pacotes Ethernet. O módulo de transmissão é responsável por montar pacotes Ethernet a partir dos dados que recebe do usuário, através de um *buffer* para transmissão, e repassá-los ao meio físico. No caso deste projeto, o usuário do adaptador Ethernet é a AAL 2. O módulo de recepção, por sua vez, desmonta os pacotes Ethernet recebidos do meio físico para poder repassá-los ao usuário (AAL 2), agora através do *buffer* de recepção. Ambos módulos implementam funções de controle de fluxo e utilizam o método de acesso ao meio conhecido como CSMA/CD (*Carrier Sense Multiple Access with Collision Detection*).

No caso do adaptador Ethernet é feita uma simplificação no projeto do MAC. As funções de controle de fluxo e acesso ao meio devem ser mantidas como no projeto original, assim como os *buffers*. A principal modificação é que no adaptador não há a necessidade de montar e desmontar pacotes Ethernet. A função do adaptador Ethernet é realizar a comunicação entre o *driver* ATM e uma rede Ethernet, como mostra a Figura 4.2. A rede de origem fornece ao adaptador Ethernet um pacote completo, e o adaptador apenas repassa este ao *driver* ATM. Esse pacote é então encapsulado em células ATM e transmitido através de uma rede ATM. Na outra extremidade da comunicação há outro adaptador, responsável por receber este pacote completo do *driver* e entregá-lo à rede Ethernet de destino.

O adaptador Ethernet deve manter uma lista dos endereços que são válidos para sua rede local. Dessa forma, ele saberá se o pacote que está transitando pela rede é direcionado a um outro nodo da mesma rede ou a um nodo em uma rede externa. Sempre que o endereço destino do pacote Ethernet não constar na lista de endereços do adaptador, significa que o destino é, na verdade, um nodo que não pertence à rede local e, por isso, o adaptador deverá encaminhar o pacote ao *driver* ATM para que seja enviado à rede destino. Problemas de endereçamento entre as redes Ethernet fonte e destino estão fora do escopo deste trabalho, sendo delegados às camadas superiores do protocolo de comunicação.

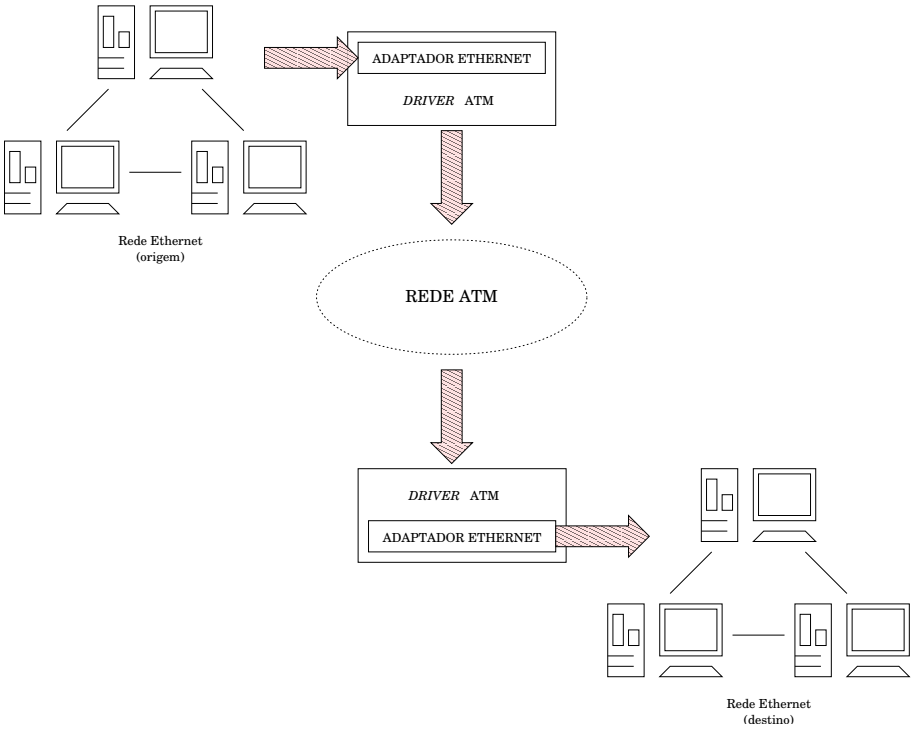


Figura 4.2: Fluxo de transmissão de pacotes Ethernet através do *driver* ATM.

4.2 Interface DSP e Processamento da Voz

Toda a transferência de pacotes de voz realizada pela AAL 2 considera que a informação está não só codificada, mas também compactada. Para codificação, é possível utilizar o padrão PCM [UNI93], onde são usados valores de 0 a 255 para representação da informação, requerendo um taxa de 64kbps. Por outro lado, utilizando compactação, a largura de banda requerida para a transferência é menor. Com o padrão ADPCM [UNI90], que é uma variação do PCM, obtém-se taxas de 40, 32, 24 ou 16 kbps. A Tabela 4.1 mostra uma comparação entre os tamanhos da informação e as taxas obtidas para a codificação PCM e os quatro tipos de compactação ADPCM.

A utilização de um DSP é necessária para que a voz seja tratada antes de ser entregue à AAL 2. A fonte de voz gera sinais analógicos que deverão ser codificados e compactados para então serem transmitidos através da Rede ATM. Como a AAL 2 deve ser independente do DSP utilizado, é preciso projetar uma interface que adapte o DSP à AAL 2. Apenas os sinais entre a Interface DSP (como o módulo foi chamado) e a AAL 2 foram definidos, para que o projeto

Tabela 4.1: Comparação entre a codificação PCM e a codificação ADPCM.

	Tamanho da informação por amostra	Taxa de transferência
PCM	8 bytes	64 kbps
ADPCM	5 bytes	40 kbps
	4 bytes	32 kbps
	3 bytes	24 kbps
	2 bytes	16 kbps

da AAL 2 pudesse ser realizado. A validação dos módulos de *hardware* da AAL 2 considera um pacote de voz compactada com quatro bytes de informação, ou seja, compactação ADPCM a 32 kbps.

4.3 Modems xDSL

As tecnologias xDSL, de um modo geral, trabalham dividindo a linha telefônica existente em duas faixas de freqüências. As baixas freqüências são reservadas para chamadas telefônicas convencionais (voz), enquanto que as altas freqüências são utilizadas para Internet (dados), fazendo com que seja possível utilizar a mesma linha telefônica tanto para chamadas comuns como para acesso à Internet, simultaneamente.

Existem vários tipos de modems DSL, entre eles os SDSL, os HDSL e os ADSL. Os modems SDSL (*Symmetric Digital Subscriber Line*) são chamados de simétricos porque suportam as mesmas taxas, tanto para *download* como para *upload*.

A tecnologia HDSL (*High-bit-rate Digital Subscriber Line*) é uma substituição transparente por uma linha repetidora T-1 na planta de distribuição. Isso permite que sinais DS-1 possam ser transportados por distâncias de até 2000 pés em cabos de cobre sem repetidores de linha. Modems HDSL utilizam dois pares de cabos de cobre para transmissão *full-duplex*, usando canceladores de eco, cada par transportando dados a 784 kbps. A baixa taxa de bits permite uma baixa faixa de freqüências de operação, o que reduz as perdas no canal.

ADSL (*Asymmetric Digital Subscriber Line*) é um serviço de Internet de banda larga de alta velocidade, mais adequado ao uso residencial. É chamado de assimétrico (*asymmetric*) porque é reservada uma maior largura de banda para a recepção de dados (*download*) do que para a transmissão (*upload*), o que faz dos modems ADSL uma solução ideal para usuários domésticos.

Um modem ADSL alcança taxas de até 9Mbps para *download* e até 1Mbps para *upload*. Essas taxas ampliam a capacidade de acesso em 50 vezes ou mais, sem a utilização de novas linhas, com relação à capacidade existente. Segundo [TUT02], um circuito ADSL conecta um modem ADSL em cada ponta de uma linha telefônica de par-trançado comum e cria três canais lógicos de alta velocidade para *download*, um canal *duplex* de média velocidade (dependendo da companhia telefônica) e uma linha de voz comum. Essa linha de voz é dividida fora do modem digital pelos filtros, garantindo assim um canal initerrupto, mesmo se houver falha do ADSL. Os canais de alta velocidade podem operar na faixa entre 1.5 e 6.1 Mbps, enquanto que o canal *duplex* opera com taxas entre 16 e 832 kbps. Cada canal ainda pode ser submultiplexado para formar canais de baixa velocidade, dependendo do sistema.

Outras informações sobre modems xDSL podem ser obtidas em sites como [ADS01],

[AYA02], [MAR02], [2Wi02], [CON02] e [UNI02].

4.4 Estimativa de Desempenho para Interfaces Internas e Externas

Mesmo tendo apenas com uma visão geral do contexto da aplicação, é possível realizar uma estimativa simplificada de desempenho dos módulos a serem implementados. A Figura 4.3 mostra o fluxo de transferência de dados e voz, ressaltando os requisitos de largura de banda máxima para transmissão e recepção.

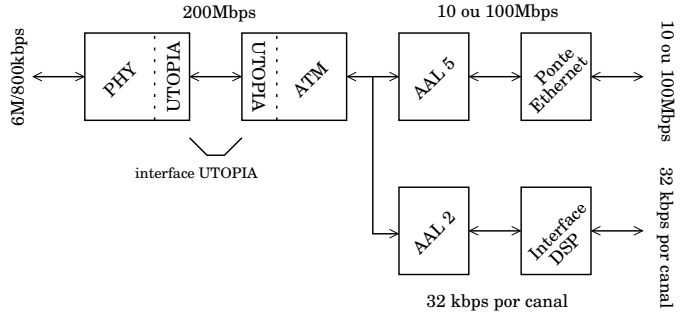


Figura 4.3: Especificação das taxas de transmissão e recepção máximas do projeto do *driver* ATM. O módulo Ethernet deve permitir taxas de 10Mbps ou 100Mbps, mas apenas a primeira está sendo utilizada no presente projeto. O módulo PHY opera a uma taxa de 6Mbps para recepção e 800kbps para envio de e para a rede ATM, respectivamente. Os caminhos sem largura de banda associada não possuem limite especificado, e possuem largura de banda tipicamente superior a todas as especificações.

A taxa de transferência, utilizada no estudo de caso entre o DSP e a Interface é de 32 kbps. Trata-se de uma das taxas obtidas com a codificação ADPCM [UNI90]. Assim, pode-se inferir que a largura de banda máxima entre a Interface DSP e a AAL 2 também será nesta taxa.

No caso de transferência de dados, a ponte Ethernet é padronizada para operar a taxas de 10Mbps ou 100Mbps. Neste projeto, utiliza-se apenas a taxa de 10Mbps. Outra consideração é quanto a taxa de operação do módulo UTOPIA. A implementação realizada utiliza a especificação nível 2 [COM95], com um sinal de *clock* de 25MHz, o que proporciona uma taxa de transferência máxima de 200Mbps. Ainda segundo [COM95], o meio físico opera neste caso a uma taxa máxima padronizada de 155,52Mbps.

Com relação ao módulo PHY, que é responsável pela interface com o meio físico, estima-se taxas de 6Mbps para recepção do meio físico e 800kbps para transmissão. Com esta avaliação premilinar, nota-se que, com a camada ATM operando a 200 Mbps, é possível multiplexar várias AALs sobre uma mesma camada ATM. Segundo a Tabela 3.3, é possível multiplexar até 248 canais de voz em uma AAL 2, uma vez que são 248 os valores possíveis para o campo `CID`. Por outro lado, é preciso observar o limite da largura de banda alcançado nesta estimativa. Se 248 canais de voz forem utilizados, a uma taxa de 32kbps, seria necessária uma largura de banda de 7,75Mbps, que excede inclusive a taxa de recepção do meio físico.

Capítulo 5

Implementação da AAL 5

A implementação da AAL 5 foi desenvolvida em VHDL, visando a prototipação em uma plataforma de *hardware* comercial. Essa implementação foi validada em uma etapa de verificação funcional, descrita na Seção 5.1.5, com a ferramenta Active HDL, da Aldec.

De acordo com a Figura 3.14 da Seção 3.6.1, o ITU-T subdivide a AAL 5 em CS (subdividida ainda em SSCS e CPCS) e SAR, onde cada uma dessas subcamadas têm suas funções definidas. Para fins de implementação, pode ser feita uma estruturação em módulos como mostra a Figura 5.1. Cada módulo tem sua função, ou seja, realiza um determinado processamento sobre as células que recebe e coloca o resultado deste processamento em *buffers* intermediários para que o módulo seguinte o receba e também realize o devido processamento. Isso ocorre sucessivamente até que uma célula passe por todas as etapas de processamento que cabem à AAL 5.

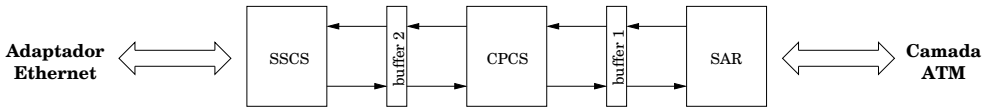


Figura 5.1: Estrutura em módulos que pode ser inferida a partir de [UNI96].

No caso de projetos específicos, é possível utilizar os recursos disponíveis para otimizar a implementação da AAL 5. No contexto abordado, onde tem-se um adaptador Ethernet, os recursos disponíveis ao projeto da AAL 5 são os *buffers de transmissão*. Esses *buffers* são acessados pelo adaptador Ethernet somente quando estiverem preenchidos com um pacote Ethernet completo, no caso da transmissão, ou quando estiverem vazios, no caso da recepção. Como mencionado anteriormente na Seção 4.1, o adaptador Ethernet espera uma sinalização para poder utilizar os *buffers*, indicando que o *buffer* de transmissão contém um pacote completo, ou que o *buffer* de recepção está pronto para receber um novo pacote. Apenas após essa sinalização, o adaptador irá considerar os dados do *buffer* de transmissão prontos para serem enviados ao meio físico, ou o *buffer* de recepção pronto para receber os dados que serão enviados à AAL 5. Por esse motivo a SAR e a CPCS podem realizar suas funções diretamente sobre os *buffers* Ethernet. Dessa forma, pode-se visualizar uma estrutura de implementação como o mostrado na Figura 5.2.

Essa implementação tem um inconveniente. No caso da recepção de dados, a CPCS precisa acessar os dados contidos no *buffer* de transmissão para realizar o cálculo do CRC. Ao mesmo tempo, a SAR está escrevendo no *buffer* os dados que ela recebe da camada ATM, agrupando-os. Nesse ponto, a CPCS estaria impossibilitada de operar enquanto a SAR não liberasse o

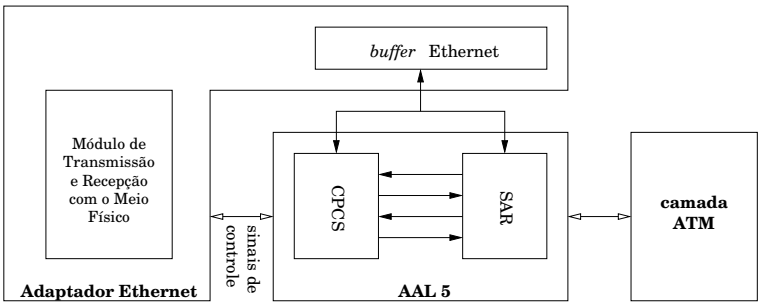


Figura 5.2: Subcamadas SAR e CPCS compartilhando os *buffers* do MAC.

buffer do adaptador Ethernet, o que causaria uma perda de eficiência. A solução para esse problema é simples. Além da SAR enviar os dados recebidos para o *buffer*, também envia-os à CPCS, através de um barramento na interface SAR/CPCS. Assim, enquanto a SAR preenche o *buffer*, a CPCS calcula o CRC. Isso melhora a eficiência da implementação. A Figura 5.3 mostra um diagrama de tempos para o processamento completo da recepção de dados pela AAL 5.

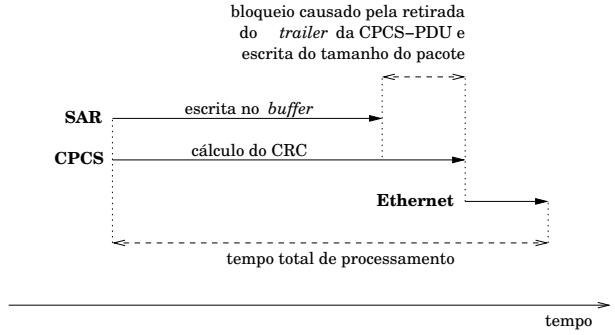


Figura 5.3: Diagrama de tempos para o processamento da AAL 5.

No caso da transmissão, ocorre uma situação semelhante. A CPCS deve calcular o CRC para enviá-lo à camada ATM, enquanto a SAR segmenta os dados recebidos a partir do *buffer* de recepção Ethernet para também enviá-los à camada ATM. Como o valor do CRC é enviado em um *trailer*, após todos os bytes de informação já terem sido transmitidos, a SAR e a CPCS também podem operar, em parte, em paralelo, melhorando também a eficiência na transmissão de dados.

A implementação da AAL 5 pode ser estruturada como mostra a Figura 5.4. Foram definidos dois grandes módulos, um para as funções de recepção e outro para as funções de transmissão. As seções seguintes apresentam detalhes da implementação desses módulos e as relações dos mesmos com a Recomendação I.363.5 do ITU-T [UNI96].

5.1 Módulo de Recepção da AAL 5

O diagrama de blocos de mais alto nível da recepção é mostrado na Figura 5.5.

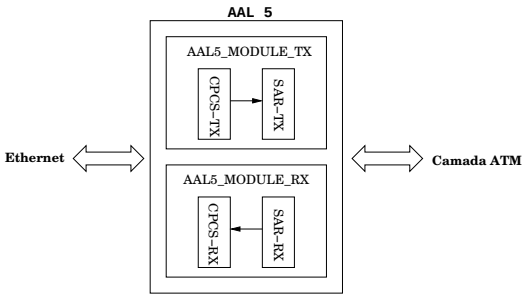


Figura 5.4: Estrutura em módulos implementada.

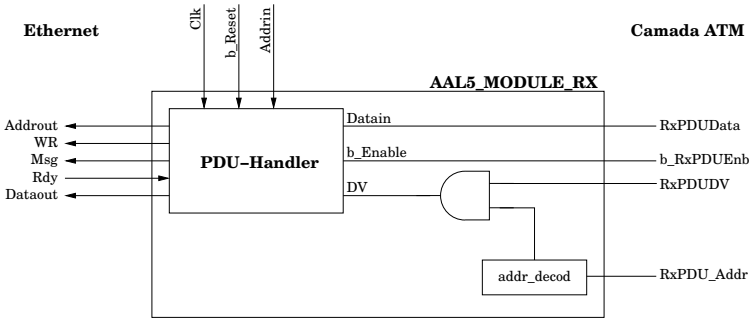


Figura 5.5: Diagrama de blocos do módulo de recepção da AAL 5.

5.1.1 Interface AAL 5/ATM

A interface com a camada ATM consiste dos seguintes sinais:

RxPDUData: É por estas linhas que os dados são enviados da camada ATM para a AAL 5. Nesta implementação o sinal foi projetado com largura de 8 bits. Futuramente, a largura será parametrizável.

RxPDUAddr: Este sinal com largura de 11 bits é utilizado para endereçar a camada de adaptação. Uma mesma camada ATM pode gerenciar mais de uma AAL, do mesmo tipo ou não. Por esse motivo foi definido um barramento de endereços, chamado `RxPDUAddr`, que, através de um decodificador de endereços, determina se os dados estão sendo enviados para a AAL em questão.

RxPDUDV: Este sinal de 1 bit é utilizado pela camada ATM para indicar que o dado presente em `RxPDUData` é válido (`RxPDUDV = 1`) ou não (`RxPDUDV = 0`). Em conjunto com a saída do decodificador de endereços, indica à AAL 5 se ela deve ou não processar o dado que está em `RxPDUData`. Isto significa que, se o dado é válido e o decodificador de endereços reconhece o endereço em `RxPDUAddr`, então o dado em `RxPDUData` deve ser processador pela AAL.

b_RxPDUEnb: Este sinal de 1 bit é enviado da AAL 5 para a camada ATM indicando se a mesma pode receber novos dados ou não. Quando `b_RxPDUEnb = 0`, a AAL 5 pode receber dados.

5.1.2 Interface AAL 5/Adaptador Ethernet

A interface com o adaptador Ethernet consiste dos seguintes sinais:

Addrout: Este barramento de 11 bits informa em qual endereço do *buffer* de transmissão do MAC o dado deve ser gravado.

Dataout: É por este barramento de 8 bits que os dados são enviados ao *buffer* de transmissão Ethernet.

WR: Este sinal de 1 bit indica se o dado presente em `Dataout` deve ser gravado no *buffer* de transmissão ou não. Se `WR = 1`, a escrita será feita.

Msg/Rdy: Esses dois sinais são utilizados conjuntamente em um *handshake* que avisa ao adaptador Ethernet que os dados estão todos no *buffer* de transmissão. Depois que a AAL 5 termina de preencher o *buffer*, ela ativa o sinal `Msg` (`Msg = 1`) e aguarda até que o adaptador perceba e ative `Rdy` (`Rdy = 0`). Após perceber que `Rdy = 0`, a AAL 5 desativa o sinal `Msg` e aguarda até que o adaptador Ethernet desative `Rdy`, indicando que o *buffer* de transmissão está livre para receber um novo pacote. Este *handshake* é implementado por uma máquina de estados, detalhada na Seção 5.1.4.2.

5.1.3 Módulo Decodificador de Endereços - `addr_decod`

O módulo do decodificador de endereços é chamado de *addr_ decod*, como mostrado na Figura 5.5. O endereço interno do decodificador, neste projeto, foi implementado de maneira *hard-wired*, ou seja, é configurável apenas por síntese. Futuramente, este protejo poderá ser modificado para que este endereço esteja em um registrador e para que seja possível configurá-lo através de um microcontrolador, ou via reconfiguração parcial.

5.1.4 Módulo Manipulador de PDUs - `PDU_Handler`

O módulo manipulador de PDUs, chamado `PDU_Handler`, pode ser considerado o mais importante dentre os que compõem o módulo de recepção. Isso pelo fato de ser ele o responsável pelas funções características da AAL 5.

Esse módulo funciona da seguinte maneira: a camada ATM envia sempre seqüências de 49 bytes, um byte por ciclo de clock. O primeiro byte dessa seqüência contém as informações de fim de pacote, prioridade de perda de células e indicação de congestionamento. Essas informações são na verdade os parâmetros da primitiva `ATM-DATA`, para a comunicação da AAL 5 com a camada ATM através do SAP. O byte recebido tem o seguinte formato, mostrado na Figura 5.6, onde:

AUU: Neste contexto, é o bit que indica fim de pacote. Quando a camada ATM está enviando a última PDU de uma AAL5_SDU, esse bit contém o valor 1.

LP - *Loss Priority*: Esse bit indica a prioridade de perda da informação a ser transmitida. Assume 0 para prioridade alta e 1 para prioridade baixa.

CI - *Congestion Indication*: Esse bit é utilizado para indicar se a célula passou por algum ponto da rede que estava congestionado.

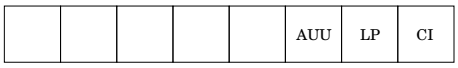


Figura 5.6: Byte com os parâmetros da comunicação entre a AAL 5 e a camada ATM.

Os 48 bytes seguintes compõem a carga útil, ou seja, a PDU propriamente dita. Se o bit `AUU` for 0, todos os 48 bytes são informação. Por outro lado, se `AUU = 1`, os últimos 8 bytes são o *trailer* inserido no lado transmissor, e que deve ser retirado no lado receptor. Os bytes de carga útil recebidos são diretamente enviados ao *buffer* de transmissão Ethernet, incluíndo os bytes de `PAD`.

Como não é possível saber a quantidade de bytes de preenchimento que foram necessários até processar-se o *trailer*, os bytes de `PAD` acabam sendo enviados ao *buffer* do adaptador Ethernet. Por outro lado, como a Ethernet precisa ser informada sobre qual é o tamanho do pacote disponível no seu *buffer*, esse problema é então facilmente resolvido. A informação de tamanho passada à Ethernet é apenas dos bytes que compõem efetivamente o pacote, desconsiderando o `PAD`, o que faz com que apenas os bytes de informação sejam realmente transmitidos ao meio físico.

Os módulos internos do manipulador de PDUs são mostrados na Figura 5.7 e detalhados nas seções seguintes. O módulo `PDU_Handler` possui apenas uma entrada para o sinal de clock (`Clk`) e uma para o sinal de reset (`b_Reset`). Na Figura 5.7 foram representadas várias linhas de clock e reset somente para deixar o diagrama mais claro. Outra consideração é quanto ao sinal `s_Addr_Control`. Ele é uma saída da máquina `Write Length State Machine` e entrada no gerador de endereços `addr_generator`. A linha na Figura foi interrompida apenas para fins de clareza.

5.1.4.1 Máquina de Estados para a Recepção

O processo de recepção de dados pela AAL 5 é basicamente controlado por uma máquina de estados, representada na Figura 5.7 pelo módulo chamado `Receive State Machine`. Ela é apresentada na Figura 5.8. As saídas desta FSM são apresentadas como um vetor de bits dentro de cada estado, onde o bit mais significativo é o primeiro sinal na lista de saídas mostrada na Figura, e assim sucessivamente.

Esta máquina consiste de seis estados distintos, chamados `STOP`, `PARAM`, `DATA`, `TRAILER`, `WRLEN` e `HANDSK`. A máquina fornece onze sinais de saída, indicados na Figura 5.8. O primeiro, `b_Enable`, avisa para a camada ATM se esta pode ou não enviar mais algum dado para a AAL 5. Os dois sinais seguintes, `CRC_Enable` e `CRC_Reset`, controlam o módulo de cálculo do CRC, que será comentado adiante. `Start_HS` e `WR` são para controle da máquina de estados para o *handshake* com o adaptador Ethernet e para controle do *buffer* de transmissão Ethernet, respectivamente. Os seis últimos são sinais de *enable* e *reset* para o gerador de endereços e para os contadores de bytes da célula e do *trailer*.

A funcionalidade associada a cada estado é descrita a seguir:

STOP: É o estado inicial da FSM para recepção, onde nenhuma função é executada. A máquina permanece neste estado enquanto a recepção não tiver início.

PARAM: Neste estado é capturado o byte com os parâmetros `AUU`, `LP` e `CI`, dando início à recepção de mais uma célula. Se o dado for válido, a máquina ficará apenas um ciclo de *clock* neste estado.

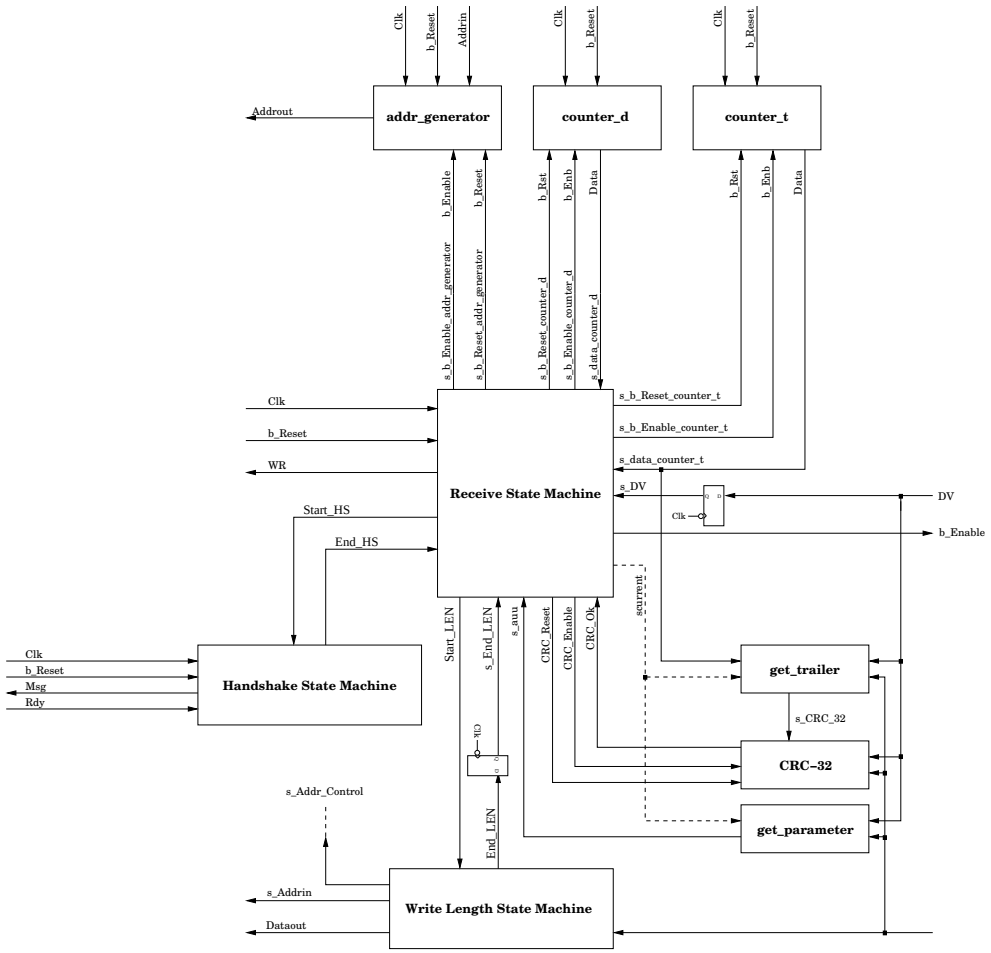


Figura 5.7: Diagrama de blocos dos módulos internos no manipulador de PDUs.

DATA: Aqui ocorre a recepção dos bytes que pertencem à célula. Sempre que um dado for inválido, assinalado por `DV = 0`, a recepção é interrompida, mas a máquina permanece neste estado.

TRAILER: Neste estado ocorre a retirada do *trailer* da célula e a verificação do CRC. Se a célula foi recebida corretamente, ou seja, o valor calculado para o CRC conferiu com o valor armazenado no campo `CRC-32` do *trailer*, então o próximo passo é realizar a escrita do tamanho no *buffer*. Se houve algum problema na recepção e os valores para o CRC não conferem, a célula é descartada e o adaptador Ethernet não deve ser sinalizado.

WRLEN: Neste estado o adaptador Ethernet deverá ser informado sobre o tamanho do pacote que está armazenado no *buffer* de transmissão. Essa informação de tamanho é escrita nos dois primeiros endereços do *buffer*, isto é, nas posições 0 e 1. Essa operação é feita através de uma máquina de estados auxiliar, inicializada pelo sinal `Start_LEN`. A máquina da recepção fica neste estado enquanto a FSM auxiliar não finalizar a escrita do tamanho.

HANDSK: Este é o estado que finaliza a recepção de uma célula. Se tudo ocorreu bem na recepção, o adaptador Ethernet deverá ser avisado para que comece a transmissão dos dados contidos no seu *buffer*. O procedimento de sinalização do adaptador é feito por

Figura 5.8: Máquina de estados para o processo de recepção de dados pela AAL 5 - *Receive State Machine*.

uma outra máquina de estados, inicializada quando a máquina de estados da recepção encontra-se em `HANDSK`. Quando o ciclo da máquina para o *handshake* termina, a máquina da recepção encerra o processo de recepção da célula corrente e inicia a busca de uma nova célula.

5.1.4.2 Máquina de Estados para o *Handshake* com o Adaptador Ethernet

Para que o adaptador Ethernet seja sinalizada ao final da recepção da célula, foi projetada uma segunda máquina de estados, identificada na Figura 5.7 como `Handshake State Machine`. Esta FSM é apresentada na Figura 5.9. Ela fornece valor de saída para dois sinais: `Msg`, para o *handshake* com o adaptador Ethernet e `End_HS`, para indicar à máquina de estados da recepção que o *handshake* terminou.

Esta máquina possui quatro estados distintos, chamados `STOP`, `AALRDY`, `MACTX` e `ENDMAC`, descritos a seguir:

STOP: É o estado inicial da máquina. Ela permanece neste estado por todo o período de recepção da célula. Quando a máquina de recepção determina o início do *handshake*, esta FSM inicia seu ciclo, sinalizando ao adaptador Ethernet que os dados já estão no *buffer*.

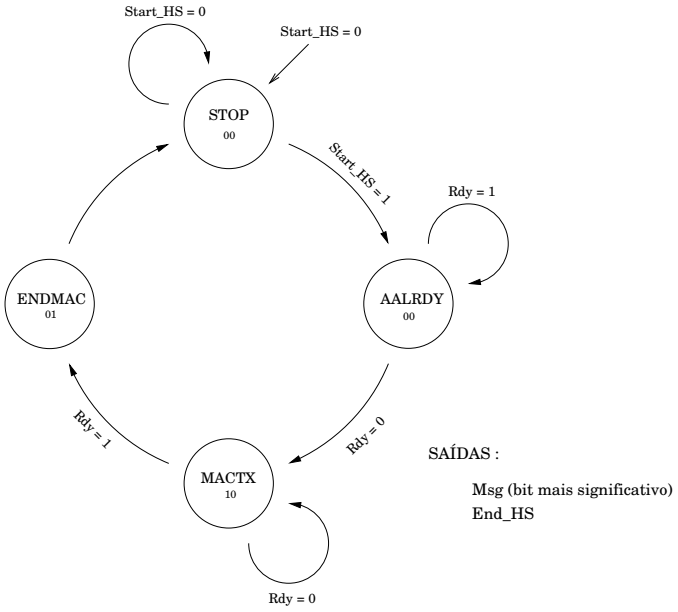


Figura 5.9: Máquina de estados para o *handshake* com o adaptador Ethernet.

AALRDY: Neste estado a AAL 5 avisa o adaptador Ethernet que existe um pacote disponível no *buffer* de transmissão, fazendo `Msg = 1`. A FSM aguarda até que o adaptador perceba o sinal `Msg` e habilite o sinal `Rdy`.

MACTX: Durante a permanência da máquina neste estado, o adaptador Ethernet transmite os dados armazenados em seu *buffer*.

ENDMAC: Este estado finaliza o processo de *handshake*, sinalizando à máquina de recepção para que continue sua operação, recebendo uma nova célula.

A Figura 5.10 mostra os sinais no *handshake* com o adaptador Ethernet. É possível observar a máquina de estados da recepção no estado de `HANDSK` enquanto a FSM responsável por esse *handshake* realiza as trocas de sinais.

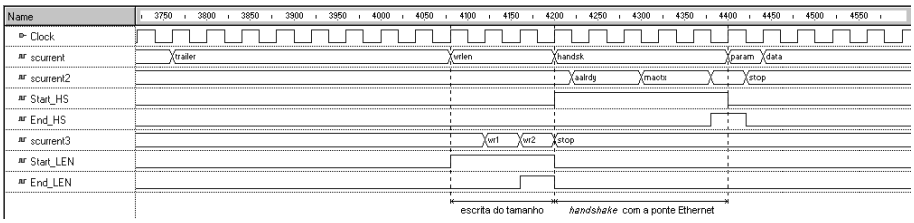


Figura 5.10: Recepção de *trailer* com CRC correto - máquinas de estado. A figura mostra as trocas de estados das três máquinas implementadas: a de recepção propriamente dita (`scurrent`), a de *handshake* com o adaptador Ethernet (`scurrent2`) e a de escrita do tamanho (`scurrent3`). Quando a máquina de recepção encontra-se no estado de `WRLEN`, a máquina de escrita do tamanho executa seu ciclo: escreve o primeiro byte do tamanho quando estiver em `WR1` e o segundo byte quando estiver em `WR2`. Da mesma forma, a máquina para o *handshake* com o adaptador executa seu ciclo quando a máquina de recepção estiver em `HANDSK`.

5.1.4.3 Máquina de Estados para a Escrita do Tamanho do Pacote

O adaptador Ethernet precisa ser informado do tamanho do pacote que encontra-se no *buffer* de transmissão. Isso é feito através das duas últimas posições desse *buffer*, onde são escritos os dois bytes que compõem o campo de tamanho do pacote. Para esta operação é utilizada uma terceira máquina de estados, identificada na Figura 5.7 pelo módulo chamado `Write Length State Machine`. Essa máquina é apresentada em detalhes na Figura 5.11, e fornece saída para quatro sinais: `s_Addrin` e `s_Addr_Control`, para controlar o gerador de endereços Ethernet, `s_Data`, que contém o dado a ser enviado ao *buffer* de transmissão, e `End_LEN`, que avisa a máquina de recepção que o ciclo de execução da terceira máquina terminou.

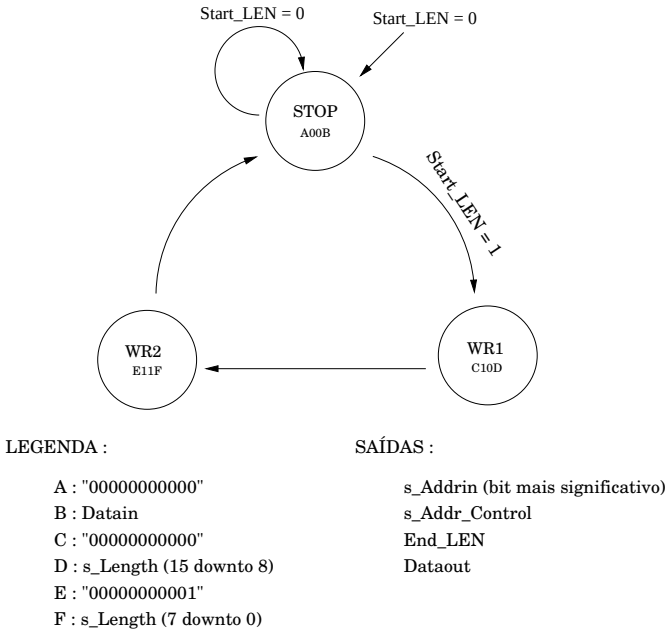


Figura 5.11: Máquina de estados para a escrita do tamanho do pacote.

Os estados que compõem a máquina para escrita do tamanho são os seguintes:

STOP: Este é o estado inicial da máquina de escrita do tamanho. Ela permanece neste estado durante todo o período de execução da máquina de recepção, sem desempenhar nenhuma função, exceto quando esta última determinar o início do processo de escrita do tamanho.

WR1: Neste estado, o primeiro byte que compõe o tamanho do pacote é escrito na posição 0 do *buffer* de transmissão.

WR2: Aqui o segundo byte do tamanho é escrito na posição 1 do *buffer* de transmisão do adaptador Ethernet. Esta máquina então avisa a de recepção que o ciclo de escrita do tamanho do pacote terminou, para que o *handshake* com o adaptador possa ser feito.

Na Figura 5.10, também é possível observar a FSM para recepção no estado de `WRLEN` enquanto a máquina de estados para a escrita do tamanho realiza a gravação do tamanho do pacote Ethernet no *buffer*.

5.1.4.4 Captura dos Parâmetros - `get_parameter`

Este módulo tem um funcionamento simples. Se o estado atual da máquina de recepção é `PARAM` e `DV = 1`, significando que o dado é válido e para esta AAL 5, então o barramento de dados contém um byte com os parâmetros `AUU`, `LP` e `CI`. Esse byte não é armazenado no *buffer* de transmissão do adaptador Ethernet e sim em um sinal auxiliar `s_parameter`. Os valores dos parâmetros também são armazenados individualmente em outros três sinais auxiliares, `s_auu`, `s_cpcs_lp` e `s_cpcs_ci`, respectivamente.

5.1.4.5 Cálculo do CRC - `CRC`

Na implementação da AAL 5 há um módulo destinado ao cálculo do CRC32, identificado na Figura 5.7 como `CRC-32`. Esse módulo calcula o CRC, de maneira serial, para todos os bytes do pacote, incluindo o *PAD* os três primeiros campos do *trailer*: `CPCS-UU`, `CPI` e `Length`. Se o valor calculado for igual ao armazenado no último campo do *trailer* (`CRC-32`), então o módulo gera um sinal `CRC_Ok` em 1 para a máquina de estados da recepção. Caso contrário, `CRC_Ok = 0`. O código VHDL da função que calcula o CRC32 foi obtido em [CRC01].

A Figura 5.12 mostra o início do cálculo do CRC sobre os dados recebidos na primeira célula. É possível observar que o CRC não é calculado sobre o parâmetro, uma vez que este não faz parte do pacote Ethernet, é apenas uma forma da camada ATM comunicar-se com a AAL 5. Na Figura 5.13 é mostrado o final da recepção de uma célula, onde o CRC é finalmente verificado. Neste caso, não houve erro de CRC.

Para todas as validações feitas foram utilizados padrões de teste disponíveis no final da Recomendação I.363.5 do ITU-T, que fornece pacotes de dados com valores corretos para o campo CRC. Os erros no CRC foram simulados alterando um bit no valor correto.

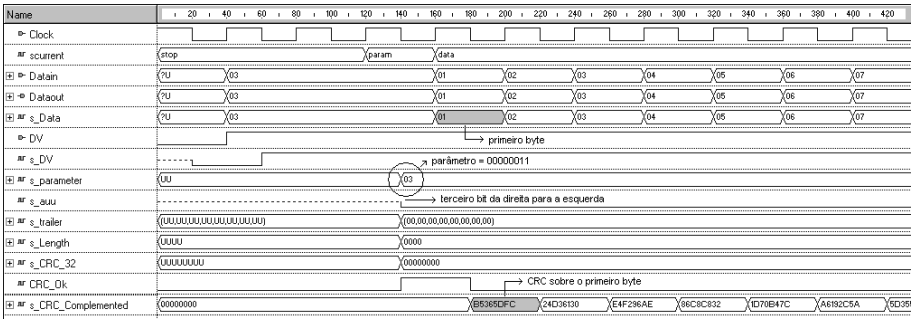


Figura 5.12: Início da recepção da primeira célula - parâmetro e CRC. Na figura pode-se observar o início do cálculo do CRC e o valor do parâmetro recebido. Neste caso, a camada ATM estaria enviando o seguinte: `AUU = 0`, `LP = 1` e `CI = 1`. Como `AUU = 0`, esta não é a última célula do pacote. O primeiro byte da célula (byte 0) tem valor `01` e o CRC calculado sobre esse valor é `B5365DFC`. A cada novo ciclo de *clock*, o valor do CRC é recalculado considerando o valor do dado que chega e o valor do CRC anterior.

5.1.4.6 Retirada do *Trailer* - `get_trailer`

Sempre que a máquina de recepção estiver no estado `TRAILER`, os dados em `Datain` não deverão ser enviados para o *buffer* de transmissão Ethernet. O *trailer* é armazenado em um sinal auxiliar chamado `s_trailer` para que seja possível identificar os campos que o compõe.

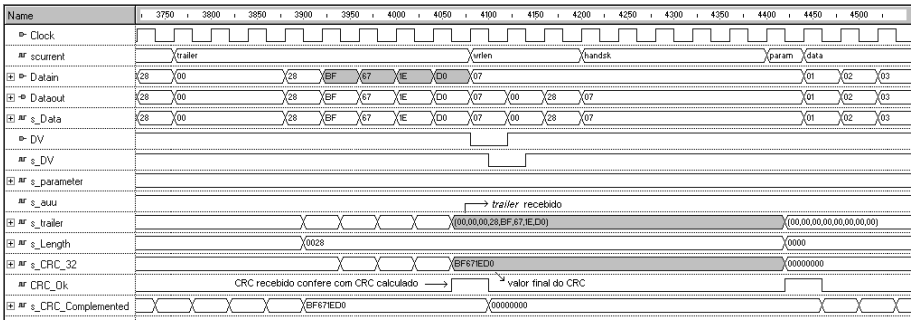


Figura 5.13: Recepção de *trailer* com CRC correto - *trailer* e CRC. A figura mostra os valores finais para os sinais que armazenam o *trailer* e o CRC. O *trailer* tem valor nulo para o primeiro campo (`Control`), `0x0028` para o tamanho (40, em decimal), e `BF671ED0` para o CRC. O cálculo do CRC também resultou o valor `BF671ED0`, armazenado do sinal `s_CRC_Complemented`, o que fez o sinal `CRC_Ok = 1` por pelo menos um ciclo de *clock*.

Os dois bytes mais significativos são do campo `s_Control`, para fins de controle, sem função explicitamente definida em [UNI96]. Os dois bytes seguintes compõem o campo `s_Length` e transportam o tamanho do pacote, em bytes.

Conforme o formato do *trailer*, apresentado na Figura 3.16, os últimos quatro bytes (`s_CRC_32`) contêm o valor do CRC-32 calculado na transmissão. Como mencionado anteriormente, esse valor é calculado também na recepção e o resultado é comparado com o `CRC` armazenado no *trailer*. Se forem iguais, o pacote foi transmitido com sucesso através da rede ATM. Na Figura `??` também pode ser observado o recebimento integral do *trailer*.

5.1.4.7 Geração de Endereços para o adaptador Ethernet - `addr_generator`

A interface com o adaptador Ethernet espera receber endereços de onze bits para o *buffer* de transmissão. Esses endereços devem ser gerados pela AAL 5. Para isso, foi projetado um módulo de geração de endereços Ethernet, mostrado na Figura5.14.

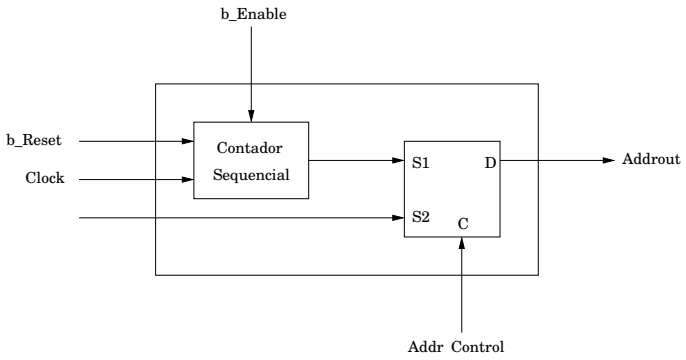


Figura 5.14: Módulo de geração de endereços para o adaptador Ethernet.

O módulo possui um gerador de números em seqüência que produz a saída do gerador de endereços, `Addrout`, quando o sinal de controle `Addr_Control` for 0. Quando `Addr_Control = 1`, a saída do módulo será o endereço contido em `Addrin`. Os sinais de `b_Enable` e `b_Reset` ajudam a controlar o módulo, interrompendo de maneira síncrona a geração de endereços ou

reiniciando-a, de maneira assíncrona.

5.1.4.8 Contadores de Bytes - `counter_d` e `counter_t`

Para manter o controle da informação no que diz respeito à quantidade de bytes transmitidos são utilizados dois contadores, chamados `counter_d` e `counter_t`. O primeiro deles é utilizado para contar a quantidade de bytes de dados recebidos por célula. A contagem vai até 48 se o parâmetro `AUU = 0`, o que significa 48 bytes de dados na célula. Por outro lado, se `AUU = 1`, a contagem pára em 40, porque os últimos 8 bytes da célula não são dados e sim *trailer*. Nesse caso é utilizado o segundo contador, `counter_t`, que conta os oito bytes do *trailer*. Ambos contadores são desabilitados quando o byte em `Datain` não é válido, seja ele dado ou *trailer*. O contador de dados, `counter_d` é carregado com 0 a cada nova célula. Já o contador de *trailer*, `counter_t`, é inicializado com 0 a cada novo pacote, mantendo-se habilitado (i.e., contando) apenas quando a máquina de recepção estiver no estado `TRAILER`.

5.1.5 Estratégia de Validação Funcional do Módulo de Recepção

O módulo de *hardware* para recepção de dados pela AAL 5 foi validado funcionalmente através de simulação na ferramenta Active HDL, da Aldec. Foram feitas simulações com o intuito de prever todas as possíveis situações que ocorrem durante a recepção de dados. Foram extraídos da ferramenta várias formas de onda que mostram os diferentes conjuntos de sinais, com o objetivo de visualizar o que ocorre nos diferentes módulos implementados. As situações ilustradas a seguir são:

1. Início da recepção da primeira célula
   (a) Sinais da máquina de estados da recepção (Figura 5.15)
   (b) Sinais de manipulação de dados (Figura 5.16)
   (c) Sinais externos do módulo `PDU_Handler` (Figura 5.17)
   (d) Parâmetro e CRC (Figura 5.12)
2. Início da recepção da segunda célula
   (a) Sinais externos do módulo `PDU_Handler` (Figura 5.18)
   (b) Sinais da máquina de estados da recepção (Figura 5.19)
   (c) Sinais de manipulação de dados (Figura 5.20)
3. Recepção do *trailer* com CRC correto
   (a) Sinais externos do módulo `PDU_Handler` (Figura 5.21)
   (b) Sinais da máquina de estados da recepção (Figura 5.10)
   (c) *Trailer* e CRC (Figura 5.13)
4. Recepção do *trailer* com CRC incorreto
   (a) Sinais externos do módulo `PDU_Handler` (Figura 5.22)
   (b) Valor do CRC (Figura 5.13)

5. Pausa na recepção dos dados por parte da camada ATM

   (a) Sinais externos do módulo `PDU_Handler` (Figura 5.24)

6. Pausa na recepção do parâmetro por parte da camada ATM

   (a) Sinais externos do módulo `PDU_Handler` (Figura 5.25)

7. Pausa na recepção do *trailer* por parte da camada ATM

   (a) Sinais externos do módulo `PDU_Handler` (Figura 5.26)

5.1.5.1 Formas de Ondas da Simulação Funcional

A Figura 5.15 mostra as trocas de estados dentro da máquina da recepção, as saídas desta máquina e os sinais que causam as trocas de estados. Na figura é possível observar que, no tempo de 120ns, ocorre a troca de estado na máquina de recepção: de `STOP` para `PARAM`. No caso do início da recepção da primeira célula, as outras máquinas de estados permencem em seus estados iniciais. Isso ocorre porque não é o momento para escrita do tamanho do pacote no *buffer* do adaptador Ethernet, nem para o *handhskake* com o mesmo. Nessa figura é importante observar três pontos: A saída `b_Enable` é habilitada quando o módulo é resetado (tempo de 120ns). O parâmetro é capturado na borda de descida seguinte (140ns), porque o sinal `DV` diz que o dado é válido, e automaticamente o sinal `s_auu` recebe o valor enviado pela camada ATM. Por fim, quando a máquina de recepção vai para o estado de `DATA`, tem início o cálculo do CRC (`CRC_Reset = 0` e `CRC_Enable = 1`), a geração de endereços Ethernet (`s_b_Enable_addr_generator = 0`) e a contagem de bytes da célula (`s_b_Enable_counter_d = 0`).

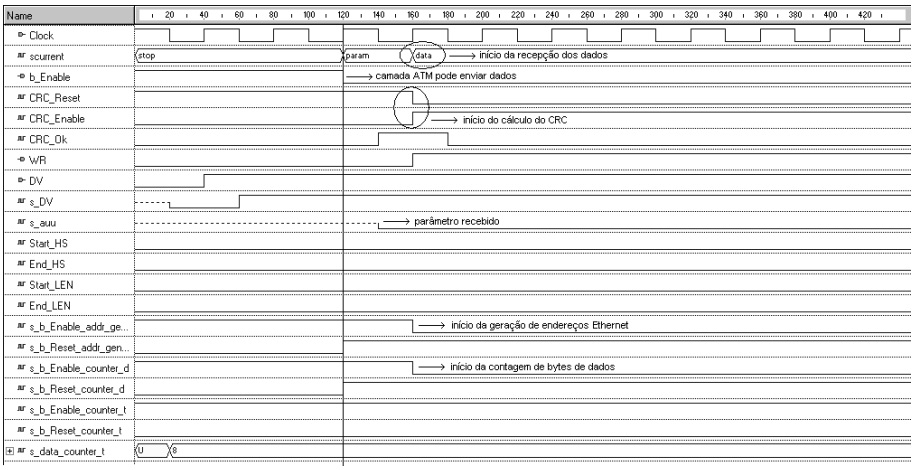


Figura 5.15: Início da recepção da primeira célula - sinais da máquina de estados da recepção.

Na Figura 5.16 é possível observar o controle da quantidade de bytes que chegam da camada ATM. O sinal `s_data_counter_d` é a saída de um contador que tem essa função. O sinal `DV` é uma entrada, fornecida pela camada ATM, que diz se o dado contido em `Datain` é válido ou não.

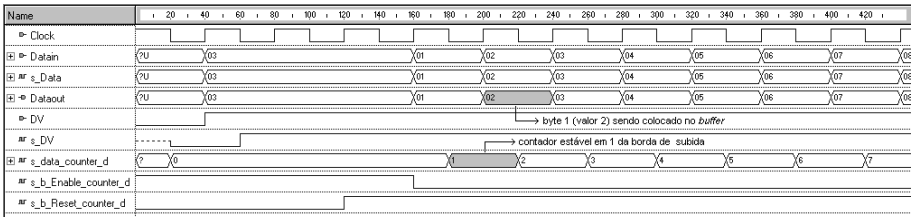


Figura 5.16: Início da recepção da primeira célula - sinais de manipulação de dados.

A Figura 5.17 mostra os sinais externos do módulo `PDU_Handler` quando a primeira célula está sendo recebida. O módulo entra em funcionamento apenas quando o sinal de `b_Reset` é desabilitado. O primeiro dado enviado pela camada ATM é o parâmetro. Ele não é gravado no *buffer* de transmissão do adaptador Ethernet porque a saída `WR = 0`. Quando a camada ATM coloca o segundo byte em `Datain` (o primeiro byte da célula), o sinal de `WR` vai para 1 e o byte (`Dataout = 01`, em hexadecimal) é gravado no primeiro endereço do *buffer*(`Addr = 0`, em decimal).

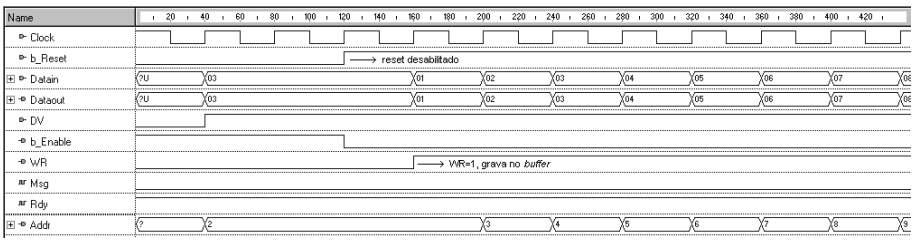


Figura 5.17: Início da recepção da primeira célula - sinais externos do módulo `PDU_Handler`.

Na recepção de várias células de um mesmo pacote, a camada ATM envia uma nova célula assim que a anterior chega ao final. A Figura 5.18 mostra as saídas externas do módulo `PDU_Handler` quando termina a recepção de uma célula e imediatamente após começa a recepção de outra. No instante 2080ns, a camada ATM desabilita o sinal `DV`, informando que o dado contido em `Datain` não é válido. Esse ciclo de *clock* é necessário para que a ATM possa montar o byte de parâmetro que deve ser enviado à AAL 5. No instante 2120ns, a camada ATM garante que o dado em `Datain` é válido e é o parâmetro para a AAL 5. Esse parâmetro não deve ser gravado no *buffer* Ethernet, o que é indicado por `WR = 0`. No instante 2160ns, a camada ATM coloca em `Datain` o primeiro byte da segunda célula para a AAL 5.

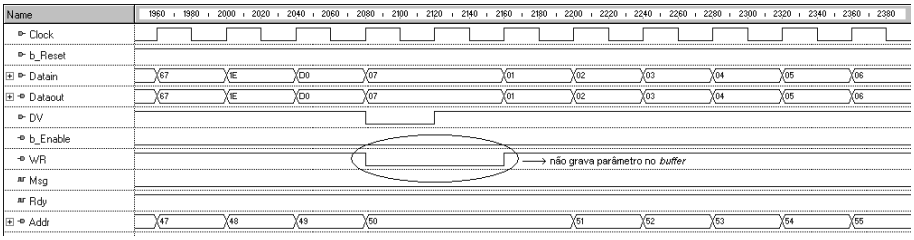


Figura 5.18: Início da recepção da segunda célula - sinais externos do módulo `PDU_Handler`.

A Figura 5.19 apresenta os sinais de saída para a máquina de estados da recepção, quando a primeira célula termina e inicia-se a recepção da segunda. É possível observar que o CRC

não é calculado sobre o parâmetro, pois `CRC_Reset = 1` e `CRC_Enable = 0`, o que desabilita o cálculo do CRC. Outra consideração é com relação ao gerador de endereços para o adaptador Ethernet, que também não está habilitado quando o dado contido em `Datain` é o parâmetro enviado pela camada ATM à AAL 5.

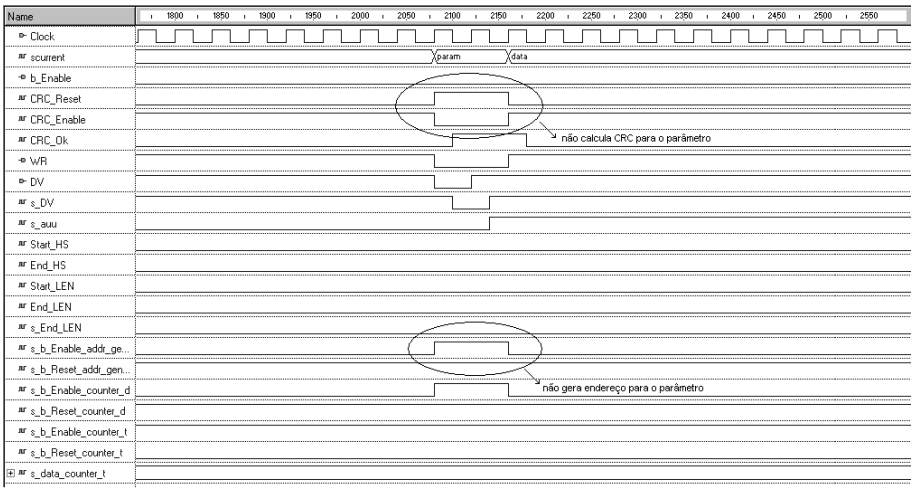


Figura 5.19: Início da recepção da segunda célula - saídas da máquina de estados da recepção.

Observando o contador de bytes da célula, mostrado na Figura 5.20, nota-se que ele é habilitado apenas no instante 2160ns, quando a máquina de estados da recepção troca para o estado de `DATA`, iniciando-se a contagem dos bytes da célula que está sendo enviada pela camada ATM.

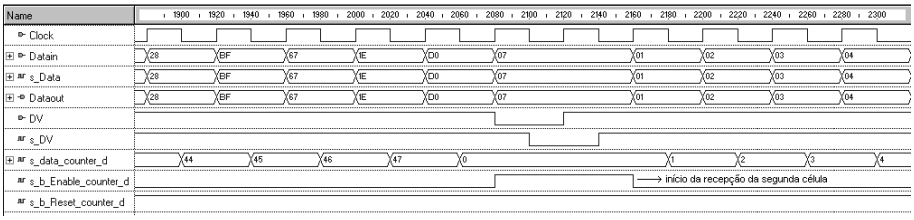


Figura 5.20: Início da recepção da segunda célula - sinais de manipulação de dados.

A Figura 5.21 mostra a recepção do *trailer* de um pacote cujo CRC armazenado confere com o CRC calculado pelo módulo `PDU_Handler`, o que significa que o pacote foi transmitido com sucesso. Assim, o adaptador Ethernet deverá ser informado do tamanho do pacote, através da escrita deste tamanho nas duas últimas posições do *buffer* de transmissão. Após esta operação, ocorre o *handshake* avisando o adaptador que ele pode transmitir os dados do *buffer* através do meio físico.

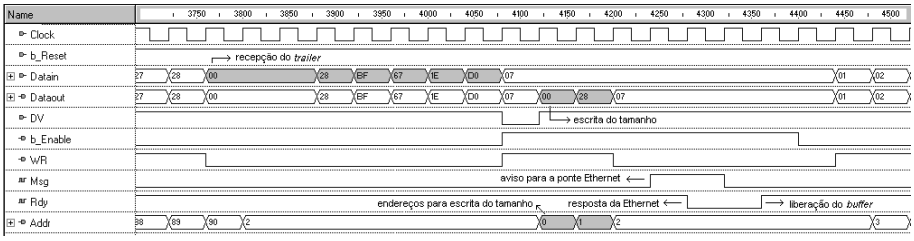


Figura 5.21: Recepção de *trailer* com CRC correto - sinais externos do módulo de recepção.

A Figura 5.22 mostra a recepção do *trailer* com um campo de CRC diferente do calculado. Neste caso, a célula é descartada e uma nova célula começa a ser recebida imediatamente.

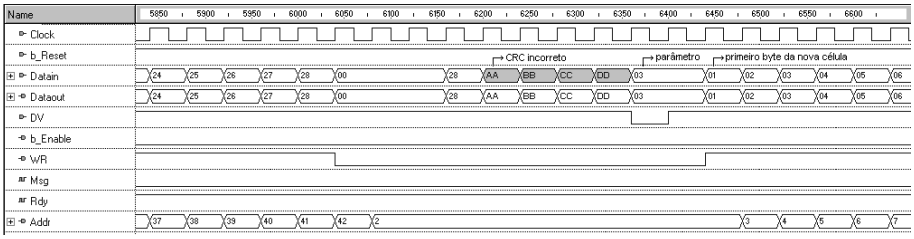


Figura 5.22: Recepção de *trailer* com CRC incorreto - sinais externos do módulo `PDU_Handler`.

Em destaque na Figura 5.23 estão os valores para o CRC. No sinal `Datain` está o valor do CRC enviado no *trailer*. No sinal `s_CRC_Complemented`, o valor do CRC calculado pelo módulo `PDU_Handler`.

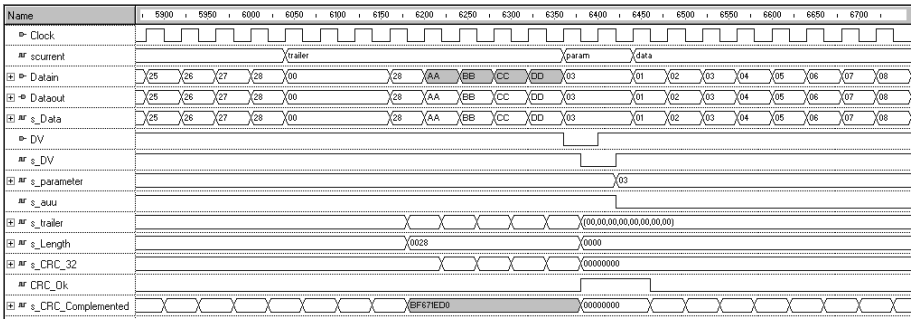


Figura 5.23: Recepção de *trailer* com CRC incorreto - valor do CRC.

Como pode ser observado na Figura 5.24, a camada ATM desabilita o sinal `DV` para indicar que o dado presente em `Datain` não é válido. Não há troca de estado em nenhuma das máquinas implementadas. Quando a camada ATM faz `DV = 1` novamente, a AAL 5 pode escrever o dado no *buffer* do adaptador Ethernet.

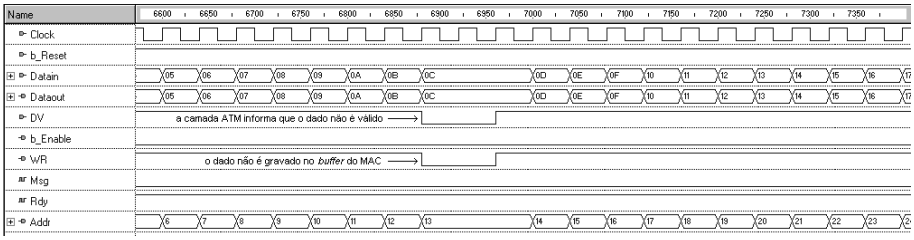


Figura 5.24: Pausa na recepção dos dados por parte da camada ATM.

A camada ATM desabilita o sinal `DV` para indicar que o dado presente em `Datain` não é válido. Não há troca de estado em nenhuma das máquinas. Quando a camada ATM faz `DV = 1` novamente, a AAL 5 pode capturar o parâmetro. Isso pode ser observado na Figura 5.25.

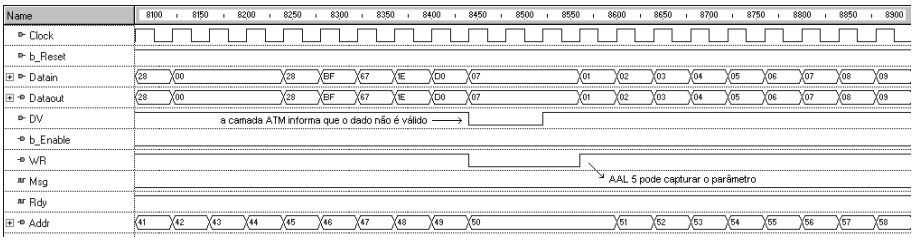


Figura 5.25: Pausa na recepção do parâmetro por parte da camada ATM.

A Figura 5.26 mostra que a camada ATM desabilita o sinal `DV` para indicar que o dado presente em `Datain` não é válido. Não há troca de estado em nenhuma das máquinas. Quando a camada ATM faz `DV = 1` novamente, a AAL 5 pode capturar o *trailer*.

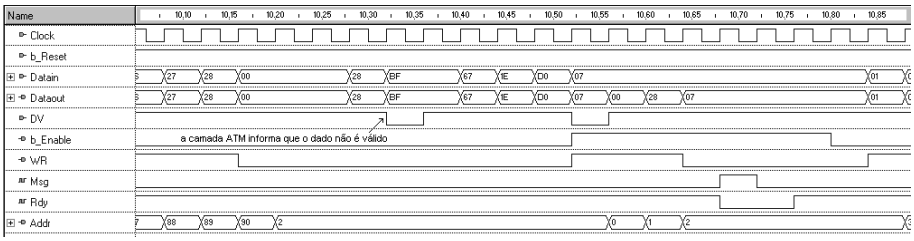


Figura 5.26: Pausa na recepção do *trailer* por parte da camada ATM.

5.2 Avaliação dos Módulos de *Hardware* da AAL 5

A avaliação de desempenho dos módulos implementados para a recepção de dados através da AAL 5 foi realizada com base nas formas de onda obtidas nas simulações que validaram o modelo. Foram determinados o número de ciclos de *clock* consumidos em cada tarefa a ser realizada pela AAL 5, e a freqüência de operação para o pior caso identificado, considerando um ciclo de *clock* com 40ns (25MHz).

Na AAL 5, um pacote da camada superior por estar contido em uma ou mais PDUs. É possível identificar isso através do parâmetro que a camada ATM envia à AAL 5 antes de cada PDU. O módulo de recepção da AAL 5 é capaz de receber da camada ATM um byte a cada

ciclo de *clock*, e escrevê-lo no *buffer* de transmissão do adaptador Ethernet nesta mesma taxa. Assim, cada PDU demora 49 ciclos de *clock* para ser integralmente recebida, considerando o envio do parâmetro antes dos 48 bytes de informação. Pode-se observar então que o tempo gasto com a recepção de um pacote, em número de ciclos, é o seguinte:

$$\text{Tempo } = \text{Número de PDUs necessárias } * 48 \text{ ciclos} \tag{5.1}$$

Considerando que um pacote Ethernet possui um tamanho entre 64 e 1518 bytes, são necessárias no mínimo duas e no máximo 32 PDUs, e o tempo consumido fica entre 96 e 1536 ciclos de *clock*. Assumindo um tamanho médio de pacote de 791 bytes e um tempo médio de 816 ciclos de *clock* para a recepção através da AAL 5, obtém-se a seguinte taxa de operação:

$$\text{Taxa } = \frac{791 \text{ bytes}}{816 \text{ciclos}} = \frac{791}{816 * 40ns} = \frac{791}{32640} = 193,87254902Mbps \tag{5.2}$$

Essa taxa de operação é apenas para a recepção dos pacotes. A operação completa da AAL 5 inclui o tempo gasto no *handshake* com o adaptador Ethernet, uma vez que o *buffer* de transmissão fica bloqueado durante a transmissão dos bytes através do meio físico. Assim, o desempenho da AAL 5 depende diretamente do desempenho do adaptador Ethernet.

No caso de ocorrer algum erro na recepção do pacote, não é consumido tempo extra para seu descarte, uma vez que a informação de `CRC` encontra-se no *trailer*, e o pacote já terá sido totalmente recebido quando o erro for identificado. Apenas o procedimento de *handshake* com o adaptador Ethernet não é realizado.

5.3 Módulo de Transmissão da AAL 5

Um diagrama de blocos da proposta do módulo para transmissão de dados pode ser visto na Figura 5.27. Assim como o módulo de recepção, este possui interfaces com o adaptador Ethernet e com a camada ATM. No lado do adaptador Ethernet deve ser implementado um *handshake*, semelhante ao da recepção, para ler os dados do *buffer* da ponte. No lado da camada ATM, o decodificador de endereços é responsável por determinar se o endereço contido no barramento é o mesmo armazenado na AAL 5 em questão, para que a mesma possa enviar dados à camada ATM. O módulo chamado `SDU_Handler` realiza as funções da AAL 5, como geração de *trailer* e cálculo do `CRC`.

A implementação deste módulo não foi realizada. Por isto, e pelo fato do módulo de transmissão de dados através da AAL 5 ser muito semelhante ao módulo de recepção, não serão feitos maiores comentários a respeito.

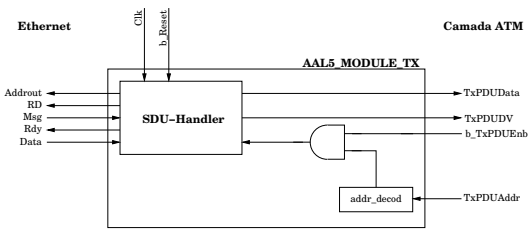


Figura 5.27: Proposta do módulo de transmissão de dados.

Capítulo 6

Implementação em *Hardware* da AAL 2

Assim como a AAL 5, a implementação da AAL 2 foi feita em VHDL, visando a prototipação em uma plataforma de *hardware* comercial. Esta Seção apresenta detalhes da implementação funcional do módulo de recepção e uma proposta de implementação do módulo de transmissão. A validação funcional do módulo de recepção implementado é apresentada na Seção 6.1.5.

De acordo com a Figura 3.6, apresentada na Seção 3.5.1, a AAL 2 é subdividida em SSCS e CPS. A SSCS pode ser nula, o que ocorre neste caso. A CPS tem, então, duas funções principais: no caso da transmissão, a CPS deve receber as informações da camada superior, montar os *CPS-Packets*, a partir deles montar as PDUs e então transmiti-las à camada ATM. No caso da recepção, deve receber as PDUs da camada ATM, recuperar os *CPS-Packets*, extrair as informações da camada superior e repassá-las à Interface DSP. Assim, pode-se estruturar a AAL 2 como mostrado na Figura 6.1.

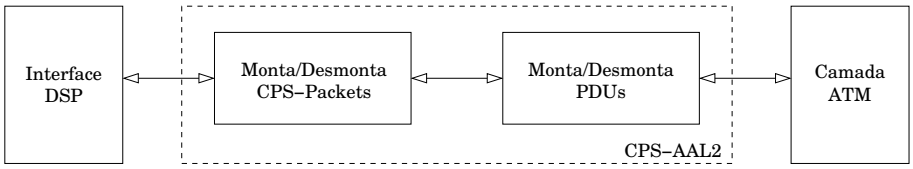


Figura 6.1: Diagrama de blocos da AAL 2.

A AAL 2 foi projetada para receber ou enviar pacotes contendo voz compactada previamente processada, por exemplo, por um DSP (*Digital Signal Processor*), conforme visto na Seção 4.2. De acordo com o DSP escolhido para esse fim, é necessário implementar uma interface com a AAL 2, como mostra a Figura 6.2. Dessa forma, o projeto da AAL 2 fica independente do processador de voz específico. Os pacotes fornecidos para a AAL 2 devem conter apenas os bytes de voz. Qualquer tipo de cabeçalho que seja inserido pelo DSP deve ser tratado pela interface.

Além do pacote de voz, a AAL 2 precisa ser informada sobre qual fonte está originando esse pacote e qual o tamanho do mesmo. A interface deve fornecer esses valores antes de enviar à AAL 2 os bytes que compõem o pacote de voz. A Figura 6.3 mostra o fluxo dos pacotes entre o DSP e a AAL 2. A saída do DSP deve ser o pacote de voz (informação útil) e alguma informação adicional dependente do DSP utilizado. A interface DSP projetada para este *driver* repassa o pacote de voz à AAL 2 e extrai dessa informação adicional o que a mesma necessita. Neste caso, trata-se do tamanho do pacote (`LI`) e a identificação do canal

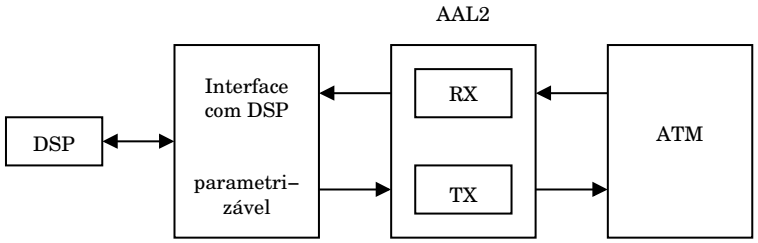


Figura 6.2: Transferência de Voz Compactada pela AAL 2.

ou fonte de voz (`CID`). A AAL 2 utiliza essas informações para compor o cabeçalho de um *CPS-Packet*, acrescentando ainda os campos `UUI` e `HEC`, calculados pela AAL 2. No nosso caso, uma AAL2_PDU é composta de vários `CPS-Packets`, o cabeçalho `OSF` e um campo de preenchimento, `PAD`, que pode ser nulo.

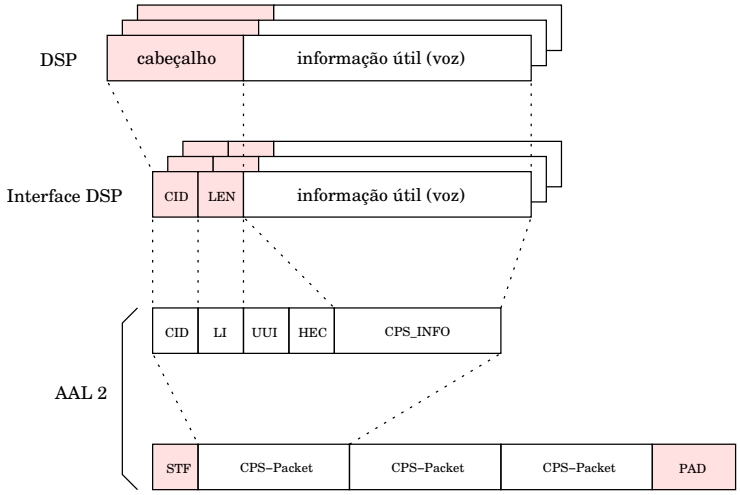


Figura 6.3: Fluxo dos pacotes entre o DSP e a AAL 2.

Cada `CPS-Packet` pode ser proveniente de uma fonte de voz distinta. Assim, é responsabilidade da AAL 2 multiplexar esses pacotes em uma ou mais AAL2_PDUs (porque um pacote pode estar parte em uma AAL2_PDU e parte em outra). Como o canal físico de comunicação da AAL 2 com a Interface DSP é único, essa multiplexação já ocorre intrinsecamente na Interface. A tarefa da AAL 2 é apenas calcular o restante dos campos do cabeçalho do `CPS-Packet` e montar a AAL2_PDU, calculando também o `STF` e determinando se há necessidade de *padding*.

Dessa forma, pode-se estruturar os módulos para implementação como mostra a Figura 6.4, usando um módulo para transmissão e outro para recepção, cada um com as funções relativas aos *CPS-Packets* e às PDUs. A Seção 6.1 apresenta o módulo desenvolvido para recepção de voz compactada através da AAL 2.

6.1 Módulo de Recepção da AAL 2

O diagrama de blocos de mais alto nível da recepção é mostrado na Figura 6.5. As interfaces com a camada ATM e com o DSP são comentadas nas seções seguintes, juntamente com os

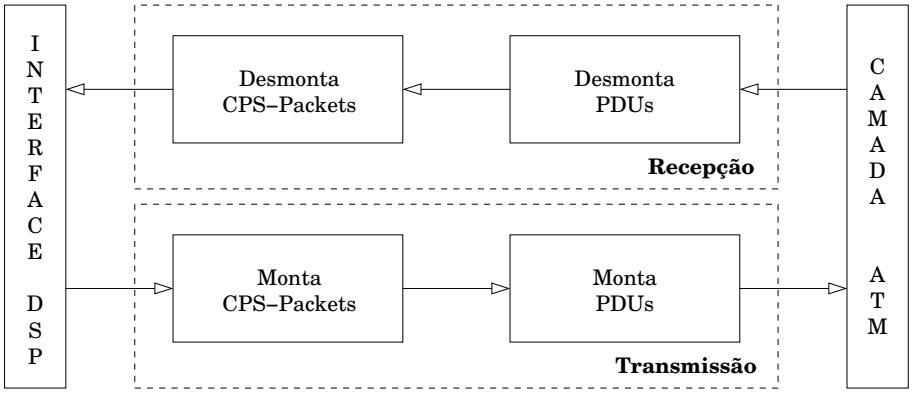


Figura 6.4: Estrutura em módulos a ser implementada para a AAL 2.

módulos internos que compõem a recepção.

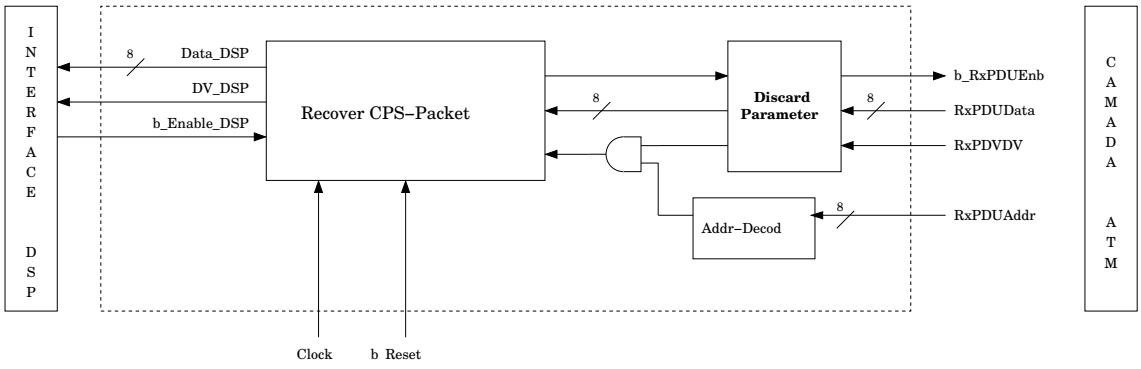


Figura 6.5: Diagrama de blocos do módulo de recepção da AAL 2.

6.1.1 Interfaces da AAL 2 com a Camada ATM e com o DSP

A interface com a camada ATM consiste dos mesmos sinais apresentados na AAL 5. Essa padronização foi mantida para facilitar a implementação da camada ATM. Dessa forma, todos os sinais pertinentes já foram apresentados e comentados na Seção 5.1.1.

A comunicação da AAL 2 com a Interface DSP é feita através dos seguintes sinais:

Data_DSP: Esse barramento tem largura de 8 bits e é por ele que a AAL 2 envia as informações para a Interface DSP. Futuramente, pretende-se parametrizar essa largura de barramento.

DV_DSP: É um sinal de um bit responsável por indicar quando o byte presente em `Data_DSP` é válido (`DV_DSP = 1`) ou não (`DV_DSP = 0`).

b_Enable_DSP: Esse sinal de 1 bit é enviado da Interface DSP para a AAL 2 indicando se a mesma está habilitada a receber novas informações. Quando `b_Enable_DSP = 0`, a Interface pode receber os bytes.

6.1.2 Módulo de Descarte de Parâmetros - `discard_parameter`

O projeto da camada ATM prevê o envio de um byte antes da PDU contendo parâmetros da primitiva de serviço entre a AAL e a camada ATM. Esse parâmetro é utilizado pela AAL 5, mas não é necessário para a AAL 2. Assim, foi feito o projeto de um módulo que descarta esse parâmetro, repassando ao módulo `recover_cps_packet` apenas os 48 bytes da PDU. O módulo responsável por essa função é chamado de `discard_parameter`, e pode ser observado na Figura 6.5.

6.1.3 Módulo Decodificador de Endereços - `addr_decod`

O módulo decodificador de endereços é idêntico, em código e funcionalidade, ao apresentado no Capítulo 5, dispensando comentários.

6.1.4 Módulo de Recuperação de *CPS-Packets* - `recover_cps_packet`

Este é o principal módulo interno na recepção de voz compactada pela AAL 2. Chamado de `recover_cps_packet`, como mostrado na Figura 6.5, esse módulo é responsável pelas funções da CPS. Isso inclui a recepção de PDUs da camada ATM, a recuperação dos *CPS-Packets* contidos nessas PDUs (por isso o nome de *recover_ cps_ packet*), e o envio das informações à Interface DSP.

O funcionamento desse módulo é o seguinte: a camada ATM envia sempre blocos de 48 bytes, onde o primeiro byte corresponde ao campo `STF`, inserido no lado transmissor pela própria AAL 2. Esse campo é verificado quanto à existência de erros através de um bit de paridade, e é determinado o valor do campo `OSF`, que indica onde começa o próximo *CPS-Packet*. Na seqüência são enviados diversos *CPS-Packets*. No caso deste projeto, a implementação é parametrizável em relação ao tamanho do *CPS-Packet*, mas a validação foi feita supondo que cada *CPS-Packet* possui tamanho igual a sete bytes, onde os três primeiros são cabeçalho (*CPS-Packet Header*) e os outros quatro são de informação útil, de acordo com o formato descrito na Seção 3.5.

Cada *CPS-Packet* recebido pela AAL 2 tem seu cabeçalho verificado quanto a erros através do cálculo do `HEC`, conforme visto na Seção 3.5. Como o último *CPS-Packet* pode não estar completo ao final da PDU, esse módulo armazena temporariamente em um *buffer* o início do pacote, até que seja possível verificar o `STF` da próxima PDU e decidir se o pacote pode ser enviado à Interface DSP. Esse procedimento será melhor detalhado ao longo das seções seguintes, que tratam dos módulos internos do *recover_ cps_ packet*. O diagrama de blocos do módulo de recuperação, com os módulos internos e suas interconexões, aparece na Figura 6.6.

6.1.4.1 Máquina de Estados para a Recepção

O processo de recepção de pacotes de voz compactada é controlado por uma máquina de estados, identificada na Figura 6.6 como `AAL_2_Receive_State_Machine`. O diagrama de transição de estados da mesma é apresentado na Figura 6.7.

Esta máquina possui dezesseis estados, denominados `STOP`, `STF_PH`, `GET_CPS_PH`, `WR_CID`, `WR_LI`, `GET_CPS_INFO`, `BUFFER_ST`, `STF_BUFF`, `SEND_BUFF`, `WR_CID_BUFF`, `WR_LI_BUFF`, `STF_REM`, `ERROR_HEC`, `ERROR_STF`, `REMAINDER_ST` e `PAD`. A máquina fornece oito sinais de saída. Os seis primeiros sinais de saída são para controle dos contadores. `s_b_enable_counter_pdu` e `s_b_reset_counter_pdu` para o contador de bytes da PDU, `s_b_enable_counter_cps_p` e `s_b_reset_counter_cps_p` para o contador de bytes do pacote, e `s_b_enable_counter_buffer`

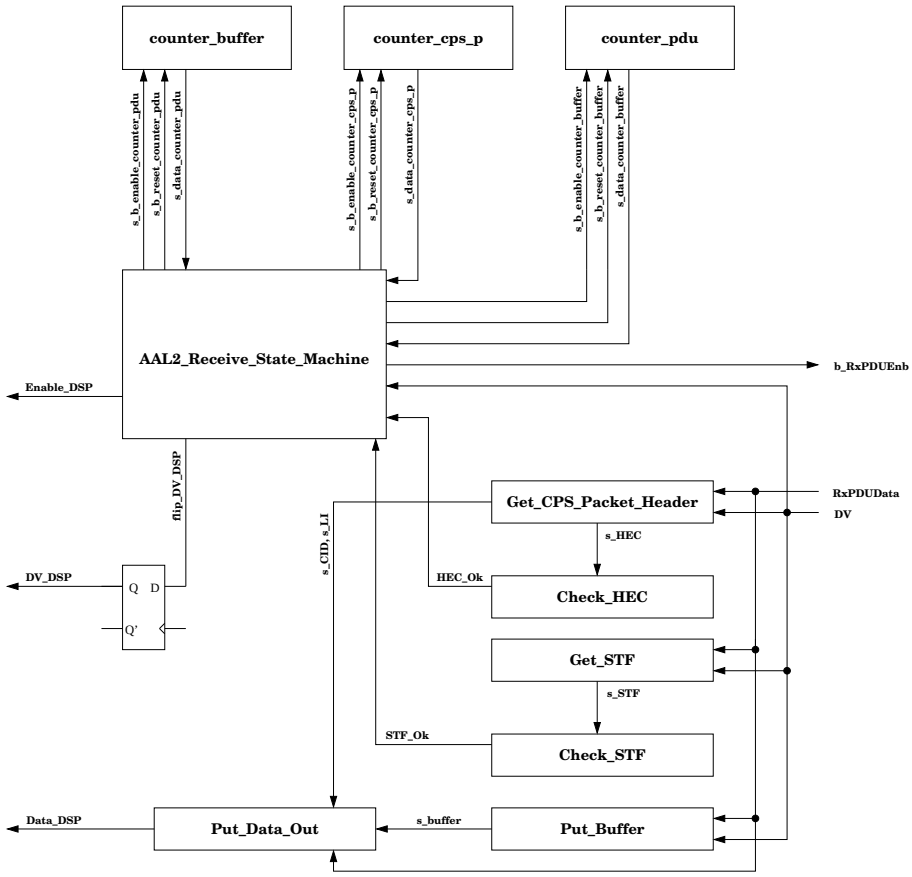
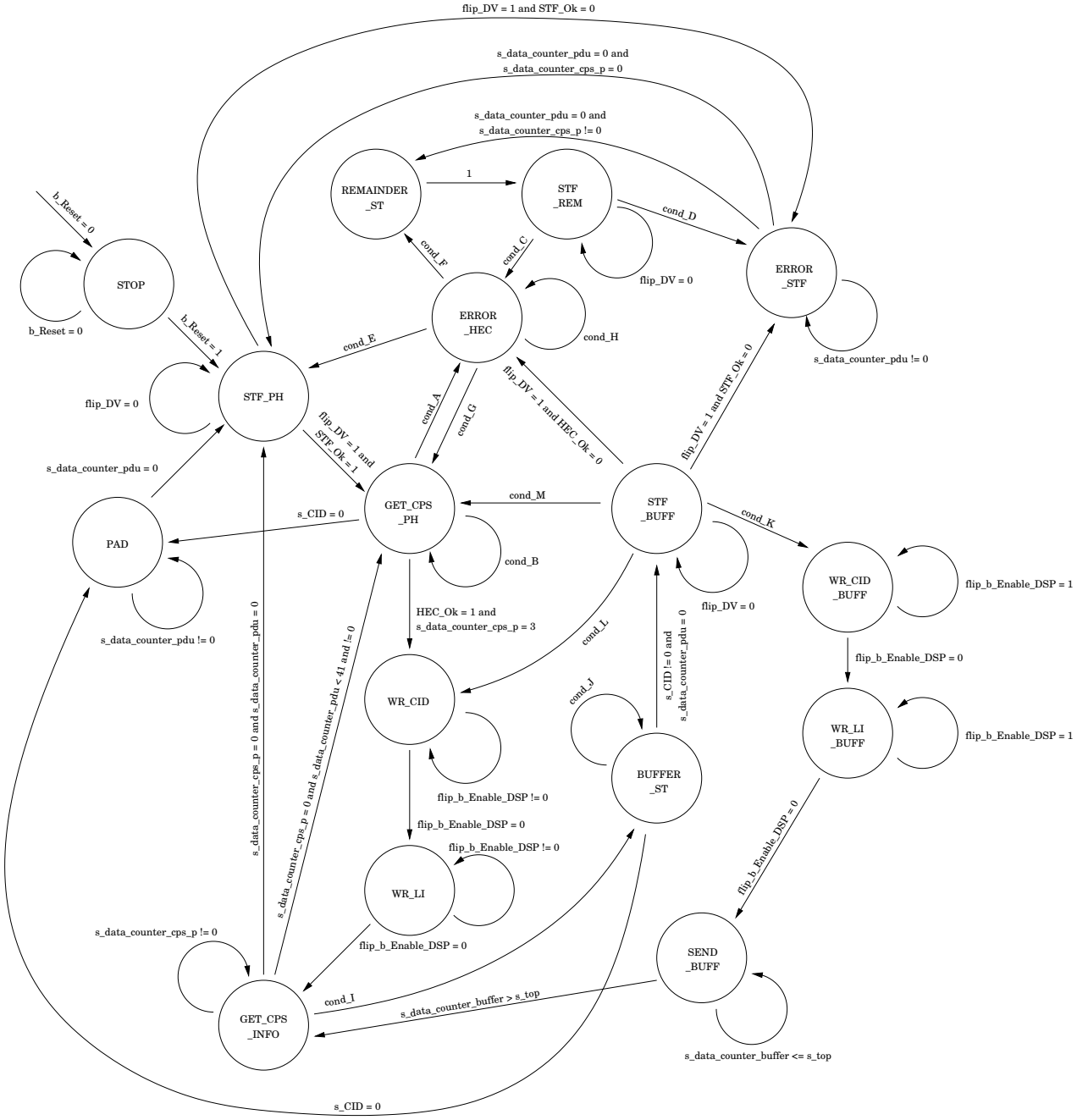


Figura 6.6: Diagrama de blocos do módulo *recover_ cps_ packet*.

e `s_b_reset_counter_buffer` para o contador de bytes do *buffer*. Esse contadores serão comentados adiante. O sinal `flip_DV_DSP` é a saída que indica quando o dado é válido para o DSP. Essa saída deverá ser atrasada meio ciclo de *clock* antes de ser enviada à Interface DSP. Por fim, `b_RxPDUEnb` é um sinal que indica para a camada ATM quando a AAL 2 pode ou não receber mais bytes.



cond_A : s_CID = 1 and s_data_counter_cps_p = 3 and HEC_Ok = 0
cond_B : s_data_counter_cps_p != 3
cond_C : flip_DV = 1 and STF_Ok = 1
cond_D : flip_DV = 1 and STF_Ok = 0
cond_E : s_data_counter_pdu = 0 and s_data_counter_cps_p = 0
cond_F : s_data_counter_pdu = 0 and s_data_counter_cps_p != 0
cond_G : s_data_counter_pdu != 0 and s_data_counter_cps_p = 0
cond_H : s_data_counter_pdu != 0 and s_data_counter_cps_p != 0
cond_I : s_data_counter_cps_p = 0 and s_data_counter_pdu < 41
cond_J : s_CID != 0 and s_data_counter_pdu != 0
cond_K : flip_DV = 1 and HEC_Ok = 1 and s_data_counter_cps_p > 3
cond_L : flip_DV = 1 and HEC_Ok = 1 and s_data_counter_cps_p = 3
cond_M : flip_DV = 1 and s_data_counter_cps_p < 3

Figura 6.7: Máquina de estados da recepção através da AAL 2.

A funcionalidade associada a cada estado é descrita seguir:

STOP: Esse é o estado inicial da FSM. Nenhuma função é executada neste estado, e a máquina permanece nele enquanto a recepção não tiver início.

STF_PH: Este é um dos estados onde o campo de `STF` é recuperado. Como existem diversas situações para detectar o final de PDU, a máquina de estados para a recepção possui três estados distintos onde se recupera o `STF`, dependendo do que ocorre ao final da PDU. No estado `STF_PH` recupera-se o `STF` quando for a primeira PDU da recepção (mostrado na Figura 6.8), quando a PDU anterior terminar com um campo de enchimento (*PAD*), ou quando a PDU terminar junto com um *CPS-Packet*.

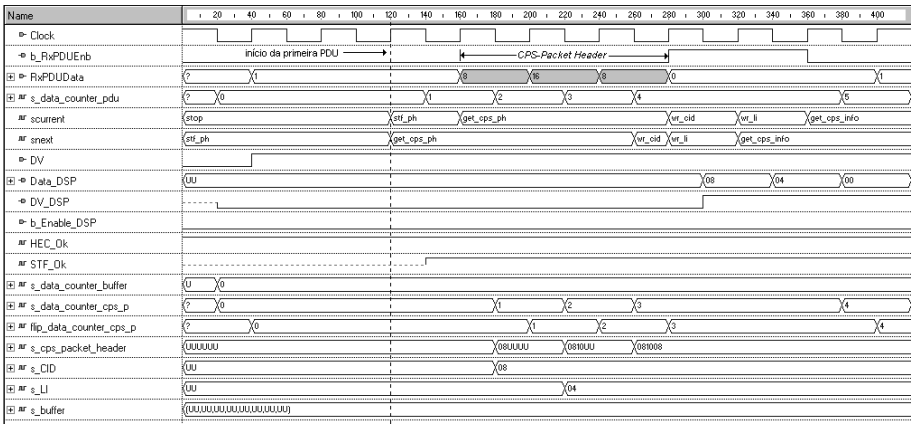


Figura 6.8: Recebimento sem erros da primeira PDU. A máquina de estados da recepção passa para `STF_PH` para receber o primeiro byte da transferência, que contém o `STF`. Após verificar que o valor contido no `STF` está correto, a máquina prossegue a recepção no estado `GET_CPS_PH`, onde recebe o *CPS-Packet Header*, conforme mostra o diagrama de tempos.

GET_CPS_PH: Neste estado são recebidos os bytes que compõem o cabeçalho do *CPS-Packet*. Os valores recebidos são separados em `CID`, `LI` (tamanho do pacote), `UUI` e `HEC`, respectivamente. Nesse estado também é calculado o valor do *HEC* para ser comparado ao campo `HEC` recebido. Se forem iguais, a máquina de estados segue recebendo os bytes de informação útil. Caso contrário, o cabeçalho é desconsiderado e os bytes de informação são recebidos e descartados.

WR_CID: Uma vez que tenha sido recebido todo o cabeçalho do *CPS-Packet* e o *HEC* esteja correto, a Interface DSP pode ser informada do valor do campo `CID` e do tamanho do pacote que irá receber. O campo `CID` é enviado à Interface DSP neste estado.

WR_LI: Neste estado envia-se o tamanho do pacote, ou seja, o campo `LI`, à Interface DSP.

GET_CPS_INFO: É o estado onde são recebidos os bytes de informação provenientes da camada ATM. Estando neste estado, tais bytes são repassados diretamente à Interface DSP.

BUFFER_ST: Quando uma PDU chega ao final, pode ocorrer que um *CPS-Packet* ocupe parte de uma PDU, e parte da PDU seguinte. Neste caso, existem dois cuidados a serem tomados: o primeiro diz respeito ao campo `HEC`. Se o mesmo for verificado e houver erro,

o pacote precisa ser descartado, inclusive a parte que se encontra na PDU seguinte. O segundo cuidado diz respeito ao campo `STF`. Se houver erro no `STF` da PDU seguinte, toda a PDU deverá ser descartada. Como só devem ser descartados *CPS-Packets* inteiros, o início do primeiro pacote, que estava na PDU anterior, e o final do último, que estará na próxima PDU, também precisam ser descartados, para que não haja inconsistência nas informações. Assim, é conveniente que esses dados sejam armazenados temporariamente em um `buffer` até que seja possível determinar se eles devem ser enviados à Interface DSP ou descartados. Tal armazenamento temporário é realizado neste estado. Os bytes de informação útil do final da PDU que fazem parte de um *CPS-Packet* incompleto são armazenados em um *buffer*. Dependendo do ponto onde a PDU termina, a máquina de estados deve reagir de forma a completar o recebimento do pacote. Existem quatro situações distintas onde a PDU pode terminar: (1) durante o recebimento dos bytes do cabeçalho, (2) exatamente entre o cabeçalho e os bytes de informação, (3) durante o recebimento dos bytes de informação, ou (4) exatamente junto com um *CPS-Packet*. Como os bytes do cabeçalho não são enviados na íntegra para a Interface DSP, eles não precisam ser armazenados no *buffer*. Se for determinado que o `CID` e o `LI` devem ser repassados à Interface, isso será feito nos respectivos estados da máquina: `WR_CID` e `WR_LI` (exceto no caso (3), onde são utilizados os estados `WR_CID_BUFF` e `WR_LI_BUFF`). Assim, apenas para a situação (3) é necessário utilizar o *buffer*, como mostrado na Figura 6.9.

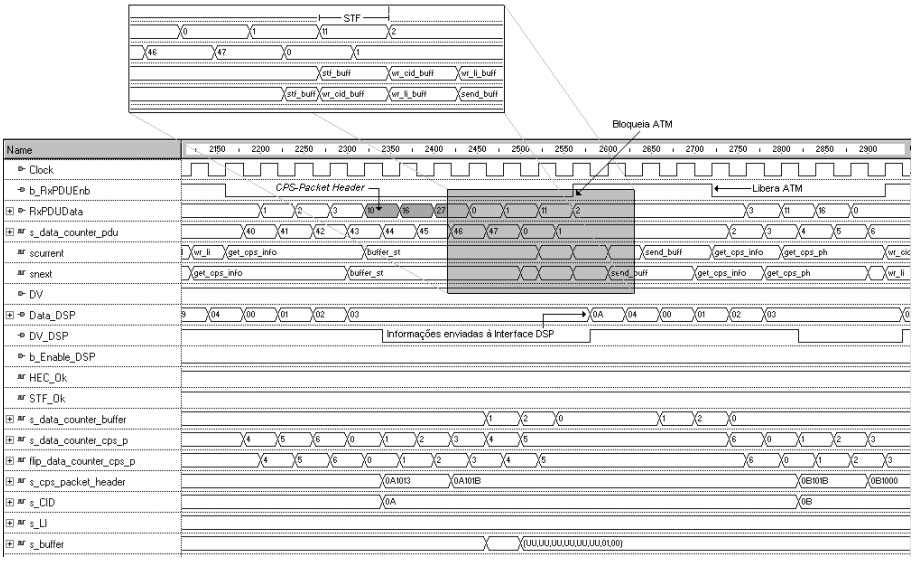


Figura 6.9: Recebimento do final de uma PDU durante os bytes do *CPS-Packet Info* sem erros. Durante o estado de armazenamento temporário são recebidos todos os bytes que pertencem ao *CPS-PH* e ainda dois bytes do *CPS-Packet Info*, que são efetivamente armazenados em um *buffer* auxiliar (conforme visto na Seção 6.1.4). O detalhe na figura mostra o instante em que a PDU termina e o primeiro byte da próxima (`STF`) é recebido. Como não há erros e os valores de `CID` e `LI` são conhecidos (porque o cabeçalho já foi completamente recebido), a máquina de estados encarrega-se de enviar esses valores à Interface DSP. Os bytes que foram armazenados no *buffer* são então enviados à Interface, no estado `SEND_BUFF`. Após isto, a AAL 2 libera a camada ATM para que ela possa enviar o restante dos bytes do *CPS-Packet Info*.

STF_BUFF: Este é outro estado onde se recupera o campo `STF`. Este estado atua sempre que for identificado um *CPS-Packet* incompleto ao final da PDU, mesmo que o *buffer* não seja efetivamente utilizado, como comentado no item anterior.

ERROR_STF: Sempre que um erro de `STF` for identificado, a máquina passa para este estado e consome toda a PDU. Se a mesma terminar com um *CPS-Packet* incompleto, o restante do pacote, que está presente no início da próxima, PDU também será descartado.

SEND_BUFF: Se foram colocados bytes de informação útil no *buffer*, então esses bytes devem ser transmitidos à Interface DSP antes que novos bytes possam ser recebidos da camada ATM. Essa é a função deste estado, que pode ser observada no detalhe da Figura 6.9.

WR_CID_BUFF: Quando a PDU termina durante o recebimento das informações úteis do *CPS-Packet*, os valores de `CID` e `LI` já foram determinados, mas ainda não foram repassados à Interface DSP. Assim, após verificar a validade do `STF`, a máquina deve ir para um estado que envie o `CID` e siga o fluxo de transmissão para a Interface DSP. O estado `WR_CID_BUFF` faz exatamente a mesma função do que `WR_CID`, com a diferença que `WR_CID_BUFF` é usado quando houve armazenamento temporário e há bytes no *buffer*.

WR_LI_BUFF: Assim como `WR_CID_BUFF`, este estado realiza a mesma função de `WR_LI`, exceto que este estado também é usado apenas quando há bytes no *buffer*. As funções dos estados `WR_CID_BUFF` e `WR_LI_BUFF` podem ser observadas também no detalhe da Figura 6.9.

ERROR_HEC: Se um erro no *HEC* for detectado, a máquina passa para este estado, onde o *CPS-Packet* é descartado. Se o pacote em questão estiver no final da PDU, com a parte restante na PDU seguinte, a máquina deverá passar para o estado que sinaliza esse fato (`REMAINDER_ST`), para que o início da próxima PDU também seja desconsiderado.

REMAINDER_ST: Neste estado, a máquina sinaliza que o início da próxima PDU deverá ser descartado. Isso pode ocorrer devido a um erro no `STF`, que causou o descarte de uma PDU completa e o final do último *CPS-Packet*, que está dividido em duas PDUs, ou devido a um erro no *HEC*, que causou o descarte de um *CPS-Packet*, também dividido em duas PDUs. A Figura 6.10 mostra, no detalhe, uma situação em que a máquina passa para o estado de `REMAINDER_ST`.

STF_REM: É o terceiro estado onde se recupera o *STF*. Ele é chamado quando ocorreu algum erro e existem bytes no início da próxima PDU que precisam ser descartados. Essa situação também pode ser observada no detalhe da Figura 6.10

PAD: Esse estado é responsável por consumir os bytes do campo de enchimento, que podem estar contidos no final de uma PDU.

6.1.4.2 Captura do *CPS-Packet Header* - `get_cps_packet_header`

Esse módulo é responsável por capturar o cabeçalho do *CPS-Packet*. Os três primeiros bytes do *CPS-Packet* são armazenados em um registrador auxiliar, chamado `cps_packet_header`, e não são enviados à Interface DSP. O cabeçalho também é armazenado em outros quatro sinais, resgatando os valores dos campos `CID`, `LI`, `UUI` e `HEC`, individualmente.

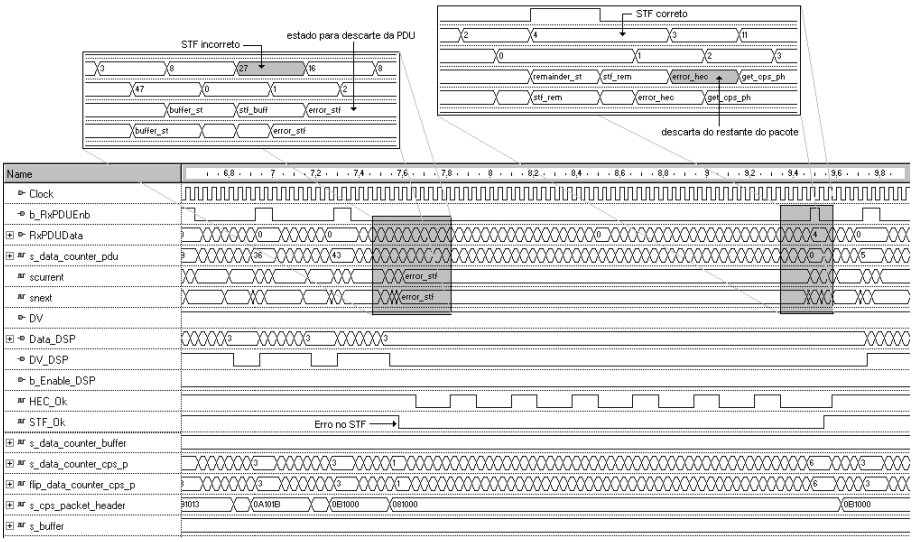


Figura 6.10: Recebimento de uma PDU com erro no `STF` quando a PDU anterior termina durante os bytes do *CPS-Packet Header*. A figura mostra a FSM no estado de `ERROR_STF`, onde toda a PDU recebida é descartada. Isso significa que os pacotes contidos nessa PDU não são repassados à Interface DSP, como indica o sinal `DV_DSP`. No detalhe à esquerda pode ser visto o início do recebimento da PDU com erro no `STF`, e a máquina de estados desconsiderando o que foi recebido durante o armazenamento temporário e passando ao estado responsável pelo descarte da PDU (`ERROR_STF`). No detalhe à direita pode ser visto o final do recebimento da PDU que está sendo descartada. Mesmo o `STF` da PDU seguinte estando correto, restam bytes nesta PDU que pertencem a pacotes da PDU anterior, que foi descartada. Assim, esses bytes também precisam ser descartados. Dessa forma, a máquina passa para o estado `REMAINDER_ST`, indicando que os primeiros bytes da próxima PDU também devem ser descartados. A seguir, é recebido o `STF` da nova PDU e, mesmo estando correto, a máquina passa para um estado de descarte que irá desconsiderar os bytes até o final do *CPS-Packet*. Ao início de um novo *CPS-Packet*, a máquina retorna ao fluxo de execução normal.

6.1.4.3 Verificação do *HEC* - `check_hec`

Uma vez que o cabeçalho tenha sido totalmente recebido, é preciso verificar se não ocorreu erro durante a transmissão do pacote. Isso é feito através do campo de `HEC`. O polinômio $x^5 + x^2 + 1$ é dividido (módulo 2) pelos 19 primeiros bits do cabeçalho, ou seja, os campos `CID`, `LI` e `UUI`, e os coeficientes do resto dessa divisão são comparados ao conteúdo do campo `HEC`. Se forem iguais, não houve erro durante a transmissão do pacote.

A Figura 6.11 mostra uma situação de recebimento de *CPS-Packet* com erro no campo `HEC`, e o pacote sendo descartado. Na figura é possível observar que os bytes do pacote com erro não são repassados à Interface DSP.

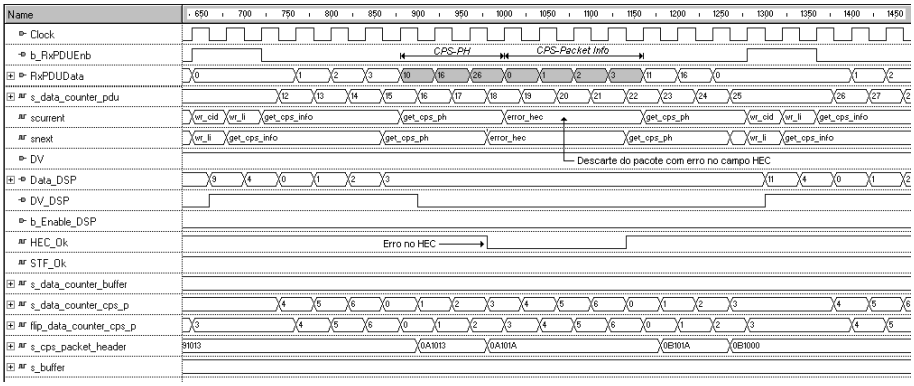


Figura 6.11: Recebimento de um *CPS-Packet* contido em uma única PDU com erro no `HEC`.A figura mostra o recebimento de um *CPS-Packet* completo, porém com erro no `HEC`, indicado pelo sinal `HEC_Ok` em 0. A máquina passa para o estado de descarte de pacote (`ERROR_HEC`) e permanece até que todo o restante do pacote seja recebido, passando então para a recepção de um novo pacote.

6.1.4.4 Captura do *Start Field* - `get_stf`

O *Start Field* (`STF`) é um campo que ocupa exatamente o primeiro byte de uma PDU. Sempre que a máquina de estados de recepção estiver em um dos três estados para recepção do `STF` (`STF_PH`, `STF_BUFF` ou `STF_REM`), o byte contido nas linhas de dados contém o *Start Field*. Assim como o cabeçalho do *CPS-Packet*, esse byte é particionado em três campos distintos: `OSF`, `SN` e `P`, e não é enviado à Interface DSP.

6.1.4.5 Verificação do *STF* - `check_stf`

Após receber o `STF` é feita uma verificação quanto a erros, usando código de paridade ímpar. Essa verificação é simples: o número de bits em "1", incluindo o bit de paridade, deve ser um número ímpar. Se for, significa que não há erro simples na PDU. A Figura 6.12 mostra um caso em que o `STF` é recebido com erro e toda a PDU é descartada, junto com o início da PDU seguinte, que contém o final do último *CPS-Packet* da PDU com erro.

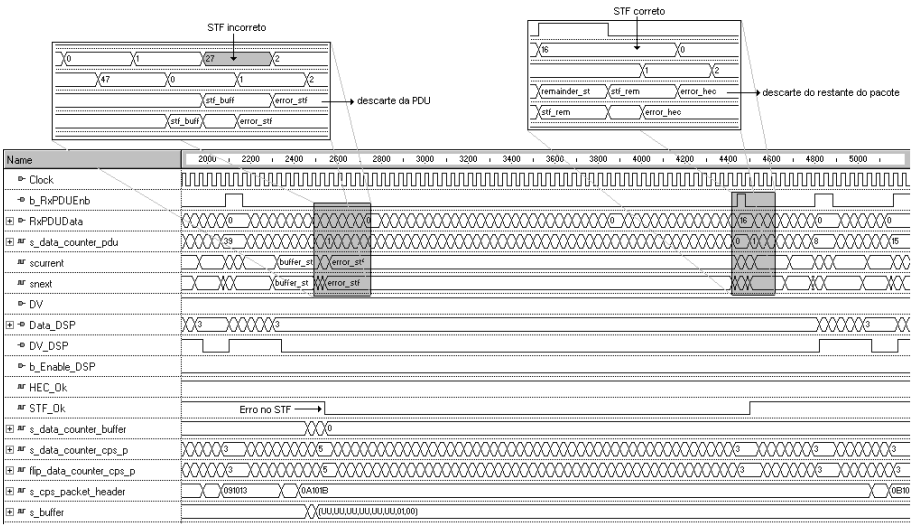


Figura 6.12: Recebimento de uma PDU com erro no `STF` quando a PDU anterior termina durante os bytes do *CPS-Packet Info*. Mostra-se o descarte da PDU cujo `STF` está incorreto. No detalhe à esquerda aparece o início do descarte. Durante o armazenamento temporário haviam sido recebidos todos os bytes do *CPS-PH* e alguns bytes do *CPS-Packet Info*. Devido ao erro no `STF`, os bytes armazenados no *buffer* não poderão ser enviados à Interface DSP, uma vez que o pacote em questão não foi recebido completamente, e o seu final encontra-se na PDU com erro. À direita, no detalhe, é mostrado o final do descarte da PDU. Como o último pacote ainda não foi totalmente descartado, a máquina passa para o estado de `REMAINDER_ST`, indicando que o início da PDU seguinte, mesmo sem erro, deverá ser descartado.

6.1.4.6 Preenchimento do *Buffer* Auxiliar - `put_buffer`

Antes que a informação contida em um *CPS-Packet* seja enviada à Interface DSP, é preciso garantir que o pacote não contém erro. Se o mesmo encontra-se todo em uma única PDU, a verificação quanto a erro é feita através do `HEC`. Por outro lado, se o pacote está dividido em duas PDUs, o campo de `STF` também deve ser utilizado. Se houver erro no `STF`, toda a PDU deverá ser descartada, o que inclui o final do *CPS-Packet* que estava sendo recebido. Como não se pode enviar à Interface DSP um pacote incompleto, o início do mesmo, que estava na PDU anterior, também deve ser descartado. Dessa forma, quando chega-se ao final de uma PDU, primeiro é necessário verificar o *Start Field* e ter certeza de que não há erro. Após, é verificado o *HEC* e só então o pacote deve ser repassado à Interface DSP. Nestes casos é utilizado um *buffer* auxiliar, que receberá os bytes de informação do *CPS-Packet* enquanto não for possível validar o pacote. O módulo chamado de `put_buffer` é responsável por direcionar os bytes recebidos da camada ATM para esse *buffer*, e mantê-los até que seja possível sua transmissão.

6.1.4.7 Saída de Dados - `put_Data_Out`

Dependendo do estado em que a máquina se encontra, um tipo diferente de informação deverá ser repassada à Interface DSP. Esse módulo é responsável por gerenciar essa saída, que pode ser o valor do campo `CID`, o tamanho do pacote (`LI`), os dados diretamente da camada ATM, ou os dados contidos no *buffer*. Na Figura 6.13 podem ser observados os bytes repassados à Interface DSP

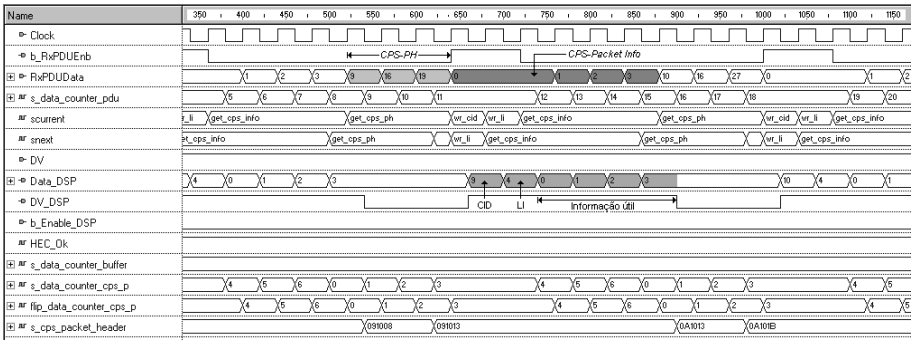


Figura 6.13: Recebimento de um *CPS-Packet* sem erros. Na figura é possível observar os bytes do *CPS-Packet* sendo repassados à Interface DSP. Apenas o `CID`, o tamanho (`LI`) e os bytes de informação útil são repassados. Os demais são consumidos pela AAL 2.

6.1.4.8 Contadores - `counter_pdu`, `counter_cps_p` e `counter_buffer`

Para controlar o que está sendo recebido da camada ATM e o que está sendo repassado à Interface DSP, são utilizados três contadores, identificados na Figura 6.6 como `counter_pdu`, `counter_cps_p` e `counter_buffer`. O primeiro é utilizado para contar a quantidade de bytes recebidos por PDU. A contagem é feita de 0 a 47, pois a PDU tem 48 bytes de tamanho. O segundo é responsável pela contagem dos bytes do pacote, que é de 0 até um limite configurável. Na validação da implementação realizada, foi utilizado um limite de sete bytes, ou seja, três para o cabeçalho e quatro para a informação. O terceiro contador, `counter_buffer`, tem duas funções: controlar a quantidade de bytes que foram armazenados no *buffer*, e gerenciar a retirada desses bytes no momento em que eles puderem ser repassados à Interface DSP.

6.1.5 Estratégia de Validação Funcional do Módulo de Recepção

O módulo de recepção de voz compactada foi validado funcionalmente através de simulações na ferramenta Active HDL, da Aldec. Foram realizadas extensas simulações, com o objetivo de prever a maior parte das possíveis situações que podem ocorrer durante a recepção de PDUs pela AAL 2.

Algumas das situações citadas a seguir já foram apresentadas durante os comentários sobre a implementação dos módulos. As demais, ainda não comentadas, são apresentadas ao final desta seção. As situações simuladas são as seguintes:

1. Recebimento de PDUs sem erros:
   (a) Recebimento da primeira PDU, proveniente da Camada ATM (Figura 6.8)
   (b) Final de uma PDU durante o recebimento de bytes do *CPS-Packet Header* (Figura 6.14)
   (c) Final de uma PDU exatamente entre o *CPS-Packet Header* e o *CPS-Packet Info* (Figura 6.15)
   (d) Final de uma PDU durante o recebimento de bytes do *CPS-Packet Info* (Figura 6.9)
   (e) Final de uma PDU junto com o final de um *CPS-Packet* (Figura 6.16)
2. Recebimento de PDUs com erro no `STF`:
   (a) Final de uma PDU durante o recebimento de bytes do *CPS-Packet Header* (Figura 6.10)

(b) Final de uma PDU exatamente entre o *CPS-Packet Header* e o *CPS-Packet Info* (Figura 6.17)

(c) Final de uma PDU durante o recebimento de bytes do *CPS-Packet Info* (Figura 6.12)

(d) Final de uma PDU junto com o final de um *CPS-Packet* (Figura 6.18)

(e) Erro no STF da PDU que termina junto com um *CPS-Packet* (Figura 6.19)

3. Recebimento de PDUs com paradas:

(a) Pausa no recebimento de um *CPS-Packet Header* de pacote contido na mesma PDU (Figura 6.20)

(b) Pausa no recebimento de um *CPS-Packet Info* de pacote contido na mesma PDU (Figura 6.21)

(c) Pausa no recebimento de um *CPS-Packet Header* durante o armazenamento temporário (Figura 6.22)

(d) Pausa no recebimento de um *CPS-Packet Info* durante o armazenamento temporário (Figura 6.23)

(e) Pausa na escrita do `CID` (Figura 6.24)

(f) Pausa na escrita do `LI` (Figura 6.25)

(g) Pausa no recebimento de um `STF` (Figura 6.26)

(h) Pausa durante o descarte de um pacote (Figura 6.27)

(i) Pausa na escrita da informação útil (Figura 6.28)

4. Recebimento de PDUs com erro no `HEC`:

(a) Recebimento de um *CPS-Packet* contido em uma única PDU (Figura 6.11)

(b) Final de uma PDU durante o recebimento de bytes do *CPS-Packet Header* (Figura 6.29)

(c) Final de uma PDU exatamente entre o *CPS-Packet Header* e o *CPS-Packet Info* (Figura 6.30)

(d) Final de uma PDU durante o recebimento de bytes do *CPS-Packet Info* (Figura 6.31)

(e) Final de uma PDU junto com o final de um *CPS-Packet* (Figura 6.32)

6.1.5.1 Formas de Onda da Simulação Funcional

Na Figura 6.14 é mostrada a máquina de estados iniciando o armazenamento temporário, faltando apenas dois bytes para o final da PDU. Esses dois bytes pertencem ao *CPS-Packet Header*. O terceiro byte do *CPS-PH*, bem como os quatro bytes do *CPS-Packet Info* encontram-se na PDU seguinte. Após receber o `STF`, a máquina conclui o recebimento do *CPS-PH* e pode então enviar os valores de `CID` e `LI` à Interface DSP.

A Figura 6.15 mostra que, durante o armazenamento temporário, foram recebidos os três bytes que compõem o *CPS-Packet Header*. A seguir, é recebido o byte `STF`. Como não há erro, os valores de `CID` e `LI`, que já são conhecidos, podem ser encaminhados à Interface DSP.

A Figura 6.16 mostra o último byte de um *CPS-Packet Info* coincidindo com o último byte de uma PDU. Não há necessidade de armazenamento temporário, por que não há bytes pertencentes a este último pacote na PDU seguinte. Após o recebimento do `STF`, inicia-se o recebimento de um novo pacote.

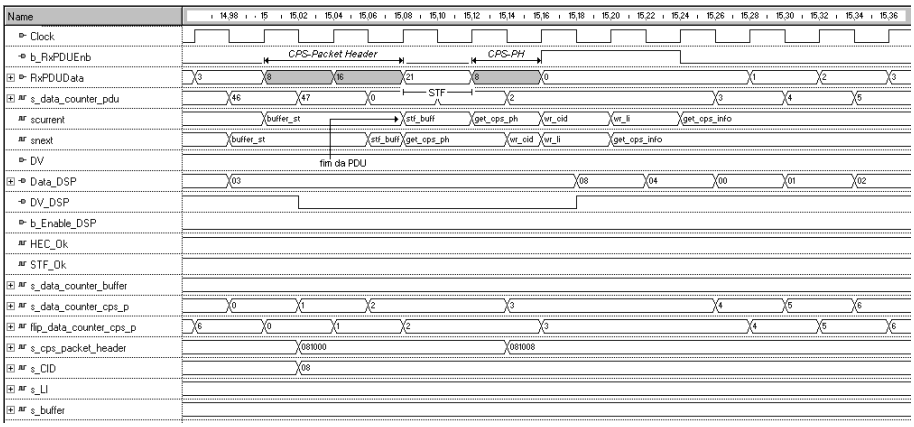


Figura 6.14: Recebimento do final de uma PDU durante os bytes do *CPS-Packet Header* sem erros.

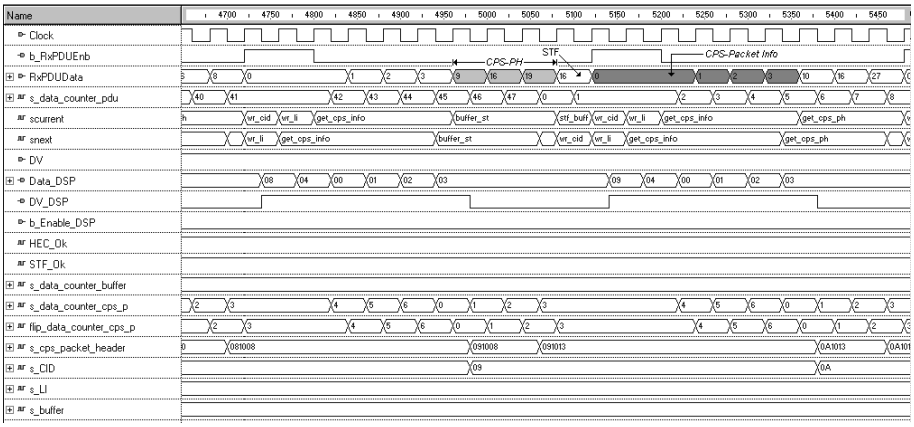


Figura 6.15: Recebimento do final de uma PDU exatamente entre os bytes do *CPS-Packet Header* e do *CPS-Packet Info* sem erros.

Na Figura 6.17 é mostrado o descarte da PDU, quando a máquina está no estado `ERROR_STF`. O detalhe à esquerda mostra o início do descarte da PDU, quando o `STF` é recebido e o erro é constatado. Durante o armazenamento temporário, todos os bytes do *CPS-Packet Header* foram recebidos, mas deverão ser descartados. Isso ocorre porque o restante do pacote encontra-se na PDU com erro. No detalhe à direita é mostrado o final do descarte da PDU. Para que o descarte seja sempre de pacotes inteiros, o início da PDU seguinte, mesmo sem erro, também deverá ser desconsiderado.

A máquina de estados permanece em `ERROR_STF` durante o recebimento da PDU, para descartá-la. À esquerda, no detalhe da Figura 6.18, é mostrado o início da operação de descarte da PDU. O *CPS-Packet Info* da PDU anterior à com erro já havia sido totalmente recebido. Assim, não houve armazenamento temporário no recebimento da PDU anterior e, portanto, não há bytes a desconsiderar. No detalhe à direita é mostrado o final do descarte da PDU. Neste caso o último pacote está incompleto, o que faz com que os primeiros bytes da próxima PDU precisem ser descartados.

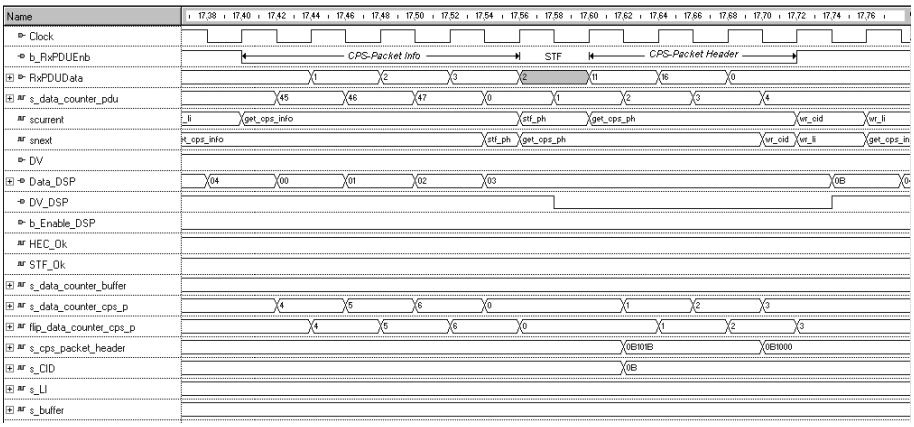


Figura 6.16: Recebimento do final de uma PDU junto com o final de um *CPS-Packet* sem erros.

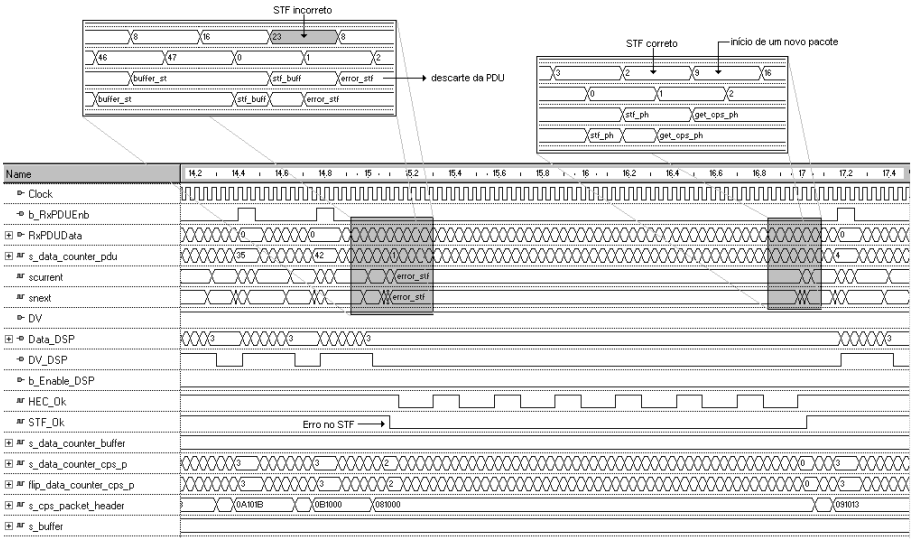


Figura 6.19: Recebimento de uma PDU com erro no `STF` que termina junto com o *CPS-Packet*.

Na Figura 6.19 observa-se a FSM no estado de descarte de PDU, `ERROR_STF`. No detalhe à esquerda é mostrado o início do descarte. Como ocorreu um processo de armazenamento temporário na PDU anterior, as informações recebidas deverão ser desconsideradas. À direita é mostrado, no detalhe, o final do descarte. O último pacote da PDU é totalmente descartado durante o estado de `ERROR_STF`, o que significa que a próxima PDU inicia com um pacote completo, e não com o final de um pacote. Assim, não há o que ser desconsiderado quando a próxima PDU for recebida, e por isso a máquina não passa pelo estado de `REMAINDER_ST`.

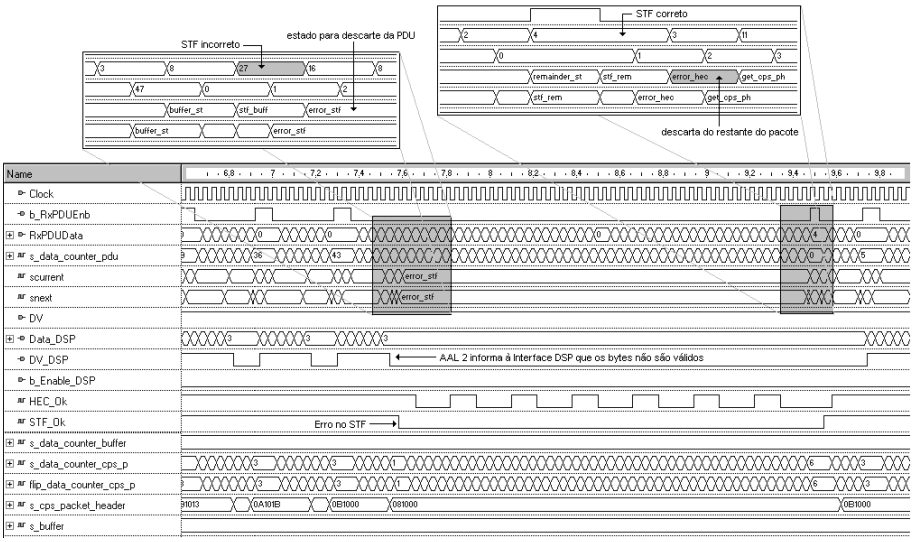


Figura 6.17: Recebimento de uma PDU com erro no `STF` quando a anterior termina exatamente entre os bytes do *CPS-Packet Header* e do *CPS-Packet Info*.

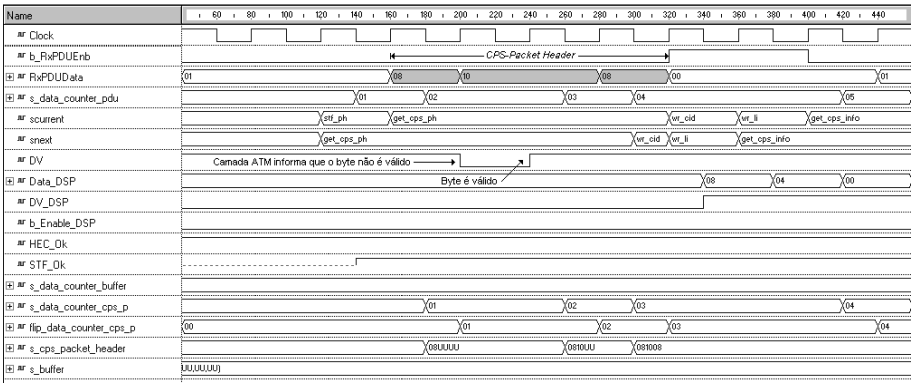


Figura 6.20: Pausa no recebimento de um *CPS-Packet Header* de um pacote contido em uma única PDU.

A Figura 6.20 mostra o instante em que a camada ATM informa à AAL 2 que não está pronta para enviar novos bytes, causando uma pausa no recebimento de informações. Esta figura apresenta uma pausa de um ciclo de *clock* durante o recebimento do segundo byte do cabeçalho do *CPS-Packet*. Esta situação causa uma parada na contagem de bytes recebidos por parte da AAL 2.

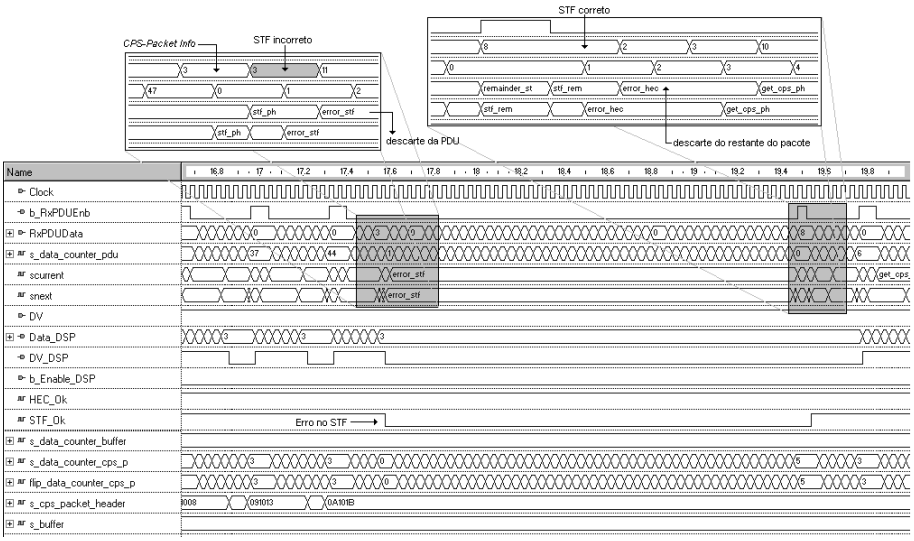


Figura 6.18: Recebimento de uma PDU com erro no `STF` quando a anterior termina junto com o *CPS-Packet*.

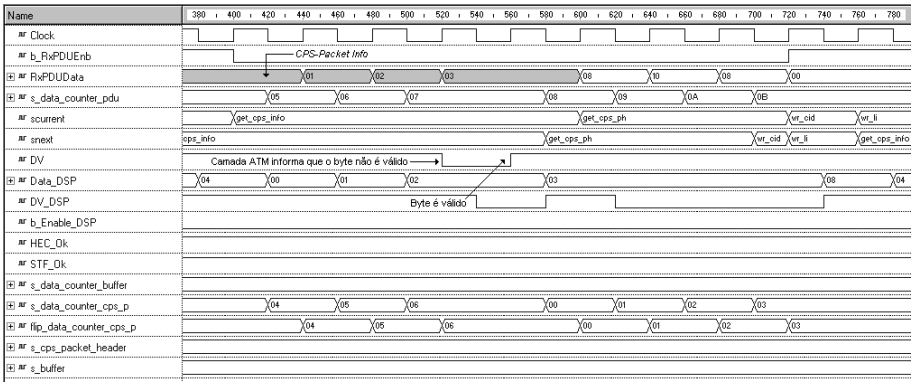


Figura 6.21: Pausa no recebimento de um *CPS-Packet Info* de um pacote contido em uma única PDU.

A camada ATM informa à AAL 2 que não possui bytes para enviar através do sinal `DV` (*data valid*), mostrado na Figura 6.21. Durante o recebimento do *CPS-Packet Info*, os bytes são repassados diretamente à Interface DSP. Quando ocorre a pausa, a AAL 2 deve informar à Interface que o byte contido no barramento não é válido, através do sinal `DV_DSP`.

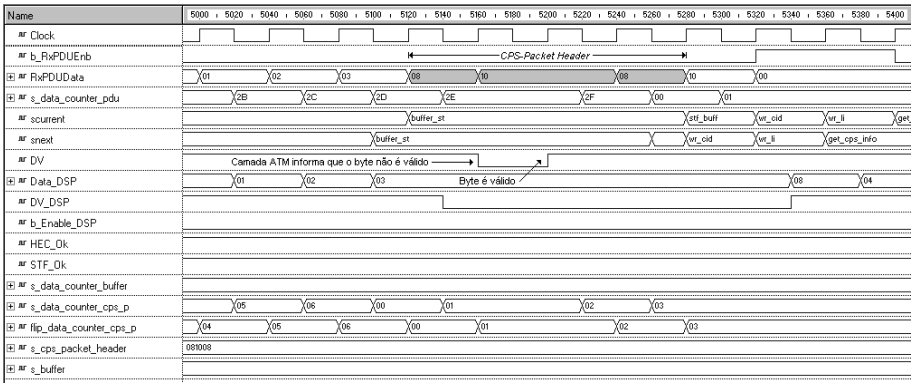


Figura 6.22: Pausa no recebimento de um *CPS-Packet Header* durante o armazenamento temporário.

A Figura 6.22 mostra o instante em que a camada ATM desabilita o sinal `DV`, indicando que o byte no barramento não é válido. Mesmo durante o armazenamento temporário, o *CPS-Packet Header* é armazernado em um sinal auxiliar, chamado *s_ cps_ packet_ header*, para ser oportunamente enviado à Interface DSP. Assim, essa situação causa apenas uma parada na contagem de bytes recebidos pela AAL 2.

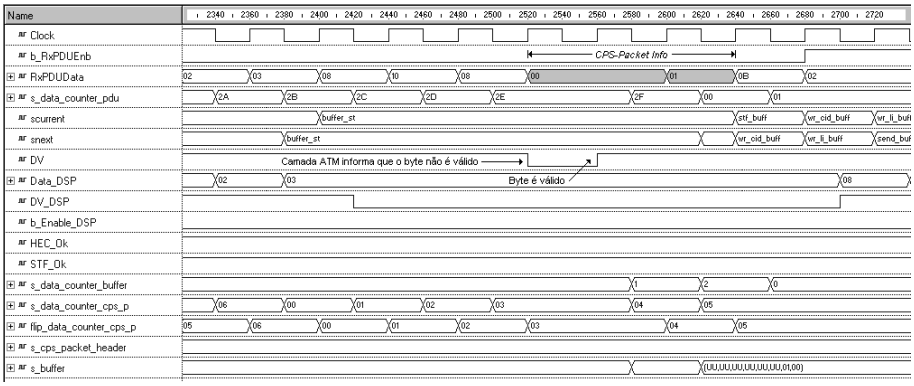


Figura 6.23: Pausa no recebimento de um *CPS-Packet Info* durante o armazenamento temporário.

A camada ATM informa à AAL 2 que os bytes contidos no barramento de dados não são válidos enquanto o sinal `DV` permanecer em zero. Neste caso, mostrado na Figura 6.23, os bytes recebidos estão sendo armazenados no *buffer* auxiliar para serem enviados à Interface DSP apenas se o `STF` e o `HEC` estiverem corretos. Dessa forma, uma pausa durante o armazenamento temporário implica não só em uma pausa na contagem de bytes recebidos, mas também em uma pausa na contagem dos bytes armazenados no *buffer*.

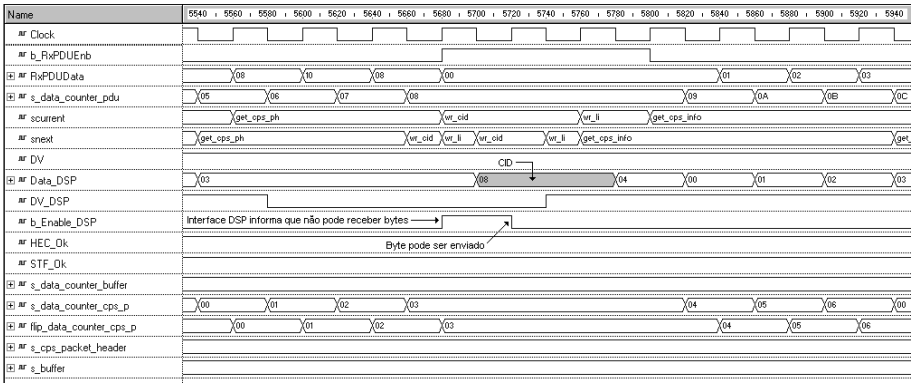


Figura 6.24: Pausa na escrita do `CID`.

A Interface DSP informa à AAL 2 que não pode receber novos bytes através do sinal `b_Enable_DSP`, como pode ser observado na Figura 6.24. Somente após a Interface DSP indicar a possibilidade de receber bytes novamente é que a AAL 2 pode habilitar o sinal `DV_DSP` e informar à Interface que o byte que encontra-se no barramento é válido. Essa parada ocorre no momento em que a AAL 2 deveria envia valor do `CID`, fazendo com que a máquina de estados permaneça em `WR_CID` até que a escrita seja possível.

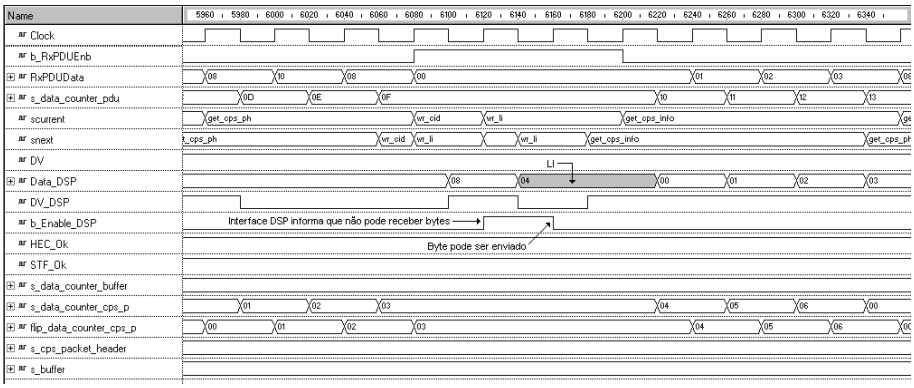


Figura 6.25: Pausa na escrita do `LI`.

Assim como na situação anterior, a Figura 6.25 mostra que a Interface DSP informa à AAL 2 que não pode receber novos bytes através do sinal `b_Enable_DSP`. Somente após a Interface DSP indicar a possibilidade de receber bytes novamente é que a AAL 2 pode habilitar o sinal `DV_DSP` e informar à Interface que o byte que encontra-se no barramento é válido. Nesta figura a parada ocorre no momento em que a AAL 2 deveria enviar valor do `LI`, fazendo com que a máquina de estado permaneça em `WR_LI` até que a escrita seja possível.

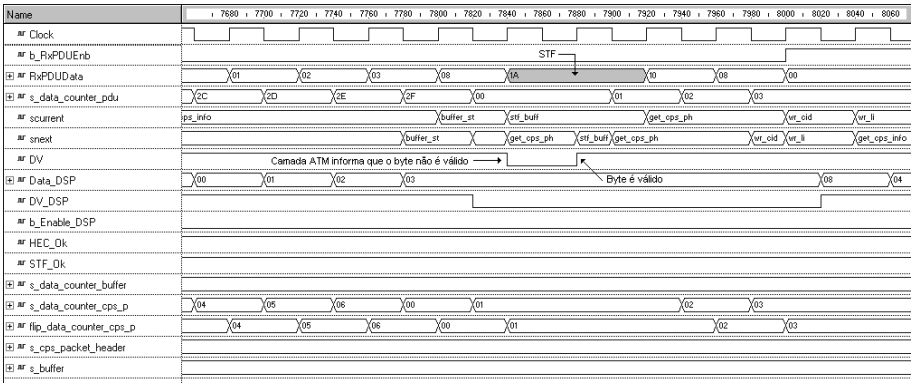


Figura 6.26: Pausa no recebimento do `STF`.

A Figura 6.26 mostra a camada ATM informando à AAL 2 que o byte presente no barramento não é válido através do sinal `DV`. Essa parada ocorre durante o recebimento do `STF`, e faz com que a máquina de estados permaneça por mais de um ciclo de *clock* no estado de recepção do `STF`.

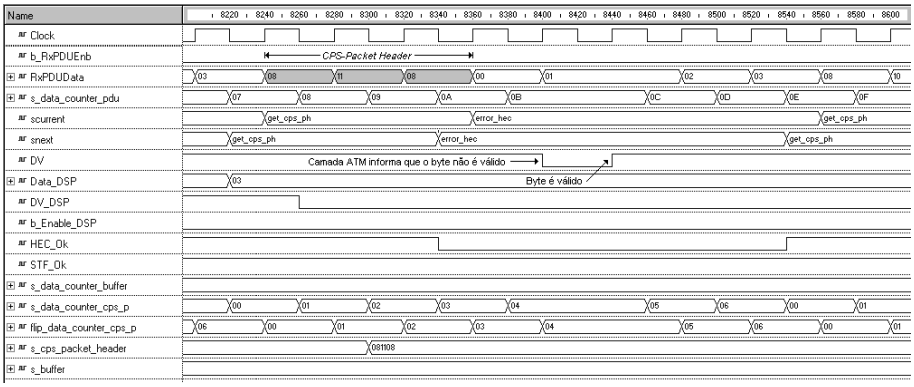


Figura 6.27: Pausa no descarte de pacote.

A Figura 6.27 mostra o que ocorre durante o descarte de apenas um *CPS-Packet*. Contudo o comportamento da máquina é exatamente o mesmo se o estado for para o descarte de uma PDU inteira. A pausa no recebimento dos bytes durante o estado de descarte reflete-se apenas em uma parada no contador de bytes recebidos pela AAL 2.

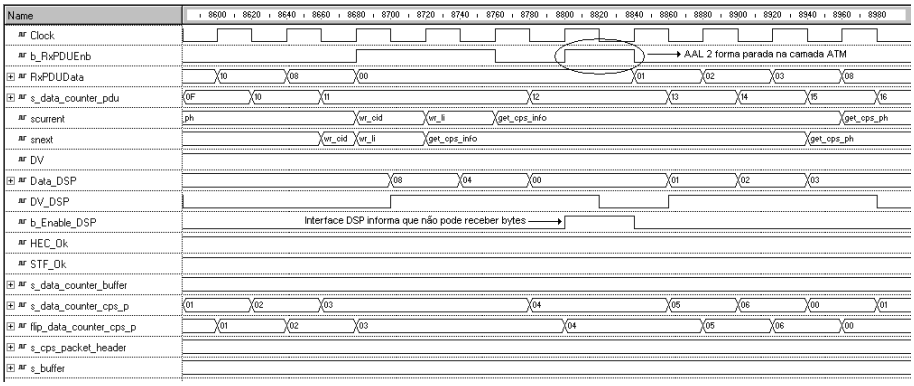


Figura 6.28: Pausa na escrita da informação.

Caso ocorra uma pausa na escrita da informação, esta parada pode ser causada pela Interface DSP, que informa não estar pronta para receber novos bytes através do sinal `b_Enable_DSP`, conforme a Figura 6.28. Somente após a Interface DSP indicar a possibilidade de receber bytes novamente é que a AAL 2 pode habilitar o sinal `DV_DSP` e informar à Interface que o byte que encontra-se no barramento é válido. Nesta figura a parada ocorre no momento em que a AAL 2 deveria repassar os bytes recebidos da camada ATM, que correspondem ao *CPS-Packet Info*.

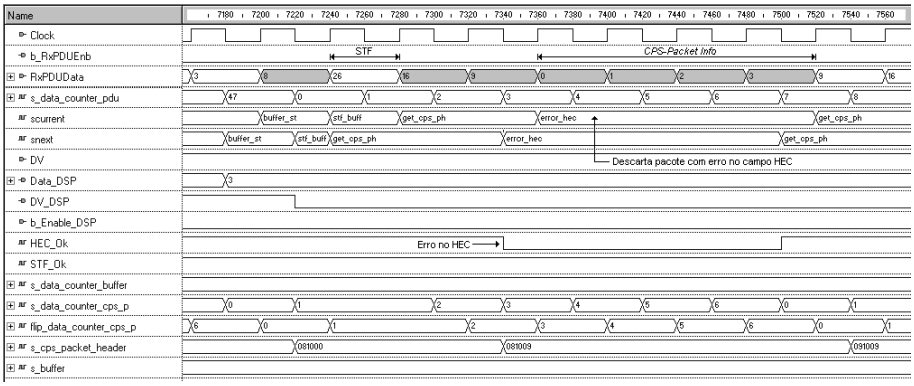


Figura 6.29: Recebimento do final de uma PDU durante os bytes do *CPS-Packet Header* com erro no `HEC`.

Quando a PDU termina, ainda não foram recebidos todos os bytes do cabeçalho para determinar se o `HEC` está correto ou não. Após o texttttSTF, o restante do `CPS-PH` é recebido e o `HEC` verificado. No caso de erro, como mostrado na Figura 6.29, a máquina passa para o estado de descarte de pacote.

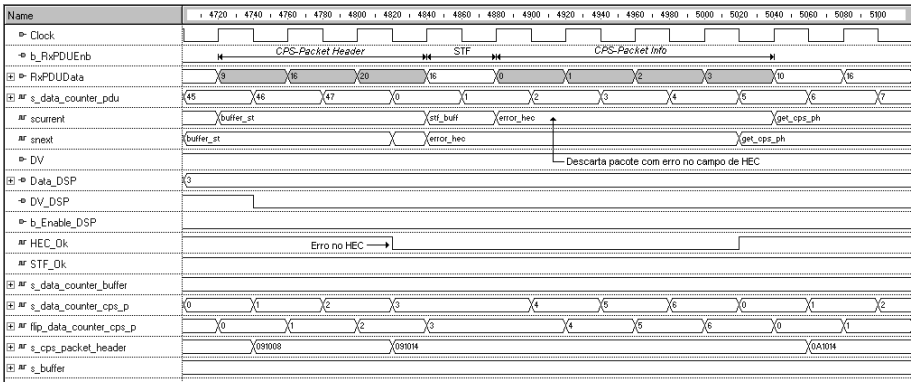


Figura 6.30: Recebimento do final de uma PDU exatamente entre os bytes do *CPS-Packet Header* e do*CPS-Packet Info* com erro no `HEC`.

A Figura 6.30 mostra um caso em que, ao final da PDU, já foram recebidos todos os bytes que compõem o *CPS-Packet Header*. Dessa forma, após constatação de erro e o recebimento do `STF`, a máquina passa diretamente para o estado de descarte de pacote.

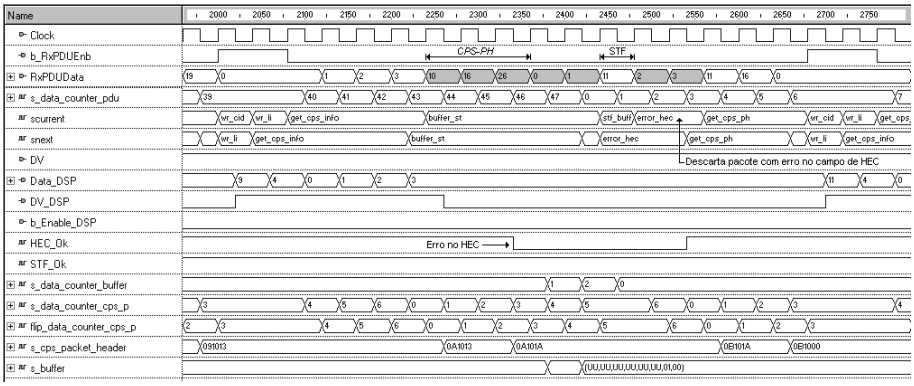


Figura 6.31: Recebimento do final de uma PDU durante os bytes do *CPS-Packet Info* com erro no `HEC`.

A Figura 6.31 mostra um caso em que está ocorrendo armazenamento temporário, pois o último pacote não está completo na PDU. Neste caso, os bytes de cabeçalho e informação são recebidos e armazenados no *buffer* (se forem de informação útil), até que a PDU termine. Após o recebimento do `STF`, como já foi sinalizado erro no `HEC` através do sinal `HEC_Ok`, a máquina passa para o estado de descarte de pacote e desconsidera o que está no *buffer* auxiliar.

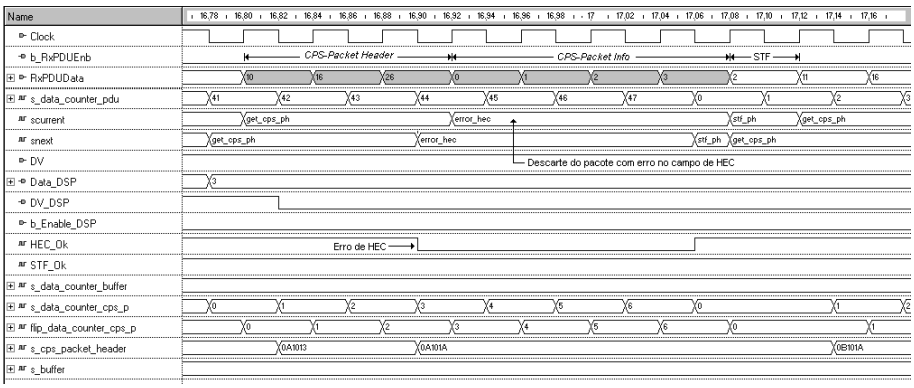


Figura 6.32: Recebimento do final de uma PDU junto com o final do *CPS-Packet* com erro no `HEC`.

A situação mostrada na Figura 6.32 é semelhante à Figura 6.11, pois o pacote encontra-se todo na mesma PDU, embora seja o último.

6.2 Avaliação dos Módulos de *Hardware* da AAL 2

O desempenho dos módulos de *hardware* desenvolvidos para a AAL 2 foi avaliado com relação ao número de ciclos de *clock* necessários para cada tarefa realizada. Essas avaliações foram feitas observando-se as formas de onda obtidas nas simulações.

Na AAL 2, um byte é recebido ou enviado a cada ciclo de *clock*. A recepção do cabeçalho de um *CPS-Packet* é feita em 3 ciclos, enquanto os bytes de informação precisam de 4 ciclos. Neste caso, considera-se um *CPS-Packet* com 3 bytes de cabeçalho e 4 bytes de informação. Como pode ser observado na Figura 6.13, após ter recebido o cabeçalho, a AAL 2 encaminha o `CID` e o `LI` à Interface DSP, causando uma parada na camada ATM. Assim, o tempo total gasto no recebimento de um *CPS-Packet* (e conseqüentemente no envio do mesmo à Interface DSP) é de 9 ciclos de *clock*, onde três são para o recebimento do cabeçalho, um para o envio do `CID`, um para o envio do `LI`, e os quatro restantes para o recebimento dos bytes de informação, vindos da camada ATM, e envio dos mesmos à Interface DSP, simultaneamente.

Para a recepção de uma PDU completa, o número de ciclos de *clock* varia, dependendo da maneira como os pacotes foram distribuídos nas PDUs. A duração fica entre 61 e 66 ciclos de *clock*. A Figura 6.33 mostra as formas como os pacotes podem estar dispostos em uma PDU e a influência que isso tem sobre o número de ciclos de *clock* consumidos na recepção de uma PDU completa. Estão sendo considerados os ciclos gastos com o descarte do parâmetro e o com recebimento da PDU propriamente dita.

Enquanto a AAL 2 recebe o cabeçalho e realiza as verificações necessárias quanto a presença de erros, a Interface DSP aguarda até que o barramento de dados contenha um byte válido. Essa espera, nos casos em que não ocorrem pausas por parte da camada ATM, é de 3 ciclos de *clock*, que é o tempo necessário para que a AAL 2 receba o cabeçalho do pacote. Esse tempo pode variar quando o pacote está dividido em duas PDUs, podendo chegar a 8 ciclos de *clock*. A Figura 6.34 mostra as situações em que os *CPS-Packets* ficam divididos e o atraso que isso causa na recepção dos mesmos pela Interface DSP.

Quando são identificados erros, no `STF` ou no `HEC`, a PDU ou o *CPS-Packet* precisam ser descartados. Se foi verificado um erro no campo de `STF`, a operação de descarte da PDU completa tem a duração mínima de 47 ciclos, além dos 2 ciclos já consumidos no descarte do

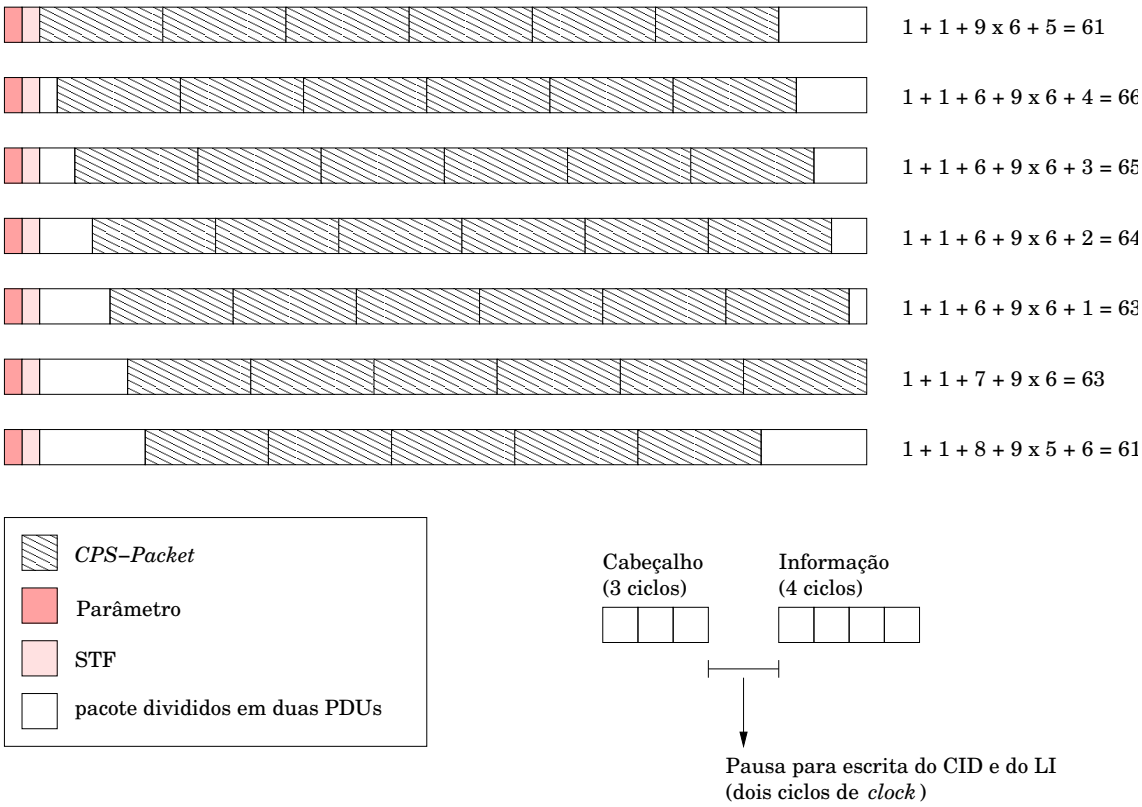


Figura 6.33: Número de ciclos de *clock* para o recebimento de uma PDU completa. De acordo com a disposição dos pacotes dentro da PDU, o número de ciclos de *clock* varia entre 61 e 66. Considera-se a necessidade de um ciclo para o descarte do parâmetro e um outro para o recebimento do `STF`, em cada PDU.

parâmetro e na recepção do `STF`. No caso do último pacote estar dividido em duas PDUs, ainda haverá o descarte do final deste pacote, que consumirá de 1 a 6 ciclos, dependendo do que resta na PDU seguinte. Se foi identificado um erro no `HEC`, o *CPS-Packet* é descartado em 7 ciclos de *clock*, ou 9 se o pacote encontra-se dividido em duas PDUs.

Diante de todas essas observações, baseadas nas formas de onda obtidas nas simulações, é possível estimar o desempenho global do módulo de *hardware* para a recepção de pacotes de voz compactada através da AAL 2. Considerando uma transferência contínua, ou seja, sem pausas por parte da camada ATM, livre de erros e com o pior caso para quebra de pacotes, a taxa de transferência estimada em bits por ciclo para o módulo de recepção é dada pela expressão:

$$\frac{49 \text{ bytes}}{66 \text{ ciclos}} = \frac{392 \text{ bits}}{66 \text{ ciclos}} = 5,939393 bits/ciclo \tag{6.1}$$

Considerando um ciclo de clock de 40ns (25MHz), a taxa estimada em bits por segundo para o módulo de recepção é a seguinte:

$$\frac{5,939393 \text{ bits/ciclo}}{40 \text{ ns}} = 148,484848 Mbps \tag{6.2}$$

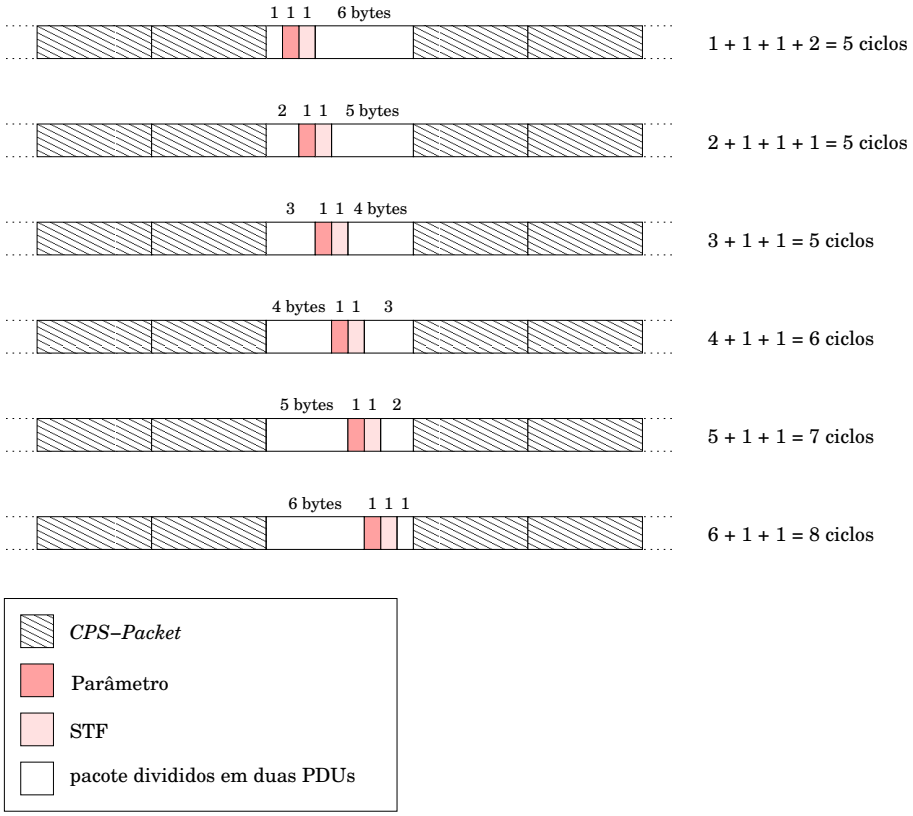


Figura 6.34: Número de ciclos de *clock* que a Interface DSP espera para receber um pacote. De acordo com a maneira que os *CPS-Packets* são divididos ao final de uma PDU, a Interface DSP pode ter que esperar de 5 a 8 ciclos de *clock* para iniciar a recepção de um novo pacote.

Capítulo 7

Implementação em *Software* da AAL 2

A implementação da AAL 2 para transferência de voz compactada foi direcionada para o uso de técnicas de projeto integrado de *hardware* e *software*. Para isso, desenvolveu-se um conjunto de funções em linguagem C que realizam as tarefas da AAL 2. A partir dessa implementação, juntamente com a implementação em *hardware*, procura-se estudar alguns compromissos de *hardware* e *software* para a AAL 2. A definição do particionamento em *hardware* e *software* é baseada em critérios como o desempenho (tempo de execução), o custo do hardware e a complexidade da implementação. Para estimar o tempo de execução desse conjunto de funções, pode-se utilizar uma técnica conhecida como *profiling*. Segundo [O'NI96], o objetivo dessa técnica é determinar o desempenho de cada função em termos de tempo de execução, medindo o tempo total de execução e verificando qual o percentual do tempo total que cada função consome. Funções que exigem mais tempo para execução possivelmente poderão obter melhor desempenho se implementadas em *hardware*. Para determinar o particionamento final em *hardware* e *software* isso ainda não é suficiente. É preciso considerar o custo de comunicação entre módulos de *hardware* e módulos de *software*. Se o *profiling* indica que uma determinada função deve ser implementada em *hardware* e outra deve ser em *software*, e existe um fluxo de transferência de informações significativo entre essas funções, provavelmente é mais interessante implementar ambas as funções em *hardware*, ou ambas em *software*. O ganho obtido com relação ao tempo de execução pode ser perdido na comunicação se as funções forem separadas.

7.1 Módulos de *Software* para a AAL 2

Para os módulos de transmissão e recepção de pacotes de voz foram criados dois programas distintos, cada um com as funções relativas à operação executada (transmissão ou recepção). Esses programas foram escritos em liguagem C e compilados para sistema operacional Linux. A divisão em módulos internos aos de recepção e transmissão foi feita de maneira semelhante à implementação em *hardware*, com o objetivo de tornar a avaliação dos módulos mais realista. Isso dispensa maiores comentários com relação aos módulos de *software*, uma vez que já foram discutidos os módulos de *hardware*.

7.1.1 Estratégia de Validação dos Módulos de *Software*

Para validar os dois módulos criados, de transmissão e de recepção, foi implementado um terceiro módulo, chamado `TX_Generate`, e foram utilizados arquivos no lugar dos *buffers* em

memória, que deverão estar presentes na implementação para a plataforma MicroBlaze.

O `TX_Generate` é uma adaptação do módulo `TX`. Ele gera valores válidos para os bytes que devem ser fornecidos pela Interface DSP, realiza as funções relativas à operação de transmissão, monta as AAL2-PDUs e grava-as em um primeiro arquivo, chamado de `buffer_atm.gen`, com informações que serão enviadas à camada ATM. Esse arquivo é então utilizado para validar o módulo de recepção. O `RX` lê as AAL2-PDUs desse arquivo, realiza as funções relativas à operação de recepção, extrai as informações que devem ser enviadas à Interface DSP e gera um segundo arquivo, chamado `buffer_dsp.aal`, com essas informações. Esse segundo arquivo é utilizado pelo módulo `TX` propriamente dito, que resgata as informações como se estivessem sendo entregues pela Interface DSP, trata-as, gera as AAL2-PDUs e grava um terceiro arquivo, chamado `buffer_atm.aal`. A validação foi obtida comparando-se o primeiro arquivo, gerado pelo `TX_Generate`, com o último, gerado pelo `TX`, e constatando-se que eram idênticos. A Figura 7.1 ilustra esse procedimento de validação.

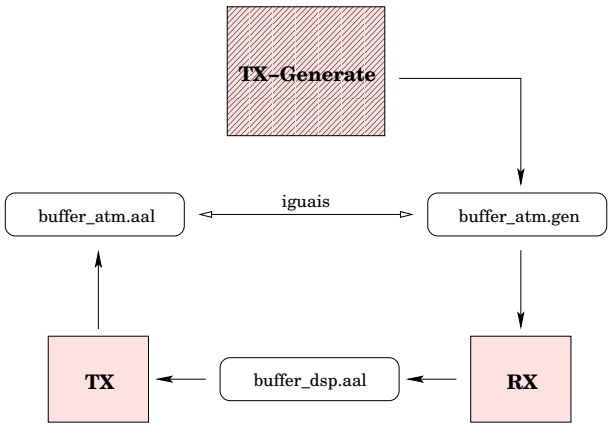


Figura 7.1: Estratégia de validação dos módulos de *software* da AAL 2.

7.2 Avaliação dos Módulos de *Software*

Os módulos da AAL 2 implementados em *software* foram avaliados utilizando a ferramenta `gprof`, do Linux, em um microcomputador com processador Intel Pentium MMX 200MHz. Os resultados obtidos dizem respeito ao tempo de processamento, avaliando separadamente os módulos de transmissão e recepção.

Para o módulo de transmissão, o programa em C lê um arquivo aproximadamente um milhão de pacotes de voz, simulando a Interface DSP, e repassa esses pacotes aos módulos da AAL 2 para serem encapsulados em AAL2-PDUs. A Tabela 7.1 apresenta os resultados da avaliação, comentados a seguir.

As funções `count_bits` e `init_buffer` são auxiliares para o processo de recebimento de pacotes da Interface DSP e envio dos mesmos pela camada ATM. A primeira realiza a contagem dos bits em 1 em um determinado byte, fazendo parte da operação de cálculo da paridade. A segunda apenas preenche com zeros o *buffer* que receberá os pacotes do arquivo.

As demais funções compõem os módulos para transmissão de pacotes de voz através da AAL 2. `rd_memory` e `send_PDU` controlam a comunicação com os *buffers* (o que contém pacotes de voz originados da Interface DSP, e o que contém PDUs a serem enviadas para a camada ATM, respectivamente). Como já comentado anteriormente, a AAL 2 recebe da Interface DSP

Tabela 7.1: Avaliação do módulo de transmissão da AAL 2 para um milhão de pacotes. A função principal (*main*) não faz parte da avaliação

t%	**ta**(s)	**tf**(s)	**nc**	**tcf**(ns)	**ttc**(ns)	**Nome da função**
55.38	2.42	2.42	999999	2420.00	2420.00	compute_HEC
16.48	3.14	0.72	999999	720.00	1490.00	create_AAL2_PDU
9.84	3.57	0.43	297874	1443.56	1443.56	count_bits
6.64	3.86	0.29	148937	1947.13	1947.13	init_buffer
5.26	4.09	0.23	999999	230.00	2730.00	create_CPS_Packet
3.43	4.24	0.15				main
1.83	4.32	0.08	999999	80.00	80.00	rd_memory
0.92	4.36	0.04	148937	268.57	3155.70	generate_odd_parity
0.23	4.37	0.01	148936	67.14	67.14	send_PDU

t% = tempo percentual
ta = tempo acumulado (s)
tf = tempo da função (s)
nc = número de chamadas
tcf = tempo por chamada da função (ns)
ttc = tempo total por chamada (incluindo descendentes) (ns)

6 bytes, sendo 4 de informação e outros 2 contendo o `CID` e o tamanho do pacote. A primeira etapa do processamento da AAL 2 deve utilizar esses 6 bytes para gerar um *CPS-Packet*. Isso é feito pela função `create_CPS_Packet`, que utiliza ainda a função `compute_HEC` para o cálculo do código de verificação de erros (HEC). Por fim, a função `create_AAL2_PDU` recebe diversos pacotes de voz e monta uma AAL2-PDU, que será armazenada no *buffer* de comunicação com a camada ATM (na função `send_PDU`, conforme já mencionado).

Assim, a Tabela 7.1 mostra o resultado da execução do módulo de transmissão, avaliando separadamente as funções do módulo e mostrando o percentual do tempo de execução total que corresponde a cada função do módulo de transmissão e o tempo, em segundos, consumido pelas mesmas.

De acordo com os resultados obtidos, é possível observar que a função `compute_HEC` é a que consome a maior tempo, consumindo cerca de $2420ns$ em cada chamada. Essa função é chamada sempre que um novo *CPS-Packet* é criado (pela função `create_CPS_Packet`). Dessa forma, com aproximadamente um milhão de pacotes, é nessa função que se concentra o maior consumo de tempo de execução: 55% do tempo total. Observando-se também a função `create_CPS_Packet`, nota-se que o consumo de tempo dessa função, isoladamente é o menor, desconsiderando as funções que não são propriamente do módulo de transmissão. A função `create_CPS_Packet` consome $230ns$. Mas tendo uma chamada a função `compute_HEC` para cada pacote a ser criado, o tempo consumido pela função e sua descendente é bem superior.

A Tabela 7.2 descreve a árvore de chamadas do programa, ordenada pelo tempo total consumido por cada função e suas descendentes. Cada entrada nessa tabela consiste de diversas linhas. A linha que possui um índice à esquerda mostra a função corrente, em destaque. As linhas acima mostram as funções que chamaram a primeira (funções ancestrais), enquanto que as linhas abaixo mostram as funções chamadas por ela (funções descendentes).

Tabela 7.2: Árvore de chamadas do módulo de transmissão da AAL 2 para um milhão de pacotes.

Índice	**t%**	**tf**(s)	**td**(s)	**Chamadas**	**Nome da função**
[1]	100.0	0.15	4.22		*main* [1]
		0.23	2.50	999999/999999	create_CPS_Packet [2]
		0.72	0.77	999999/999999	create_AAL2_PDU [4]
		0.23	2.50	999999/999999	main [1]
[2]	62.5	0.23	2.50	999999	*create_CPS_Packet* [2]
		2.42	0.00	999999/999999	compute_HEC [3]
		0.08	0.00	999999/999999	rd_memory [8]
		2.42	0.00	999999/999999	create_CPS_Packet [2]
[3]	55.4	2.42	0.00	999999	*compute_HEC* [3]
		0.72	0.77	999999/999999	main [1]
[4]	34.1	0.72	0.77	999999	*create_AAL2_PDU* [4]
		0.04	0.43	148937/148937	generate_odd_parity [5]
		0.29	0.00	148937/148937	init_buffer [7]
		0.01	0.00	148936/148936	send_PDU [9]
		0.04	0.43	148937/148937	create_AAL2_PDU [4]
[5]	10.8	0.04	0.43	148937	*generate_odd_parity* [5]
		0.43	0.00	297874/297874	count_bits [6]
		0.43	0.00	297874/297874	generate_odd_parity [5]
[6]	9.8	0.43	0.00	297874	*count_bits* [6]
		0.29	0.00	148937/148937	create_AAL2_PDU [4]
[7]	6.6	0.29	0.00	148937	*init_buffer* [7]
		0.08	0.00	999999/999999	create_CPS_Packet [2]
[8]	1.8	0.08	0.00	999999	*rd_memory* [8]
		0.01	0.00	148936/148936	create_AAL2_PDU [4]
[9]	0.2	0.01	0.00	148936	*send_PDU* [9]

t% = tempo percentual
tf = tempo da função (s)
td = tempo das descendentes (s)

Dessa forma é possível calcular a taxa de transmissão obtida com a implementação em *software* da AAL 2. Uma vez que foram transmitidos quase um milhão de pacotes em 4.37 segundos, e cada pacote possui 6 bytes, a taxa, em Mbps é dada pela equação 7.1:

$$taxa = \frac{999999 * 6 * 8}{4.37} = \frac{47999952}{4.37} = 10.98397071 Mbps \quad (7.1)$$

Para o módulo de recepção, o programa em C lê de um arquivo as AAL2-PDUs, simulando um envio pela camada ATM. A seguir, extrai pacotes de voz e repassa os mesmos à Interface DSP. A Tabela 7.3 apresenta os resultados da avaliação. As funções `compute_HEC`, `count_bits` e `generate_odd_parity` são idênticas às anteriores, utilizadas no módulo de transmissão. A função `rd_memory` simula agora a camada ATM, retirando do arquivo as PDUs enviadas. Cada PDU recebida tem seu campo de `OSF` verificado com relação à presença de erros, tarefa esta que cabe à função `check_OSF`. Por outro lado, `recover_CPS_Packet` é a função responsável por extrair da AAL2-PDU os *CPS-Packets* e destes as informações que serão repassadas à

Interface DSP.

De acordo com a Tabela 7.3, a função `recover_CPS_Packet` é a que consome o maior tempo de execução, como era esperado, pois trata-se da função que realiza a maior parte do processo de recepção de pacotes pela AAL 2. Essa função é chamada a cada nova PDU recebida (o que significa quase 150000 vezes) e extrai delas aproximadamente 1 milhão de *CPS-Packets*.

A Tabela 7.4 descreve a árvore de chamadas do programa que realiza a recepção de PDUs, ordenada pelo tempo total consumido por cada função e suas descendentes, mostrando também as funções ancestrais e as funções descendentes.

A taxa de recepção calculada para o módulo de *software* é dada pela equação 7.2.

$$taxa = \frac{999998 * 6 * 8}{4.39} = \frac{47999904}{4.39} = 10.93391891 Mbps \tag{7.2}$$

Tabela 7.3: Avaliação do módulo de recepção da AAL 2.

t%	**ta**(s)	**tf**(s)	**nc**	**tcf**(ns)	**ttc**(ns)	**Nome da função**
74.72	3.28	3.28	999998	3280.01	3280.01	compute_HEC
16.86	4.02	0.74	148936	4968.58	29408.61	recover_CPS_Packet
7.06	4.33	0.31	297872	1040.72	1040.72	count_bits
0.91	4.37	0.04	148936	268.57	2350.00	generate_odd_parity
0.23	4.38	0.01	148937	67.14	67.14	rd_memory
0.23	4.39	0.01	148936	67.14	2417.15	check_OSF

t% = tempo percentual
ta = tempo acumulado (s)
tf = tempo da função (s)
nc = número de chamadas
tcf = tempo por chamada da função (ns)
ttc = tempo total por chamada (ns)

Tabela 7.4: Árvore de chamadas do módulo de recepção da AAL 2.

Índice	**t%**	**tf**(s)	**td**(s)	**Chamadas**	**Nome da função**
[1]	100.0	0.00	4.39		*main* [1]
		0.74	3.64	148936/148936	recover_CPS_Packet [2]
		0.01	0.00	148937/148937	rd_memory [7]
		0.74	3.64	148936/148936	main [1]
[2]	99.8	0.74	3.64	148936	*recover_ CPS_ Packet* [2]
		3.28	0.00	999998/999998	compute_HEC [3]
		0.01	0.35	148936/148936	check_OSF [4]
		3.28	0.00	999998/999998	recover_CPS_Packet [2]
[3]	74.7	3.28	0.00	999998	*compute_ HEC* [3]
		0.01	0.35	148936/148936	recover_CPS_Packet [2]
[4]	8.2	0.01	0.35	148936	*check_ OSF* [4]
		0.04	0.31	148936/148936	generate_odd_parity [5]
		0.04	0.31	148936/148936	check_OSF [4]
[5]	8.0	0.04	0.31	148936	*generate_ odd_ parity* [5]
		0.31	0.00	297872/297872	count_bits [6]
		0.31	0.00	297872/297872	generate_odd_parity [5]
[6]	7.1	0.31	0.00	297872	*count_ bits* [6]
		0.01	0.00	148937/148937	main [1]
[7]	0.2	0.01	0.00	148937	*rd_ memory* [7]

t% = tempo percentual
tf = tempo da função (s)
td = tempo das descendentes (s)

7.2.1 Conclusões

As taxas de transmissão e recepção foram calculadas com base nos pacotes trocados com a Interface DSP. Na transmissão, foram observados os pacotes recebidos da Interface DSP, ou seja, aproximadamente um milhão de pacotes contendo 6 bytes, um para o `CID`, um para o `LI` e quatro de informação. Na recepção, foram observados os pacotes enviados para a Interface DSP, contendo os mesmos 6 bytes, que foram extraídos dos *CPS-Packets*.

Os resultados obtidos ficaram muito abaixo daqueles alcançados com os módulos de *hardware*, como já era esperado. Ainda assim, 11 Mbps para transferência de voz é muito satisfatório, considerando que a taxa de transmissão dos canais de voz é de 32 kbps, e que pretende-se multiplexar apenas 4 canais. Com a taxa obtida no módulo de *software*, é possível utilizar mais de 340 canais de voz operando a 32 kbps.

Encontra-se em desenvolvimento uma segunda implementação em *software*, que executa sobre um processador embarcado em um FPGA. Apenas o módulo de recepção de voz pela AAL 2 está sendo utilizado e a avaliação desse módulo será feita através de um *timer*, disponível na plataforma.

7.3 Determinação do *Timer_ CU*

O *Timer_ CU* (*Timer Combined Use*) é um parâmetro importante na operação de transmissão através de AAL 2. Ele indica quanto tempo a AAL 2 deverá esperar por um novo pacote de voz antes de enviar a PDU à camada ATM. Se esse parâmetro for dimensionado de maneira incorreta, o desempenho da implementação da AAL 2 pode ser significativamente prejudicado [LIU99]. Com um valor de *Timer_ CU* pequeno, a AAL 2 espera pouco tempo antes de enviar sua PDU à camada ATM, fazendo com que as PDUs sejam enviadas incompletas, o que pode prejudicar a densidade de empacotamento da AAL 2. Por outro lado, se o valor do *Timer_ CU* for grande, a AAL 2 pode esperar demais pela chegada de um pacote, fazendo com que as PDUs sejam enviadas com atraso.

Para a determinação de um valor adequado para o *Timer_ CU*, Liu, Munir e Jain propuseram em [LIU99] um algoritmo e, através de simulações e deduções matemáticas, chegaram a um valor para o *Timer_ CU* em torno de 14ms para uma aplicação específica. Segundo [LIU99], a voz humana consiste em períodos de silêncio e períodos de fala (*talkspurt*), e um modelo aceitável de transmissão considera os períodos de fala com duração de 352ms, enquanto que os períodos de silêncio possuem duração de 650ms. O padrão de chegada de pacotes considerado no artigo é de 12 pacotes por período de fala, os mesmos possuindo 20 bytes de informação útil, com mais 3 de cabeçalho, totalizando 23 bytes de tamanho, de acordo com a Recomendação G.723.1 do ITU-T [UNI96a]. Os autores definem uma Cadeia de Markov para analisar o processo de empacotamento da AAL 2, e determinam o valor do *Timer_ CU* através de deduções matemáticas, estas sendo comprovadas pela simulação realizada. Eles consideram ainda a restrição de que uma mesma fonte não gera dois pacotes dentro do mesmo período de *Timer_ CU*.

O algoritmo proposto em [LIU99] foi utilizado como base em [LU01] para a determinação de um valor de *Timer_ CU* para mensagens de sinalização da AAL 2. Os autores analisam como as mensagens de sinalização afetam a densidade de empacotamento e mostram que um valor adequado de *Timer_ CU* para sinalização reduz o atraso de estabelecimento de chamadas sem comprometer a densidade de empacotamento.

Para a aplicação apresentada no Capítulo 4 também será utilizado o algoritmo de [LIU99] para a estimativa de um valor adequado para o *Timer_ CU*. A primeira consideração a ser feita diz respeito ao tamanho do pacote, que é 7 bytes (3 de cabeçalho e 4 de informação útil). Dessa forma, é possível ter de 5 a 6 pacotes inteiros de voz compactada em uma mesma PDU, considerando-a completamente preenchida.

De acordo com [LIU99], uma fonte gera amostras a cada 30ms, enviando 12 pacotes a cada período de fala de 352ms. No caso do *Driver* ATM, o DSP gera amostras de voz a cada 1ms. Dessa forma, tem-se 352, e não 12 pacotes a cada período de fala. Assim, é possível determinar a probabilidade de que nenhum pacote chegue nos próximos τms, onde τ é o valor do *Timer_ CU*. Essa probabilidade é dada pela Equação 7.3.

$$p = \frac{352 + 650 - 352\tau}{352 + 650} = \frac{1002 - 352\tau}{1002}, \text{ onde } \tau < 1ms \tag{7.3}$$

Para várias fontes, é interessante determinar a probabilidade de nenhum pacote chegar após receber o primeiro. Isto leva a calcular a probabilidade de nenhum pacote de voz chegar após iniciar a contagem do *Timer_ CU*. Essa probabilidade, chamada de R_0, é dada pela Equação 7.4, onde N é o número de fontes.

$$R_0 = p^{N-1} \tag{7.4}$$

Da mesma forma, R_1 a R_5 representam as probabilidades de que cheguem de 1 a 5 pacotes após receber o primeiro, respectivamente. Essas probabilidades são calculadas usando a Equação 7.5

$$R_i = (N-1)p^{N-i+1}(1-p)^i \text{ onde i = 1, 2, ..., 5} \tag{7.5}$$

Utilizando as probabilidades R_0 a R_5 é possível determinar a densidade de empacotamento média. O número médio de bytes em uma PDU depende da quantidade de bytes que restaram do último pacote e foram colocados na PDU seguinte, chamados de r_n (resto na célula n). Como o tamanho do pacote é de 7 bytes, existem 7 casos possíveis para início de uma PDU:

$r_n = 0$: Ocorre quando o *Timer_ CU* para a PDU anterior expira ou quando $r_{n-1} = 5$ e o *timer* não expira.

$r_n = 1$: Ocorre quando $r_{n-1} = 6$ e o *timer* não expira.

$r_n = 2$: Ocorre quando $r_{n-1} = 0$ e o *timer* não expira.

$r_n = 3$: Ocorre quando $r_{n-1} = 1$ e o *timer* não expira.

$r_n = 4$: Ocorre quando $r_{n-1} = 2$ e o *timer* não expira.

$r_n = 5$: Ocorre quando $r_{n-1} = 3$ e o *timer* não expira.

$r_n = 6$: Ocorre quando $r_{n-1} = 4$ e o *timer* não expira.

Portanto, r_n forma uma Cadeia de Markov. Considerando o estado estacionário, onde todos os r_n possuem a mesma probabilidade, chamada π, tem-se as seguintes equações:

$$\pi_i = P\{r_n = i\}, \text{ onde i = 1, 2, ..., 6} \tag{7.6}$$

$$Q_i = P\{\text{Timer\_CU expirar}|r_n = i\}, \text{ onde i = 1, 2, ..., 6} \tag{7.7}$$

A equação 7.6 define π_i como sendo a probabilidade de que r_n seja igual a i. Da mesma forma, a equação 7.7 define Q_i como sendo a probabilidade de que o *Timer_ CU* expire quando $r_n = i$. Considerando que o *Timer_ CU* expire quando $r_n = 0$, Q_0 é então a probabilidade de chegarem no máximo 6 pacotes em τ ms. Assim, pode-se observar o seguinte:

Q_0: É a probabilidade de chegarem no máximo 6 pacotes em τms quando $r_n = 0$.

Q_1: É a probabilidade de chegarem no máximo 6 pacotes em τms quando $r_n = 1$.

Q_2: É a probabilidade de chegarem no máximo 6 pacotes em τms quando $r_n = 2$.

Q_3: É a probabilidade de chegarem no máximo 6 pacotes em τms quando $r_n = 3$.

Q_4: É a probabilidade de chegarem no máximo 6 pacotes em τms quando $r_n = 4$.

Q_5: É a probabilidade de chegarem no máximo 6 pacotes em τms quando $r_n = 5$.

Q_6: É a probabilidade de chegarem no máximo 5 pacotes em τms quando $r_n = 6$.

Simplificando, pode-se dizer que Q é a probabilidade de chegarem no máximo 6 pacotes em τ ms, e S é a probabilidade de chegarem no máximo 5 pacotes nesse mesmo tempo, onde.

$$Q = R_0 + R_1 + R_2 + R_3 + R_4 + R_5 \tag{7.8}$$

$$S = R_0 + R_1 + R_2 + R_3 + R_4 \tag{7.9}$$

Com isso, a matriz de transição P é dada a seguir:

$$P = \begin{pmatrix} Q & 0 & 1-Q & 0 & 0 & 0 & 0 \\ Q & 0 & 0 & 1-Q & 0 & 0 & 0 \\ Q & 0 & 0 & 0 & 1-Q & 0 & 0 \\ Q & 0 & 0 & 0 & 0 & 1-Q & 0 \\ Q & 0 & 0 & 0 & 0 & 0 & 1-Q \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ S & 1-S & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{7.10}$$

Pode-se, então, determinar os valores para π, como mostrado nas Equações 7.11 a 7.17, abaixo:

$$\pi_0 = Q + \pi_5 * (1-Q) + \pi_6 * (S-Q) \tag{7.11}$$

$$\pi_1 = \pi_6 * (1-S) \tag{7.12}$$

$$\pi_2 = \pi_0 * (1-Q) \tag{7.13}$$

$$\pi_3 = \pi_1 * (1-Q) \tag{7.14}$$

$$\pi_4 = \pi_2 * (1-Q) \tag{7.15}$$

$$\pi_5 = \pi_3 * (1-Q) \tag{7.16}$$

$$\pi_6 = \pi_4 * (1-Q) \tag{7.17}$$

Assim, em função de π_0 tem-se as Equações 7.18 a 7.24:

$$\pi_0 = \frac{Q}{1-(1-Q)^6 * (1-S) - (1-Q)^3 * (S-Q)} \tag{7.18}$$

$$\pi_1 = \pi_0 * (1-Q)^3 * (1-S) \tag{7.19}$$

$$\pi_2 = \pi_0 * (1-Q) \tag{7.20}$$

$$\pi_3 = \pi_0 * (1-Q)^4 * (1-S) \tag{7.21}$$

$$\pi_4 = \pi_0 * (1-Q)^2 \tag{7.22}$$

$$\pi_5 = \pi_0 * (1-Q)^5 * (1-S) \tag{7.23}$$

$$\pi_6 = \pi_0 * (1-Q)^3 \tag{7.24}$$

Usando essas probabilidades, o número médio de bytes em uma PDU da AAL 2 é dado pela Equação 7.25:

$$\begin{aligned} C = \pi_0 * (7*R_0 + 14*R_1 + 21*R_2 + 28*R_3 + 35*R_4 + 42*R_5 + 47*(1-Q)) \\ + \sum_{i=1}^{5} \pi_i * (i*R_0 + i*R_1 + i*R_2 + i*R_3 + i*R_4 + i*R_5) \\ + \sum_{i=1}^{5} \pi_i * (7*R_1 + 14*R_2 + 21*R_3 + 28*R_4 + 35*R_5 + 47*(1-Q)) \\ + \pi_6 * (6*R_0 + 13*R_1 + 20*R_2 + 27*R_3 + 34*R_4 + 47*(1-S)) \end{aligned} \tag{7.25}$$

Com esse valor é possível determinar a densidade de empacotamento média para as PDUs da AAL 2, como mostra a Equação 7.26.

$$D = \frac{C}{53} * \frac{4}{7} * 100\% \tag{7.26}$$

A densidade de empacotamento máxima é, então:

$$D = \frac{47}{53} * \frac{4}{7} * 100\% = 50,673854\% \tag{7.27}$$

Com as equações, foi elaborado um programa que estima valores para o $Timer\_CU$ e para o número de fontes de voz, e busca a carga média e a densidade de empacotamento média. Diversos experimentos foram realizados e os resultados e conclusões são apresentados a seguir.

Os valores estimados para o $Timer\_CU$ ficam entre 0.01ms e 1ms ($0.01ms \leq \tau < 1ms$), com intervalos de aproximadamente 0.01ms. Supondo 15 fontes de voz, obtém-se a Tabela 7.5 com os resultados para carga e densidade médias, e os gráficos da Figura 7.2.

Tabela 7.5: Determinação de carga e densidade de empacotamento para 15 fontes de voz. Considerando que a carga (C) máxima é 47 bytes e a densidade de empacotamento (D) máxima é 50,673854%.

Timer_ CU(ms)	**p**	**C**(bytes)	**D**(%)
0.010000	0.996487	7.373117	7.949452
0.100000	0.964870	12.587153	13.571054
0.200000	0.929741	20.168106	21.744589
0.300000	0.894611	27.576857	29.732460
0.400000	0.859481	33.754506	36.392998
0.500000	0.824351	38.411048	41.413529
0.600000	0.789222	41.671087	44.928395
0.700000	0.754092	43.824097	47.249700
0.800000	0.718962	45.178109	48.709552
0.900000	0.683832	45.993245	49.588404
0.990000	0.652216	46.428353	50.057523

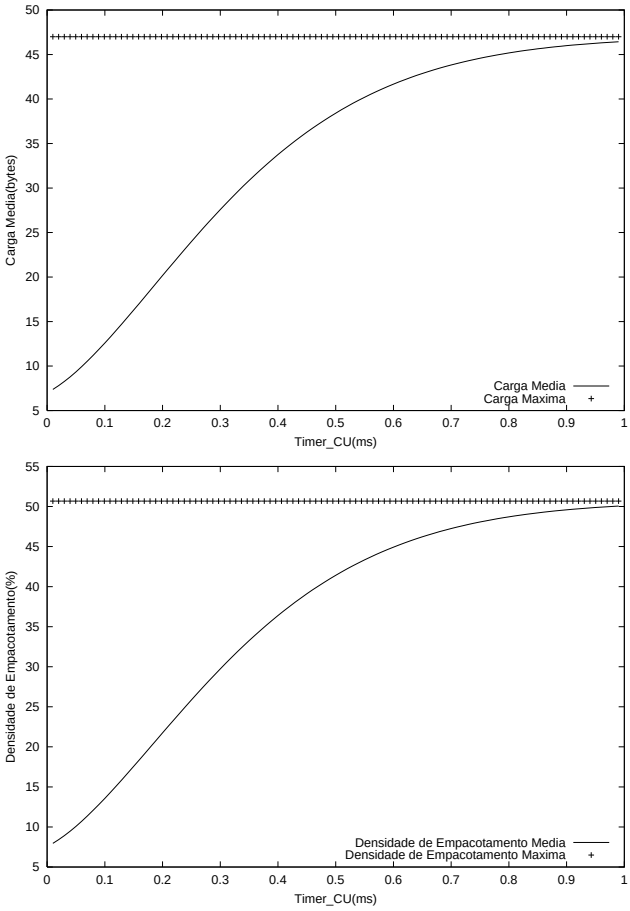


Figura 7.2: Gráfico de variação da carga média (esquerda) e da densidade de empacotamento média (direita) em função do *Timer_ CU*, para 15 fontes de voz. Com o aumento do *Timer_ CU*, diminui a probabilidade de que nenhum pacote de voz chegue neste intervalo de tempo. Em conseqüência disso, os valores de C (carga média) e D (densidade de empacotamento) aproximam-se dos valores máximos quando o *Timer_ CU* atinge 0.99ms. Isso também pode ser observado na Tabela 7.5.

No gráfico da direita na Figura 7.2 é possível observar que a densidade de empacotamento cresce, aproximando-se da máxima, conforme aumenta o *Timer_ CU*. Isso é explicado pelo fato de que, quanto mais tempo a AAL 2 espera por um pacote, maior a probabilidade de que ele chegue e, portanto, mais pacotes serão enviados em uma mesma PDU. A Tabela 7.5 mostra, na segunda coluna, os valores decrescentes para a probabilidade de que nenhum pacote chegue em τ ms (p). O gráfico da esquerda na Figura 7.2 mostra o crescimento da carga média com o aumento do *Timer_ CU*, explicado da mesma forma que a densidade de empacotamento.

Na Figura 7.3 observa-se o descréscimo da probabilidade de não chegar pacotes em τms com o aumento do *Timer_ CU*. Se a AAL 2 espera mais tempo por algum pacote, a probabilidade de que nenhum chegue é menor.

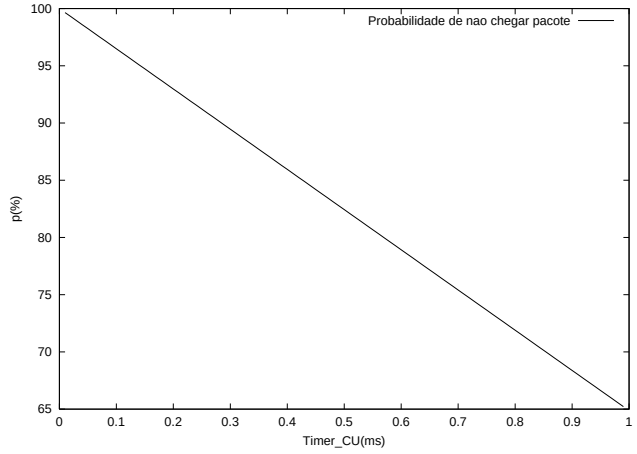


Figura 7.3: Gráfico da probabilidade de não chegarem pacotes de voz em função do *Timer_ CU*. Em um intervalo de tempo muito pequeno, a probabilidade de que nenhum pacote seja recebido é bem próxima de 100%, uma vez que o tempo de espera é curto. Quanto mais tempo espera-se por um pacote, maior a chance de que ele seja recebido. Em função do tamanho do pacote transmitido e do *Timer_ CU* máximo possível, a probabilidade (p) mínima é em torno de 65%.

Considerando agora o *Timer_ CU* com seu valor máximo, $\tau = 0.99$ms, e variando o número de fontes entre 2 e 80, foram gerados os valores para carga média e densidade de empacotamento média, mostrados nos gráficos da Figura 7.4. Observa-se que ambos valores atingem o máximo quando são utilizadas aproximadamente 20 fontes de voz.

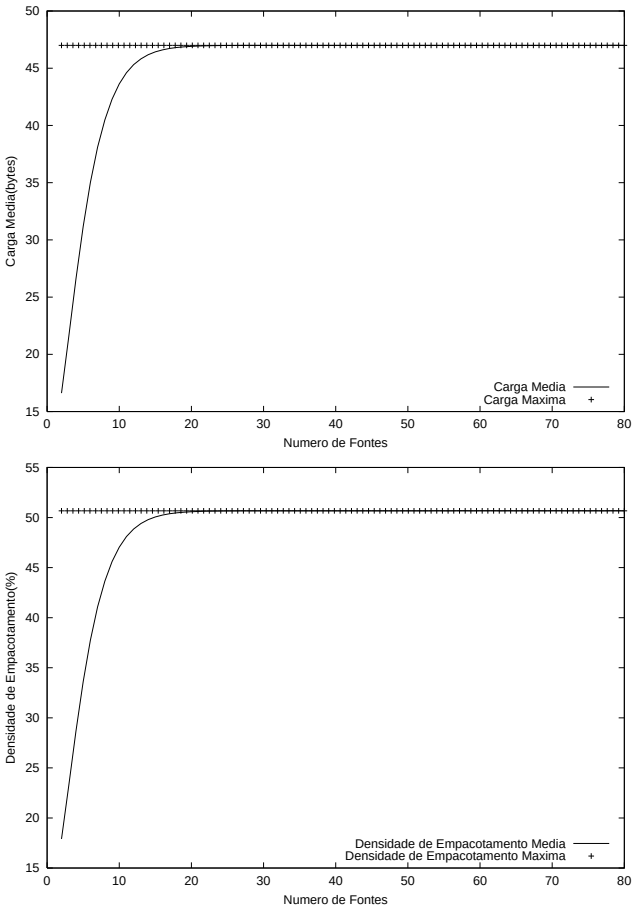


Figura 7.4: Gráficos de variação da carga média (esquerda) e da densidade de empacotamento média (direita) em função do número de fontes de voz. Com a utilização de poucas fontes não é possível atingir os valores máximos de carga e densidade de empacotamento nas PDUs da AAL 2.

Uma última avaliação tenta observar um compromisso adequado entre número de fontes e *Timer_ CU*. Os gráficos apresentados na Figura 7.5 mostram, para cada número de fontes utilizado, o tempo necessário para que se obtenha a carga e a densidade de empacotamento máximas, observando sempre o limite imposto ao *Timer_ CU*. Utilizando menos do que 45 fontes, aproximadamente, os maiores valores para carga e densidade de empacotamento somente são alcançados com um *Timer_ CU* máximo, ou seja, $\tau = 0.99$ms. Mas isso não significa que foram atingidos os valores máximos para carga e densidade. Como observado na Figura 7.4, mesmo com o valor máximo para *Timer_ CU*, a carga e a densidade máximas só são obtidas se forem utilizadas, no mínimo, cerca de 20 fontes de voz.

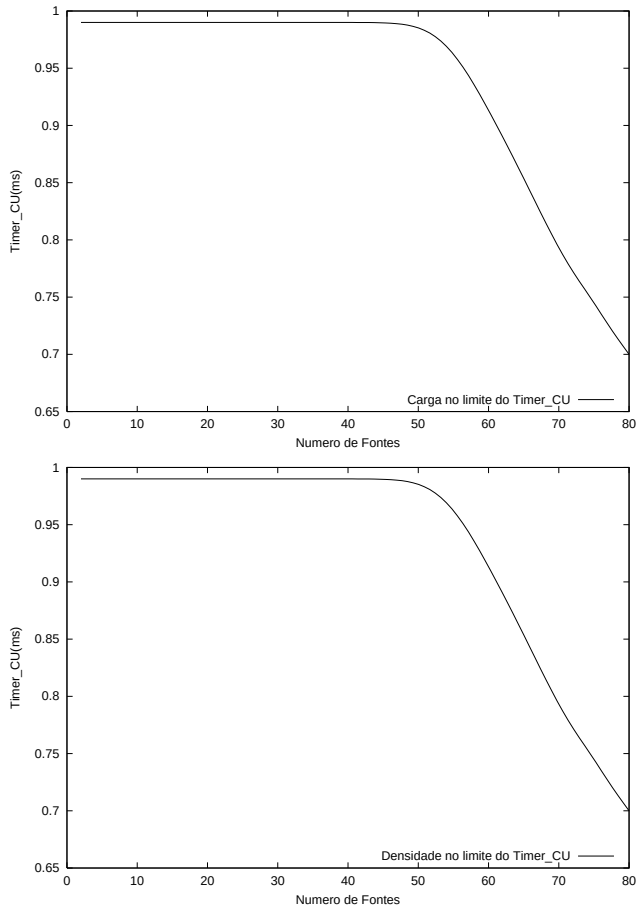


Figura 7.5: Gráficos de carga máxima (esquerda) e densidade de empacotamento máxima (direita) obtidas em função do número de fontes e do *Timer_ CU*. Para uma quantidade de fontes inferior a 45, desejando-se a máxima densidade de empacotamento, o valor mais apropriado para o *Timer_ CU* é $\tau = 0.99$ms, sendo que abaixo de 20 fontes não serão alcançadas a carga e a densidade máximas. Para uma quantidade superior de fontes, o valor do *Timer_ CU* diminui à medida que aumenta o número de fontes.

7.3.1 Conclusões

Um dos objetivos principais em uma transmissão é conseguir o melhor aproveitamento do meio utilizado, enviando a maior quantidade de informações possível no menor tempo. No caso da AAL 2, o *Timer_ CU* é um fator fundamental para o desempenho da aplicação. Para isso é importante saber qual a carga máxima e a densidade de empacotamento máxima possíveis, e determinar a carga e densidade médias para saber o desempenho da aplicação em certas condições.

A aplicação mostrada no Capítulo 4 sugere a utilização de apenas 4 fontes de voz. Neste caso, o valor adequado para o *Timer_ CU* é o máximo, ou seja, $\tau = 0.99$ms. Ainda assim, não será possível obter-se a carga máxima ou a densidade máxima em uma transmissão, e o desempenho da aplicação pode não ser satisfatório.

Uma solução seria aumentar o número de fontes de voz, preferencialmente para um valor acima de 20. Dessa forma seria possível obter carga e densidade máximas utilizando o mesmo *Timer_ CU* de $0.99ms$.

Capítulo 8

Integração das AALs com a Camada ATM

Como fechamento do projeto de *hardware* do *Driver* ATM foi feita a integração dos módulos de recepção de voz, através da AAL 2, e de dados, através da AAL 5, com o módulo da camada ATM. Um projeto correlato, descrito em [CAS02], desenvolveu a implementação em *hardware* para o módulo de recepção de células pela camada ATM que será utilizado neste trabalho, uma vez que a camada ATM está fora do escopo do mesmo. A validação dessa implementação conjunta foi feita através de simulações na ferramenta Active HDL, da Aldec. Nas seções seguintes serão discutidos alguns detalhes relevantes da camada ATM e da estratégia de validação adotada nesta integração de *hardware*.

8.1 Implementação da Camada ATM

A camada ATM foi implementada como parte de um projeto correlato descrito em [CAS02]. Esse projeto gerou, do ponto de vista das AALs, um módulo de recepção de células ATM.

O meio físico, através da camada física, envia à camada ATM células de 53 bytes, onde os 5 primeiros constituem o cabeçalho que será consumido pela mesma, e não repassado às AALs. Desse cabeçalho são extraídas informações importantes para o processo de recepção. Primeiro, os valores dos campos VCI e VPI são utilizados para determinar que AAL deve receber os bytes de carga útil, a AAL 2 ou a AAL 5. A maneira como isso é feito não diz respeito às AALs, e portanto, não será descrita. O procedimento de interpretação dos valores de VCI e VPI pode ser encontrado em [CAS02]. Segundo, os valores dos campos `AUU`, `LP` e `CI` são utilizados para a composição do parâmetro de comunicação da camada ATM com as AALs, em especial com a AAL 5. Esses valores, de um bit cada, são enviados à AAL antes dos 48 bytes de carga útil da célula. No caso da AAL 5, esse parâmentro determina, por exemplo, se a PDU que está sendo recebida é a última e, portanto, contém o *trailer*. No caso da AAL 2, o parâmetro não é utilizado. A camada ATM comunica-se com as AALs através de interfaces, cada uma com um *buffer* de tamanho parametrizável em função do número de células.

8.2 Estratégia de Validação do *Driver* ATM

Para validar os módulos implementados e integrados, foi desenvolvido um *testbench* que engloba duas partes distintas da operação de recepção. Por um lado, realiza as funções da camada física, juntamente com sua interface UTOPIA, sendo capaz de gerar padrões de células

Tabela 8.1: Padrões válidos para pacotes de voz (*CPS-Packets*).

Fonte de Voz	**CID**	**LI**	**UUI**	**HEC**	**Payload**
1	8	4	0	$8 = 1000\text{b} = x^3$	0 1 2 3
2	9	4	0	$19 = 10011\text{b} = x^4 + x + 1$	0 1 2 3
3	10	4	0	$27 = 11011\text{b} = x^4 + x^3 + x + 1$	0 1 2 3
4	11	4	0	$0 = 0\text{b} = 0$	0 1 2 3

ora para a AAL 2, ora para a AAL 5. Por outro, desempenha as funções da Interface DSP, no caso de pacotes de voz, ou do Adaptador Ethernet, no caso da transferência de dados, fazendo as verificações necessárias.

As primeiras validações foram feitas apenas com a AAL 2 integrada à camada ATM. O *testbench*, que já havia sido implementado para validação apenas da camada ATM, foi adaptado para gerar células com valores válidos para os pacotes de voz. Essa validação foi feita supondo a existência de quatro fontes de voz e, portanto, quatro padrões distintos de *CPS-Packets*. Esses padrões são apresentados na Tabela 8.1.

Os valores para o campo `CID`, como visto na Seção 3.5, não podem estar entre 0 e 7. Assim, os valores escolhidos, de 8 a 11, são válidos e podem ser utilizados no *testbench* para validação do módulo. Pacotes com campo `CID` igual a zero representam bytes de enchimento (`PAD`). O segundo campo, `LI`, contém o tamanho da informação contida em um pacote de voz. Neste caso, utiliza-se `LI = 4`, como resultado de uma compactação ADPCM, com taxa de 32 kbps. Esse tipo de compactação foi escolhido por ser um valor intermediário dentre os possíveis em compactações ADPCM, mas a estimativa de desempenho apresentada no Capítulo 4 já mostrava que qualquer das taxas obtidas com ADPCM era suportada pelo *Driver* ATM. O campo `UUI` não é utilizado pela AAL 2, e por isso seu valor é zero. Finalmente, o último campo, `HEC`, contém o resultado de um cálculo de verificação de erros para os três campos anteriores. Esse cálculo é detalhado na Recomendação 363.2 do ITU-T [UNI97].

Como o procedimento é de recepção, o que é gerado pela parte do textittestbench que representa a camada física são células ATM que contenham PDUs para a AAL 2, que por sua vez contém os pacotes de voz mostrados anteriormente. Para cada AAL2-PDU calcula-se o byte de `STF`, com os valores adequados de paridade e número de seqüência, além dos bits de `OSF`, que indicam onde começa o próximo pacote de voz. Ainda, para cada célula, compõe-se um cabeçalho de 5 bytes com valores válidos para os campos de VCI e VPI. Esse valores devem estar de acordo com o que foi configurado, para que a camada ATM possa distinguir as células: se devem ser encaminhadas à AAL 2 ou à AAL 5. Atualmente, só é possível alterar essa configuração por síntese, mas estuda-se a possibilidade de reconfiguração parcial para este caso, como já mencionado também no Capítulo 4.

Uma célula pronta é então enviada à camada ATM, como sendo originária da camada física. Na camada ATM, o cabeçalho da célula é consumido e não é repassado a nenhuma das AALs. Desse cabeçalho são retirados os campos de VCI e VPI, que determinam para qual AAL deverão ir os 48 bytes de carga útil da célula. Porém, antes de enviar esses bytes, a camada ATM envia um byte contendo um parâmetro utilizado na sua comunicação com as AALs. Se o destino desses bytes for a AAL 2, esse parâmetro será descartado. Na outra extremidade da recepção, o *testbench* também representa a Interface DSP, que recebe os dados da AAL 2 e os envia ao DSP.

A segunda validação foi feita integrando-se a AAL 5 ao conjunto AAL 2/ATM/UTOPIA.

Para este caso, o *testbench* precisa gerar os padrões de células adequados para a AAL 5. PDUs para a AAL 5 contêm oito bytes de *trailer* ao final. Se um mesmo pacote Ethernet ocupar mais de uma PDU, apenas a última contém o *trailer*. Alguns padrões de teste para AAL5-PDUs encontram-se ao final da Recomendação 363.5 do ITU-T [UNI96], com valores corretos para os campos que compõem o *trailer*. Para este caso, o cabeçalho da célula ATM deverá conter valores em VCI e VPI que encaminhem a informação útil da célula para a AAL 5. Na outra extremidade da recepção, o *testbench* comporta-se como o adaptador Ethernet, respondendo ao *handshake* de comunicação do adaptador com a AAL 5, descrito na Seção 5.1.

A terceira e última validação foi feita com o *testbench* gerando as células ora para a AAL 2, ora para a AAL 5, de maneira aleatória. O resultado final pode ser observado na Figura 8.1, e o *testbench* utilizado encontra-se no Apêndice A.

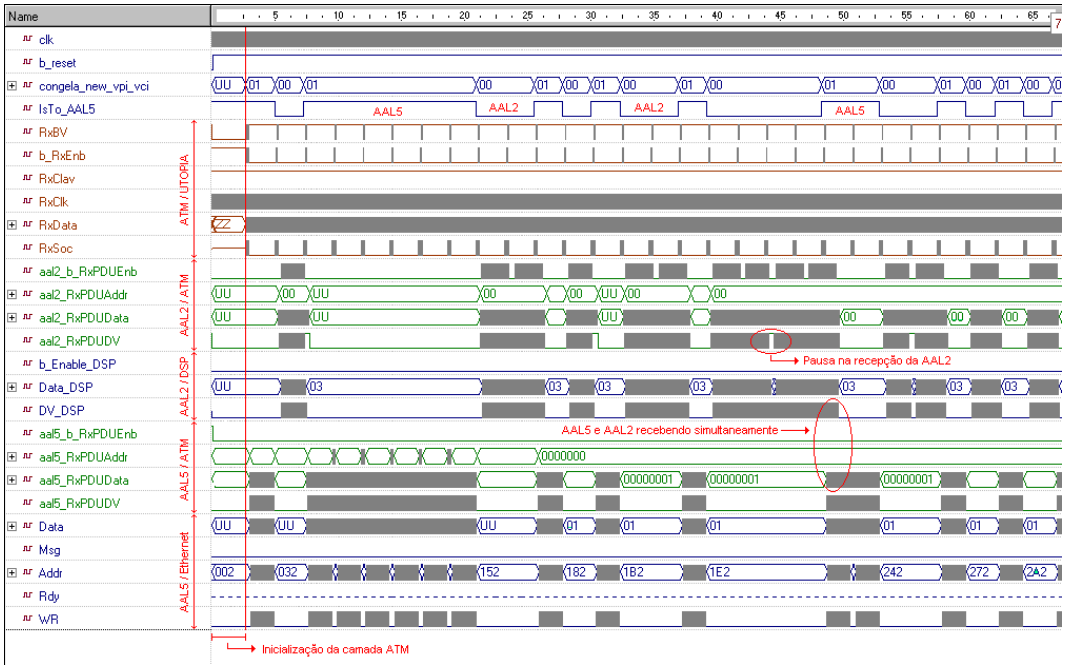


Figura 8.1: Integração da camada ATM com as AALs 2 e 5.

No início da recepção, a camada ATM necessita de um intervalo de tempo para a inicialização das memórias CAM, que contém os endereços válidos de VCI e VPI. Maiores detalhes sobre a implementação da camada ATM, como já foi mencionado, podem ser encontrados em [CAS02].

A Figura 8.1 mostra os sinais das interfaces envolvidas. Para a interface da camada ATM com o controlador UTOPIA são utilizados os sinais `RxBV`, `b_RxEnb`, `RxClav`, `RxClk`, `RxData` e `RxSoc`. Na interface da camada ATM com as AALs existem os sinais `b_RxPDUEnb`, `RxPDUAddr`, `RxPDUData` e `RxPDUDV`, sendo um grupo para cada AAL (identificados pelos prefixos `_aal2` e `_aal5`). Por fim, as interfaces das AALs com suas camadas superiores possuem os sinais `b_Enable_DSP`, `Data_DSP` e `DV_DSP`, para a comunicação da AAL 2 com a Interface DSP, e `Data`, `Msg`, `Addr`, `Rdy` e `WR`, para a comunicação da AAL 5 com o adaptador Ethernet. Além destes, outros quatro sinais são mostrados na Figura 8.1: os sinais de *clock* e *reset* (`clk` e `b_reset`), o sinal de indica para qual AAL a célula deve ser enviada (`IsTo_AAL5`), e o sinal que indica qual o VCI e VPI atuais da célula (`congela_new_vpi_vci`). Convencionou-se nesta integração que VCI=1024 e que, quando VPI=0, a célula deve ser enviada à AAL 2, e quando

VPI=8, a célula é para a AAL 5. Estes valores são válidos e arbitrários, sem qualquer critério de escolha.

O *testbench* gera aleatoriamente valores para o sinal `congela_new_vpi_vci`, ora igual a 0, ora igual a 1. De acordo com o valor deste sinal é gerada uma célula com VCI e VPI correspondentes à AAL 2, ou com VCI e VPI correspondentes à AAL 5. Esta célula é então entregue à camada ATM que, avaliando o cabeçalho, determina qual a AAL deve receber a PDU obtida a partir da célula. A Figura 8.1 mostra os períodos onde a AAL 2 recebe PDUs e onde é a AAL 5 que as recebe. É possível observar esses períodos pelo sinal `IsTo_AAL5`. Se `IsTo_AAL5=0`, a PDU é para a AAL 2, caso contrário, é para a AAL 5.

No intervalo de tempo entre $40\mu s$ e $50\mu s$, ocorre uma parada na recepção através da AAL 2. Embora a camada de adaptação não pudesse receber novos bytes, a camada ATM continua preenchendo o *buffer* de comunicação com a AAL 2. Quando a AAL 2 estiver pronta novamente, irá retomar a recepção buscando bytes nesse *buffer*. Em um determinado instante, em torno de $47\mu s$, a camada ATM termina de colocar no *buffer* a PDU destinada à AAL 2, e inicia o preenchimento do outro *buffer*, com uma PDU para a AAL 5. Dessa maneira, ambas as camadas de adaptação estarão recebendo bytes simultaneamente durante um certo intervalo de tempo. Isso é possível devido a maneira como a camada ATM foi configurada para este estudo de caso, disponibilizando dois *buffers* para comunicação com até 128 AALs em cada um. No estudo aqui descrito, são apenas duas AALs, e optou-se por colocar cada uma vinculada a um *buffer*. Detalhes sobre o projeto da camada ATM podem ser encontrados em [CAS02].

Capítulo 9

Conclusões e Trabalhos Futuros

Os estudos realizados neste trabalho sobre a camada de protocolos denominada camada de adaptação ATM (AAL) mostraram que esta é a principal parte da tecnologia ATM a garantir independência de aplicações que utilizem a B-ISDN. A flexibilidade da tecnologia ATM para suportar diversos tipos de aplicações com variado grau de qualidade de serviço em seus requisitos deriva em boa parte das definições de diferentes formatos para a camada AAL.

Neste Capítulo apresentam-se algumas conclusões, junto com perspectivas de continuação do trabalho aqui iniciado.

9.1 Conclusões

No Capítulo 4 foi definido um contexto de aplicações, qual seja a implementação de *drivers* ATM para suportar a transmissão simultânea de dados e voz. Neste sentido, o trabalho desenvolvido, mesmo limitado a apenas algumas aplicações no domínio da multimídia, é genérico o suficiente para servir a uma gama de produtos com variadas exigências de desempenho.

A implementação da AAL 5 pode suportar velocidades de até 155Mbps, usando a interface padrão UTOPIA nível 2. Isto é muito mais que a faixa de desempenho prevista no presente para modems ADSL, e pode ser usado, por exemplo, em aplicações onde o *driver* funcione como parte de um sistema de multiplexação de diversos canais de dados. Além disto, a forma de implementação do módulo AAL 5 faz com que sua interface possa ser adaptada a operar com outras interfaces de rede no lado do equipamento terminal, não apenas através de adaptadores Ethernet 10/100. Pode-se imaginar o emprego do módulo AAL 5 em redes Token-Ring ou mesmo sem o uso de um padrão de comunicação *upstream* como parte interna de um equipamento terminal. Isto implica a substituição da ponte Ethernet por outro módulo adequado, ou sua mera supressão. A emergência de redes Gigabit, seguindo o padrão Ethernet ou não, é outra possibilidade, embora neste caso uma avaliação do desempenho seja necessária para rever as necessidades de bufferização na interface entre a AAL 5 e a ponte, dada a maior largura de banda de redes Gigabit.

A implementação da AAL 2, por outro lado, excedeu facilmente as especificações do `driver`, visto que a idéia deste foi de prover até 4 conexões de voz simultâneas à transmissão de dados via AAL 5. A implementação da AAL 2 e sua avaliação quantitativa, realizada no Capítulo 7, demonstraram que este valor leva a uma baixa utilização do meio de transmissão, mesmo que estes últimos resultados devam ser melhor avaliados em vista da multiplexação existente no escopo do *driver* de pacotes de dados e voz, desconsiderada na avaliação da AAL 2. As implementações em *software* para a AAL 2 ainda se encontram ainda em fase

muito preliminar para fornecer dados conclusivos sobre como sua utilização se compara com a implementação em *hardware*.

As implementações em *hardware* da AAL 2 e da AAL 5 foram integradas com um módulo, também de *hardware*, que provê serviços da camada ATM. A implementação deste último módulo é discutida em trabalho correlado [CAS02]. A integração foi validada funcionalmente através de simulações funcionais das descrições dos módulos em VHDL.

A validação em *hardware* propriamente dita é dificultada pela não disponibilidade de equipamentos de teste para redes ATM, que pudessem ser utilizados nos testes dos protótipos desses módulos em FPGAs.

9.2 Trabalhos Futuros

A implementação das camadas AAL 2 e AAL 5 foi parcialmente realizada, para validar as idéias desenvolvidas aqui. Divisa-se diversos caminhos para dar continuidade a este trabalho, alguns dos quais são discutidos a seguir.

Primeiro, é necessário completar a implementação dos módulos de propriedade intelectual AAL 2 e AAL 5. A implementação deste trabalho limitou-se, por questões de tempo disponível à parte de recepção em ambos casos. A implementação da parte de transmissão para AALs 2 e 5 foram analisadas e possuem complexidade análoga da parte de recepção.

Segundo, etapas de prototipação em *hardware* do tipo FPGA devem ser realizadas para validar os módulos de forma mais confiável e de forma a se ter uma idéia do porte dos módulos implementados, em termos de ocupação em um sistema. Uma boa medida é a avaliação em termos de número de portas lógicas equivalentes ocupadas pelos módulos em um FPGA ou em um circuito dedicado do tipo ASIC.

Terceiro, após obtidos resultados das etapas descritas nos parágrafos anteriores, e após trabalhos equivalentes serem conduzidos para a camada ATM, planeja-se a integração em *hardware* dos módulos AAL 2 e AAL 5 com o módulo ATM, completando assim a implementação de um protótipo para o *driver* ATM.

Apêndice A

Testbench da Integração dos Módulos de *Hardware*

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.CONV_STD_LOGIC_VECTOR;

-- Add your library and packages declaration here ...

entity driver_atm_tb is
-- Generic declarations of the tested unit
generic(
   N_up_layers : NATURAL := 2;
   N_bits_addr_up_layers : NATURAL := 7;
   N_bits_addr_memory : NATURAL := 3;
   N_bytes_to_ib : NATURAL := 50;
   N_bits_addr_rom : NATURAL := 5;
   Address_AAL2 : NATURAL := 0;
   Address_AAL5 : NATURAL := 0;
   N_connection: NATURAL := 2
);
end driver_atm_tb;

architecture TB_ARCHITECTURE of driver_atm_tb is
-- Component declaration of the tested unit
component driver_atm
generic(
   N_up_layers : NATURAL := 2;
   N_bits_addr_up_layers : NATURAL := 7;
   N_bits_addr_memory : NATURAL := 3;
   N_bytes_to_ib : NATURAL := 50;
   N_bits_addr_rom : NATURAL := 5;
   Address_AAL2 : NATURAL := 0;
   Address_AAL5 : NATURAL := 0 );
port(
```

```
   clk : in std_logic;
   b_reset : in std_logic;
   RxClk : out std_logic;
   b_RxEnb : out std_logic;
   RxData : in std_logic_vector(7 downto 0);
   RxSoc : in std_logic;
   RxClav : in std_logic;
   n_words : in std_logic_vector((N_bits_addr_rom-1) downto 0);
   rom_addr : out std_logic_vector((N_bits_addr_rom-1) downto 0);
   rom_data : in std_logic_vector(23 downto 0);
   Data_DSP : out std_logic_vector(7 downto 0);
   DV_DSP : out std_logic;
   b_Enable_DSP : in std_logic;
   Data : out std_logic_vector(7 downto 0);
   Addr : out std_logic_vector(10 downto 0);
   WR : out std_logic;
   Msg : out std_logic;
   Rdy : in std_logic);
end component;

component LFSR8_8E
port(
   clock : in std_logic;
   b_enable : in std_logic;
   q : out bit_vector(7 downto 0));
end component;

-- Stimulus signals - signals mapped to the input and inout ports of tested entity
signal clk : std_logic;
signal b_reset : std_logic;
signal RxData : std_logic_vector(7 downto 0);
signal RxSoc : std_logic;
signal RxClav : std_logic;
signal n_words : std_logic_vector((N_bits_addr_rom-1) downto 0);
signal rom_data : std_logic_vector(23 downto 0);
signal b_Enable_DSP : std_logic;
-- Observed signals - signals mapped to the output ports of tested entity
signal RxClk : std_logic;
signal b_RxEnb : std_logic;
signal rom_addr : std_logic_vector((N_bits_addr_rom-1) downto 0);
signal Data_DSP : std_logic_vector(7 downto 0);
signal DV_DSP : std_logic;
signal Data : std_logic_vector(7 downto 0);
signal Addr : std_logic_vector(10 downto 0);
signal WR : std_logic;
signal Msg : std_logic;
signal Rdy : std_logic;
-- Add your code here ...
```

```
signal enable_lfsr : std_logic;
signal out_lfsr : bit_vector(7 downto 0);

type RX_STATES is (RX, STOPPED);
signal rx_scurrent, rx_snext : RX_STATES;

signal sigRxdata: std_logic_vector(7 downto 0);
signal sig_b_RxEnb: std_logic;  -- para recepcao

signal RxBV : std_logic;
signal Rx_BVM : std_logic;
signal sig_rx_BVM : std_logic;

type type_rom is array(2**N_bits_addr_rom -1 downto 0) of
                 std_logic_vector(23 downto 0);
signal rom : type_rom;

type Tp_Header is array (4 downto 0) of std_logic_vector(7 downto 0);
signal ATM_Header : Tp_Header;

type Tp_Trailer is array (7 downto 0) of std_logic_vector(7 downto 0);
signal AAL5_Trailer : Tp_Trailer;

signal new_vpi_vci : std_logic_vector(7 downto 0);
signal congela_new_vpi_vci : std_logic_vector(7 downto 0);

signal OSF_tb : std_logic_vector(5 downto 0);
signal SN_tb : std_logic;
signal P_tb : std_logic;
signal Contador : std_logic_vector(5 downto 0);
signal Contador2 : std_logic_vector(2 downto 0);
signal Contador3 : std_logic_vector(7 downto 0);
signal Ultima : std_logic;
signal PARAM_Signal : std_logic_vector(7 downto 0);
signal STF_Signal : std_logic_vector(7 downto 0);
signal PH1_Signal : std_logic_vector(7 downto 0);
signal PH2_Signal : std_logic_vector(7 downto 0);
signal PH3_Signal : std_logic_vector(7 downto 0);
signal I1_Signal : std_logic_vector(7 downto 0);
signal I2_Signal : std_logic_vector(7 downto 0);
signal I3_Signal : std_logic_vector(7 downto 0);
signal I4_Signal : std_logic_vector(7 downto 0);
signal IsTo_AAL5 : std_logic;
signal not_clk : std_logic;

begin

-- Unit Under Test port map
```

```
UUT : driver_atm
generic map(
   N_up_layers => N_up_layers,
   N_bits_addr_up_layers => N_bits_addr_up_layers,
   N_bits_addr_memory => N_bits_addr_memory,
   N_bytes_to_ib => N_bytes_to_ib,
   N_bits_addr_rom => N_bits_addr_rom,
   Address_AAL2 => Address_AAL2,
   Address_AAL5 => Address_AAL5 )
port map(
   clk => clk,
   b_reset => b_reset,
   RxClk => RxClk,
   b_RxEnb => b_RxEnb,
   RxData => RxData,
   RxSoc => RxSoc,
   RxClav => RxClav,
   n_words => n_words,
   rom_addr => rom_addr,
   rom_data => rom_data,
   Data_DSP => Data_DSP,
   DV_DSP => DV_DSP,
   b_Enable_DSP => b_Enable_DSP,
   Data => Data,
   Addr => Addr,
   WR => WR,
   Msg => Msg,
   Rdy => Rdy );

XXX : LFSR8_8E
port map(
   clock => not_clk,
   b_enable => enable_lfsr,
   q => out_lfsr);

-- Add your stimulus here ...

not_clk <= not(clk);

process -- clock
begin
   clk <= '1','0' after 20ns;
   wait for 40ns;
end process;

process(b_Reset, clk) -- conta ate 53
begin
   if b_Reset = '0' then
```

```
      Contador <= (others => '0');
      Contador2 <= (others => '0');
      Contador3 <= (others => '0');
   elsif clk'event and clk = '0' then
      if RxBV = '1' then
         if Contador = 52 then
            Contador <= (others => '0');
         else
            Contador <= Contador + 1;
         end if;
         if Contador3 = 52 then
            Contador3 <= (others => '0');
         else
            Contador3 <= Contador3 + 1;
         end if;
         -- 0, 1, 2, 3, 4 para cabecalho ATM, 5 para stf
         if Contador > 5 and IsTo_AAL5 = '0' then
            if Contador2 = 6 then
               Contador2 <= (others => '0');
            else
               Contador2 <= Contador2 + 1;
            end if;
         end if;
      end if;
   end if;
end process;

sigRxdata <= ATM_Header(0) when Contador = 0 else
    ATM_Header(1) when Contador = 1 else
    ATM_Header(2) when Contador = 2 else
    ATM_Header(3) when Contador = 3 else
    ATM_Header(4) when Contador = 4 else
    STF_Signal when Contador = 5 and IsTo_AAL5 = '0' else -- OSF
    PH1_Signal when Contador2 = 0 and IsTo_AAL5 = '0' else
    PH2_Signal when Contador2 = 1 and IsTo_AAL5 = '0' else
    PH3_Signal when Contador2 = 2 and IsTo_AAL5 = '0' else
    I1_Signal when Contador2 = 3 and IsTo_AAL5 = '0' else
    I2_Signal when Contador2 = 4 and IsTo_AAL5 = '0' else
    I3_Signal when Contador2 = 5 and IsTo_AAL5 = '0' else
    I4_Signal when Contador2 = 6 and IsTo_AAL5 = '0' else
    (Contador3 - 4) when Ultima = '1' and Contador3 <= 44
                                      and IsTo_AAL5 = '1' else
    (Contador3 - 4) when Ultima = '0' and IsTo_AAL5 = '1' else
    AAL5_Trailer(0) when Ultima = '1' and Contador3 = 45
                                      and IsTo_AAL5 = '1' else
    AAL5_Trailer(1) when Ultima = '1' and Contador3 = 46
                                      and IsTo_AAL5 = '1' else
    AAL5_Trailer(2) when Ultima = '1' and Contador3 = 47
```

```
                                        and IsTo_AAL5 = '1' else
    AAL5_Trailer(3) when Ultima = '1' and Contador3 = 48
                                        and IsTo_AAL5 = '1' else
    AAL5_Trailer(4) when Ultima = '1' and Contador3 = 49
                                        and IsTo_AAL5 = '1' else
    AAL5_Trailer(5) when Ultima = '1' and Contador3 = 50
                                        and IsTo_AAL5 = '1' else
    AAL5_Trailer(6) when Ultima = '1' and Contador3 = 51
                                        and IsTo_AAL5 = '1' else
    AAL5_Trailer(7) when Ultima = '1' and Contador3 = 52
                                        and IsTo_AAL5 = '1';

compute_stf:process(b_Reset,clk,Contador)
begin
    if b_Reset = '0' then
        SN_tb <= '0';
    elsif clk'event and clk = '1' then
        if Contador = 8 and IsTo_AAL5 = '0' then
            SN_tb <= not SN_tb;
        end if;
    end if;
end process;

OSF_tb(5 downto 3) <= "000";
OSF_tb(2 downto 0) <= 7 - Contador2 when Contador2 /= 0 else "000";
P_tb <= not(OSF_tb(0) xor OSF_tb(1) xor OSF_tb(2) xor OSF_tb(3) xor OSF_tb(4)
        xor OSF_tb(5) xor SN_tb);

STF_Signal <= OSF_tb & SN_tb & P_tb;
PH1_Signal <= x"08";
PH2_Signal <= x"10";
PH3_Signal <= x"08";
I1_Signal <= "00000000";
I2_Signal <= "00000001";
I3_Signal <= "00000010";
I4_Signal <= "00000011";

ATM_Header <= ("00000100","00000011","00000010","00000001","00000000")
              when Ultima = '1' and congela_new_vpi_vci = "00000001" else
              ("00000100","00000001","00000010","00000001","00000000")
              when Ultima = '0' and congela_new_vpi_vci = "00000001" else
              ("00000100","00000011","00000010","00000000","00000000");

AAL5_Trailer <= (x"d0",x"1e",x"67",x"bf",x"28",x"00",x"00",x"00")
                when Ultima = '1' else
                (x"fb",x"f1",x"4f",x"35",x"28",x"00",x"00",x"00");

b_reset <= '0', '1' after 105ns;
```

```
b_Enable_DSP <= '0';

IsTo_AAL5 <= '0' when congela_new_vpi_vci = "00000000" else '1';
Ultima <= '1';

RxClav <= '1';

RxBV <= sig_rx_BVM and RxClav;
enable_lfsr <= '0';

-- Codigo para deducao do valor de RxBV
process(RxClk)
begin
    if RxClk'event and RxClk = '1' then
        sig_rx_BVM <= rx_BVM;
    end if;
end process;

rx_control : process(RxClk, b_reset)
begin
    if b_reset = '0' then
        rx_scurrent <= STOPPED;
    elsif RxClk'event and RxClk = '0' then
        rx_scurrent <= rx_snext;
    end if;
end process;

rx_combinational : process(rx_scurrent, b_RxEnb)
begin
    case rx_scurrent is
        when RX => -- recepting
            rx_BVM <= '1';
            if b_RxEnb = '0' then
                rx_snext <= RX;
            elsif b_RxEnb = '1' then
                rx_snext <= STOPPED;
            end if;
        when STOPPED => -- stopped
            rx_BVM <= '0';
            if b_RxEnb = '1' then
                rx_snext <= STOPPED;
            elsif b_RxEnb = '0' then
                rx_snext <= RX;
            end if;
    end case;
end process;

process(RxClk)
```

```
begin
    if RxClk'event and RxClk = '0' then
        sig_b_RxEnb <= b_RxEnb;
    end if;
end process;

process (RxClk)
begin
    if RxClk'event and RxClk = '1' then
        if b_reset = '1' then
            if sig_b_RxEnb = '0' then
                if Contador = 0 then
                    RxSoc <= '1';
                else
                    RxSoc <= '0';
                end if;
                if Contador = 0 then -- new_vpi_vci;
                    congela_new_vpi_vci <= "00000001" and TO_STDLOGICVECTOR(out_lfsr);
                end if;
                RxData <= sigRxdata;
            else
                RxSoc <= 'Z';
                RxData <= "ZZZZZZZZ";
            end if;
        end if;
    end if;
end process;

mem:for i in N_connection to 2**N_bits_addr_rom - 1 generate
        rom(i) <= CONV_STD_LOGIC_VECTOR(2**24-1,24);
    end generate;

rom(0) <= "000000000010000000000000";
rom(1) <= "000000000010000000010000";

n_words <= "11111";

rom_data <= rom(CONV_INTEGER(rom_addr));

end TB_ARCHITECTURE;
```

Referências Bibliográficas

[2Wi02] 2Wire, Inc. **Learning Center - DSL**. Disponível por WWW em http://www.2wire.com/support/l_center_bbrc_dsl.html#1 (2002).

[ADA96] ADAMS, Jay K.; THOMAS, Donald E. The Design of Mixed Hardware/Spftware Systems. In: 33RD DESIGN AUTOMATION CONFERENCE, 1996, Las Vegas, NV, USA. **Anais...** 1996.

[ADA97] ADANEZ, Xavier Garcia; VERSCHEURE, Olivier; FROSSARD, Pascal. Reliable Transmission of MPEG-2 VBR Video Streams over New Network Adaptation and ATM Adaptation Layers. In: IEEE ATM WORKSHOP, 1997, Lisboa, Portugal. **Anais...** 1997. (Disponível por WWW em citeseer.nj.nec.com/garciaadanez97reliable.html).

[ADS01] ADSL tutorial. 2001. (Disponível por WWW em http://www.dslforum.org/aboutdsl/adsl_tutorial.html).

[AYA02] AYALON, Eyal; LEVY, Tzachi; KERER, Guy. **ADSL technology description**. Disponível por WWW em http://www.alliancedatacom.com/adsl-tutorial.htm (2002).

[CAS01] CASTANHEIRA, Leonardo Dutra. Redes ATM - Um Estudo de Algoritmos de Gerenciamento de Tráfego e Qualidade de Serviço. Porto Alegre, RS: Pontifícia Universidade Católica do Rio Grande do Sul/Faculdade de Informática - PUCRS/FACIN, 2001. (Trabalho Individual I).

[CAS02] CASTANHEIRA, Leonardo Dutra. **Medição de Desempenho de Hardware ATM no Nível de Célula**. Porto Alegre, RS: Pontifícia Universidade Católica do Rio Grande do Sul, 2002. Dissertação de Mestrado.

[COM01] COMER, Douglas E. **Redes de Computadores e Internet**. Porto Alegre, RS: Bookman, 2001. (segunda ed.).

[COM94] COMMITTEE, The ATM Forum Technical. **UTOPIA Specification Level 1, Version 2.01**. Mountain View, CA: ATM Forum, 1994. (Specification af-phy-0017.000, Disponível por WWW em www.atmforum.com).

[COM95] COMMITTEE, The ATM Forum Technical. **UTOPIA Level 2, Version 1.0**. Mountain View, CA: ATM Forum, 1995. (Specification af-phy-0039.000, Disponível por WWW em www.atmforum.com).

[COM97] COMMITTEE, The ATM Forum Technical. **ATM in Europe - The User Handbook**. European Market Awareness Committee, 1997. (version 1.0).

[COM99] COMMITTEE, The ATM Forum Technical. **UTOPIA 3 Physical Layer Interface**. Mountain View, CA: ATM Forum, 1999. (Specification af-phy-0136.000, Disponível por WWW em www.atmforum.com).

[COM99a] COMMITTEE, The ATM Forum Technical. **Trafic Management Specification Version 4.1**. Mountain View, CA: ATM Forum, 1999. (Specification af-tm-0121.000, Disponível por WWW em www.atmforum.com).

[COM00] COMMITTEE, The ATM Forum Technical. **UTOPIA Level 4**. Mountain View, CA: ATM Forum, 2000. (Specification af-phy-0144.001, Disponível por WWW em www.atmforum.com).

[CON02] CONSORTIUM, International Engineering. **DSL World Forum - Asymmetric Digital Subscriber Line (ADSL)**. Disponível por WWW em http://www.iec.org (setembro de 2002).

[DUT95] DUTTON, Harry J. R.; LENHARD, Peter. **Asynchronous Transfer Mode (ATM) - Technical Overview**. New Jersey, USA: Prentice Hall PTR, 1995.

[EIL00] EILERS, Dirk; VOGLGSNAG, Alfred; PLANKL, Arnold; KöRNER, Gerri; STECKENBILLER, Helmut; KNORR, Rudi. A Prototype of an AAL for High Bit Rate Real-Time Data Transmission System over ATM Networks Using a RSE CODEC. In: 11TH INTERNATIONAL WORKSHOP ON RAPID SYSTEM PROTOTYPING (RSP 2000), 2000. **Anais...** 2000. p.109–114.

[CRC01] GENERATOR of synthesizable CRC functions. 2001. (Disponível por WWW em http://www.easics.be/webtools/crctool).

[GIR98] GIROUX, Natalie; GANTI, Sudhakar. **Quality of Service in ATM networks**. Upper Saddle River, NJ: Prentice Hall PTR, 1998.

[HAN95] HANDEL, Rainer; HUBER, Manfred N.; SCHRODER, Stefan. **ATM Networks. Concepts, Protocols, Applications**. Workingham, England: Addison-Wesley, 1995.

[HUN96] HUNT, Ray. ATM - Protocols and Architecture. **Computer Communications**, v.19, n.6, p.597–611, June 1996.

[KIM97] KIM, Joonhwan; BAHK, Saewoong. Comparison of MPEG Data Transferring Schemes in ATM Networks. **Computer Communications**, v.20, p.825–831, 1997.

[KWO98] KWOK, Timothy. **The New Paradigm for Internet, Intranet, and Residential Broadband Services and Applications**. Upper Saddle River, NJ: Prentice Hall PTR, 1998.

[LIU99] LIU, Chunlei; MUNIR, Sohail; JAIN, Raj. Packing Density of Voice Trunking Using AAL2. In: IEEE GLOBAL TELECOMMUNICATIONS CONFERENCE, 1999. **Anais...** 1999. p.611–615.

[LU01] LU, Shi; THOMPSON, Richard. Improving VOTA AAL2 Signaling Transmitter Performance. In: IEEE GLOBAL TELECOMMUNICATIONS CONFERENCE (GLOBECOM'2001), 2001. **Anais...** 2001. v.1, p.110–114.

[LUN93] LUNN, Andrew; SCOTT, Andrew; SHEPHERD, Doug. ATM, AAL5 and Multimedia Devices. In: 5TH BANGOR COMMUNICATIONS SYMPOSIUM, 1993. **Anais...** 1993. (Disponível por WWW em citeseer.nj.nec.com/175878.html).

[MAR02] MARTIN, Sean. **HDSL**. Disponível por WWW em http://www.arcelect.com/High-bit-rate_Digital_Subscriber_Line-HDSL.htm (2002).

[MCL97] MCLOUGHLIN, Mike; MUMFORD, Keith. A Management Briefing on Adapting Voice for ATM Networks - A Comparison of AAL1 Versus AAL2. Middlebury, Connecticut, USA: General DataComm, 1997.

[MCL97a] MCLOUGHLIN, Mike; O'NEIL, John. A Management Briefing on Adapting Voice for ATM Networks - An AAL2 Tutorial. Middlebury, Connecticut, USA: General DataComm, 1997.

[MEH97] MEHAOUA, Ahmed; BOUTABA, Raouf; PUJOLLE, Guy. Proposal of an Extended AAL5 for VBR MPEG-2 over ATM. In: INTERNATIONAL CONFERENCE ON MULTIMEDIA COMPUTING AND SYSTEMS (ICMCS'97), 1997. **Anais...** 1997.

[NAN02] NANANUKUL, Soracha; KEKKI, Sami. Simulation Studies of Bandwidth Management for the ATM/AAL2 Transport in the UTRAN. In: 56TH IEEE VEHICULAR TECHNOLOGY CONFERENCE (VTC 2002-FALL), 2002. **Anais...** 2002. p.1134–1138.

[O'NI96] O'NILS, Mattias. Hardware/Software Partitioning of Telecommunication Systems. 1996. (Disponível por WWW em citeseer.nj.nec.com/nils96hardwaresoftware.html).

[PET95] PETR, David W.; VATTE, Raghushankar R.; SAMPATH, Prema; LU, Young-Qing. Efficiency of AAL2 for Voice Transport - Simulation Comparison with AAL1 and AAL5. **IEEE MultiMedia**, v.2, n.2, p.60–74, 1995. (Disponível por WWW em citeseer.nj.nec.com/pan96tutorial.html).

[PRY95] PRYCKER, Martin de. **Asynchronous Transfer Mode - Solution for Broadband ISDN**. London, UK: Prentice Hall International (UK) Limited, 1995.

[QIG01] QIGANG, Zhou; XIANGYANG, Zhu; LIZHONG, Gui; PENGFEI, Shi. Logic Design on FPGA of ATM SAR Function in Broadband Access Network. In: 4TH INTERNATIONAL CONFERENCE ON ASIC, 2001. **Anais...** 2001. p.494–497.

[RAH98] RAHMAN, Mohammad A. **Guide to ATM Systems and Technology**. Norwood, MA: Artech House, 1998.

[SOA95] SOARES, Luiz Fernando Gomes; LEMOS, Guido; COLCHER, Sergio. **Redes de Computadores - Das LANs, MANs e WANs às Redes ATM**. Rio de Janeiro, RJ: Editora Campus, 1995.

[SOU01] SOUZA, Sheila Moreira. Redes ATM - Um Estudo das Camadas de Protocolo e suas Interfaces. Porto Alegre, RS: Pontifícia Universidade Católica do Rio Grande do Sul/Faculdade de Informática - PUCRS/FACIN, 2001. (Trabalho Individual I).

[SRI99] SRIRAM, Kotikalapudi; WANG, Yung-Terng. Anomalies Due To Delay and Loss in AAL2 Packet Voice Systems - Performance Models and Methods of Mitigation. **IEEE Journal on Selected Areas in Communications**, v.17, n.1, p.4–17, January 1999.

[SUD98] SUDA, Tatsuya. Asynchronous Transfer Mode (ATM) Networks. In: **Encyclopedia of electrical and electronics engineering**. University of California, Irvine, CA, USA: Wiley and Sons, 1998.

[TAN94] TANEMBAUM, Andrew S. **Redes de Computadores**. Rio de Janeiro: Campus, 1994.

[THA95] THAI, Kim-Loan; FDIDA, Serge. XTP over AAL 5 for Multimedia Applications. 1995. (Disponível por WWW em citeseer.nj.nec.com/245013.html).

[TOR01] TOROK, Delfim Luiz. **Projeto Visando a Prototipação do Protocolo de Acesso ao Meio em Redes Ethernet**. Porto Alegre, RS: Pontifícia Universidade Católica do Rio Grande do Sul, 2001. Dissertação de Mestrado.

[TUT02] TUTORIAL de ADSL. 2002. (Disponível por WWW em http://www.geocities.com/wagnerol/Tutorial.htm).

[UNI90] UNION, International Telecommunication. **General Aspects of Digital Transmission Systems; Terminal Equipments - 40,32,24,16 kbits/s Adaptive Differencial Pulse Code Modulation (ADPCM)**. Geneva: ITU-T, 1990. (Recommendation G.726).

[UNI91] UNION, International Telecommunication. **B-ISDN Protocol Reference Model and its Application**. Geneva: ITU-T, 1991. (Recommendation I.321).

[UNI93] UNION, International Telecommunication. **General Aspects of Digital Transmission Systems; Terminal Equipments - Pulse Code Modulation (PCM) of Voice Frequencies**. Geneva: ITU-T, 1993. (Recommendation G.711).

[UNI93a] UNION, International Telecommunication. **Synchronous Digital Hierarchy Bit Rates**. Geneva: ITU-T, 1993. (Recommendation G.707).

[UNI93b] UNION, International Telecommunication. **B-ISDN ATM Adaptation Layer (AAL) Specification**. Geneva: ITU-T, 1993. (Recommendation I.363).

[UNI96] UNION, International Telecommunication. **B-ISDN ATM Adaptation Layer Specification - Type 5 AAL**. Geneva: ITU-T, 1996. (Recommendation I.363.5).

[UNI96a] UNION, International Telecommunication. **General Aspects of Digital Transmission Systems - Dual Rate Speech Coder for Multimedia Communications Transmitting at 5.3 and 6.3 kbits/s**. Geneva: ITU-T, 1996. (Recommendation G.723.1).

[UNI97] UNION, International Telecommunication. **B-ISDN ATM Adaptation Layer Specification - Type 2 AAL**. Geneva: ITU-T, 1997. (Recommendation I.363.2).

[UNI99] UNION, International Telecommunication. **B-ISDN User-Network Interface - Physical Layer Specification - General Characteristics**. Geneva: ITU-T, 1999. (Recommendation I.432.1).

[UNI99a] UNION, International Telecommunication. **B-ISDN ATM Layer Specification**. Geneva: ITU-T, 1999. (Recommendation I.361).

[UNI02] UNIVERSITY, Indiana. **What is SDSL?** Disponível por WWW em http://kb.indiana.edu/data/ahwl.html (July 2002).