Hugo Jorge F. de Carvalho

Um Circuito Integrado de Aplicação Específica

DEEC
FEUP
1996

UNIVERSIDADE DO PORTO
FACULDADE DE ENGENHARIA

HAL5 II

Um Circuito Integrado de Aplicação Específica
para implementação de um Terminal ATM

Hugo Jorge Ferreira de Carvalho

PORTO, Setembro 1996



FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Departamento de Engenharia Electrotécnica e de Computadores

HAL5 II

Um Circuito Integrado de Aplicação Específica para implementação de um Terminal ATM

681.3(043) CARV/HAL
DEMAC - FACULDADE DE ENGENHARIA DO PORTO
Biblioteca
N.º 6345
Data ____ / 19__

cat

Hugo Jorge Ferreira de Carvalho

Licenciado em Engenharia Electrotécnica e de Computadores
pela Faculdade de Engenharia da Universidade do Porto

DEEC

Dissertação submetida para satisfação parcial dos requisitos
do programa do curso de
Mestrado em Engenharia Electrotécnica e de Computadores
(área de especialização em Informática Industrial)

FEUP

PORTO, Setembro 1996

Dissertação realizada sob a supervisão do
Professor Doutor José Alfredo Ribeiro da Silva Matos,
Professor Associado do
Departamento de Engenharia Electrotécnica e de Computadores da
Faculdade de Engenharia da Universidade do Porto

Agradecimentos

Chegado aqui, sinto que é meu dever expressar publicamente gratidão a todos os que, directa ou indirectamente, contribuíram e apoiaram para levar a bom termo o trabalho a que me propus.

Em particular gostaria de mencionar e enaltecer:

O meu orientador na realização deste trabalho, Professor José Alfredo Ribeiro da Silva Matos, pelos seus conselhos, sugestões e estímulo.

O Eng. João P. Canas Ferreira pela sua ajuda desinteressada, e disponibilidade sempre total.

Por último, quero agradecer à minha Esposa, à minha Família, a confiança que depositaram em mim e na empresa que levei a cabo.

A todos o meu obrigado.

Resumo

A tecnologia de comunicação ATM (Modo de Transferência Assíncrono) foi escolhida para suportar as redes de comunicação de banda larga com integração de serviços (RDIS-BL). A tecnologia de comunicação ATM permite a implementação de redes de comunicação capazes de suportar de forma eficiente, as aplicações ditas de banda larga (multimédia, *video-on-demand*, telemedicina, teletrabalho), que terão um impacto económico e social significativo na sociedade do futuro.

O trabalho aqui apresentado, consistiu no projecto de um Circuito Integrado de Aplicação Específica (CIAE), para desempenhar as funções das camadas AAL (ATM Adaptation Layer) e ATM do modelo de referência protocolar da RDIS-BL sobre ATM. Este CIAE constitui o núcleo de uma placa adaptadora para implementação de um terminal ATM em velocidades até 155Mbit/s.

O projecto do circuito integrado envolveu, além da análise e da concepção da arquitectura do sistema, a modelização e simulação do CIAE numa linguagem de descrição de *hardware* (HDL) - 'Verilog HDL', e posterior mapeamento em tecnologia '*standard-cell*' CMOS 0.7μ, através de ferramentas de síntese de alto nível. A fabricação do CI não foi possível no âmbito deste curso de mestrado e, por isso, não constituiu uma das metas deste trabalho. Contudo, o trabalho desenvolvido permitiu avaliar das características físicas de um eventual protótipo.

Índice

RESUMO ..... **II**

INTRODUÇÃO **1**

REDES ATM **10**

2.1 APLICAÇÕES E SERVIÇOS DE BANDA LARGA 10
2.2 A TECNOLOGIA ATM 12
2.3 ASPECTOS TÉCNICOS BÁSICOS 14
2.3.1 As Células ATM *14*
2.3.2 Os Canais Virtuais *16*
2.3.3 As Funções de Operação e Manutenção *19*
2.3.4 O Meio Físico *22*
2.4 O TERMINAL ATM 24
2.4.1 O Modelo de Referência Protocolar da RDIS-BL *24*
2.4.2 A Camada Física *27*
2.4.3 A Camada ATM *28*
2.4.4 A Camada AAL *29*
2.4.4.1 O Protocolo AAL5 31

ARQUITECTURA DO HAL5 II **33**

3.1 INTRODUÇÃO 33
3.2 A ARQUITECTURA 37
3.3 AS INTERFACES COM O EXTERIOR 41
3.3.1 Interface PCI *41*
3.3.2 Interface UTOPIA *43*
3.4 O FLUXO DOS DADOS 44
3.4.1 Recepção *44*
3.4.2 Transmissão *46*
3.5 AS ESTRUTURAS DE DADOS DE CONTROLO 48
3.5.1 Recepção *48*
3.5.2 Transmissão *52*

COMPONENTES DO HAL5 II **54**

4.1 INTERFACE UTOPIA 54
4.2 CRC32 58
4.3 BLOCO TX 60
4.4 BLOCO RX 65
4.5 INTERFACE PCI 70
4.5.1 BackEnd PCI *70*
4.5.2 Bloco PCI *74*

PROJECTO DIGITAL DO HAL5 II **76**

5.1 PROJECTO DO CIAE 76
5.2 VALIDAÇÃO DA ARQUITECTURA 86

CONCLUSÃO **90**

BIBLIOGRAFIA **92**

APÊNDICE **94**

GLOSSÁRIO **120**

Capítulo 1

Introdução

As Telecomunicações e a Micro-Electrónica desempenham cada vez mais um papel de extrema importância no nosso dia-a-dia. Nos Computadores (em sentido lato, ou seja, como qualquer elemento capaz de processar informação), evoluiu-se de um cenário de autênticas ilhas de informação para um ambiente distribuído e interligado. As Redes de Comunicação, permitiram a partilha da informação e do poder de processamento, e influenciaram a nossa forma de trabalhar e mesmo o relacionamento entre as pessoas. A evolução na tecnologia Micro-Electrónica contribuiu para este fenómeno e também para o desenvolvimento dos elementos que processam a informação, elevando o poder de processamento e permitindo a utilização de novos formatos de dados, novos procedimentos e formas mais evoluídas de interface utilizador-máquina.

Presentemente caminha-se para a "globalização da sociedade de informação", com o despontar de novas aplicações e potencialidades, de que o ensino à distância, a telemedicina, o teletrabalho, a videoconferência e o acesso universal a grandes bases de dados de texto, som e imagem de alta qualidade, são apenas alguns exemplos. As repercussões na sociedade e na economia serão de vulto.

Em contrapartida são colocadas exigências de monta às vias de comunicação de dados, particularmente ao nível da largura de banda. Para fazer face ao extraordinário aumento no débito de informação imposto pelas novas aplicações e serviços, tornou-se necessário desenvolver novas tecnologias de redes de comunicação. De facto, muitas das redes de comunicação existentes recorrem frequentemente à rede pública telefónica para transmissão de dados, a qual foi concebida com um objectivo específico (bem menos exigente) e não se adapta fácil e eficientemente às novas aplicações. As suas limitações incluem uma largura de banda e flexibilidade muito reduzidas. Para ultrapassar as limitações da

rede telefónica pública, surgiram as redes dedicadas - redes de comunicação (públicas ou privadas) interligando, por exemplo, várias instituições de investigação ou as várias dependências de uma Empresa. O exemplo mais conhecido de uma tal rede é a InterNet. Mas mesmo estas redes apresentam problemas, como largura de banda insuficiente (Ethernet: 10Mbit/s), e uma grande variedade de equipamento, interfaces e protocolos que por vezes levantam dificuldades à total interligação dos elementos que constituem a rede.

As Redes Digitais com Integração de Serviços (RDIS) constituíram um primeiro passo no sentido de responder de forma adequada às necessidades emergentes. Uma RDIS é "uma rede de comunicação que oferece conectividade digital entre quaisquer dois pontos da rede, para suportar uma variedade de serviços, incluindo serviços de voz, e à qual os utilizadores têm acesso através de um conjunto finito de interfaces utilizador-rede padronizadas"[18]. A RDIS original é baseada na rede telefónica digital, a qual é caracterizada por uma largura de banda de 64Kbit/s, e proporciona na sua interface de acesso básica dois canais de 64Kbit/s (designados por canais B), e um canal de 16Kbit/s para sinalização (designado por canal D). Com a utilização de mais canais (canais do tipo H, de alta velocidade) é possível atingir cerca de 2Mbit/s. Estas redes estão a ser implementadas nesta década e os seus benefícios para as redes de comunicação e para os seus utilizadores incluem [3]:

- Uma interface entre a rede e o utilizador comum para aceder a uma grande variedade de serviços.
- Capacidades de sinalização melhoradas.
- Integração de serviços.
- Provisão para serviços novos e mais avançados.

Contudo, mesmo os 2Mbit/s que a RDIS pode oferecer mostram-se insuficientes para certas aplicações. Por exemplo, para a interligação entre redes locais ou para a transmissão de imagens em movimento em alta qualidade são necessárias taxas de transferência muito superiores. Esta necessidade levou à concepção da RDIS de Banda Larga, uma extensão da RDIS de 64Kbit/s, com o objectivo inicial de: i) acrescentar canais de alta velocidade aos já existentes; ii) definir novas interfaces utilizador-rede adaptadas à banda larga.

A RDIS de Banda Larga foi concebida com o objectivo de ser uma rede universal, que suporte diferentes tipos de aplicações, serviços, e categorias de utilizadores com diferentes necessidades. Para atingir este objectivo foi escolhida a tecnologia de comunicação ATM (Modo de Transferência Assíncrono)[20]. Outro alicerce importante das RDIS de Banda Larga é o possível e cada vez mais frequente recurso a tecnologia óptica, para a transmissão física dos dados, permitindo taxas de transferência de várias centenas de Mbit/s.

Pela análise das características do tráfego gerado pelas principais Aplicações de Banda Larga, apresentadas na tabela 1.1 [17], é possível concluir que a tecnologia adoptada deve:

i) Ter a capacidade de responder eficazmente a serviços com débitos muito variados, atribuindo e gerindo dinamicamente a largura de banda disponível entre os vários serviços e utilizadores.
ii) Estar preparada para lidar e tirar partido de tráfego do tipo *burst*.
iii) Atender às necessidades das aplicações sensíveis a atrasos e perdas na comunicação.

Serviço	Débito (Mbit/s)	Pico / Médio
Transmissão de Dados	1.5 - 130	1 - 50
Acesso a Bases de Dados	1.5 - 45	1 - 20
Videoconferência/Videotelefone	1.5 - 130	1 - 5
Videotexto Multimédia	1.5 - 130	1 - 20
Televisão	30 - 130	1
Televisão de Alta qualidade	130	1

Tabela 1.1: Características dos serviços de Banda Larga

A característica nuclear do Modo de Transferência Assíncrono é a segmentação de toda a informação a transmitir em células de tamanho fixo, constituídas por um cabeçalho de 5 byte e um campo de dados de 48 byte. Uma das funções do cabeçalho é permitir aos protocolos de comunicação ATM identificar a célula, enquanto os restantes 48 byte estão disponíveis para transportar dados do utilizador. As principais tarefas numa rede ATM são a

comutação, *multiplexagem* e o transporte das células, tarefas que são realizadas de forma independente do conteúdo das células bem como da aplicação que as gerou. A tecnologia de comunicação ATM é baseada no conceito de canais de comunicação virtuais, e a implementação destes canais virtuais é realizada através da identificação existente no cabeçalho das células. A utilização de células de tamanho reduzido e as altas velocidades envolvidas (150Mbit/s) resultam em atrasos pequenos e estáveis na transmissão de dados, o que permite a utilização da tecnologia em variados tipos de aplicações, incluindo a transmissão de dados em tempo real, como voz e imagem em movimento. A comutação e *multiplexagem* dos dados é realizada ao nível da célula, o que permite uma gestão e distribuição flexível da largura de banda.

O recurso à tecnologia ATM (ver capítulo 2) para implementação da RDIS de Banda Larga, preenche o objectivo de a mesma constituir uma rede universal, que suporte diferentes tipos de serviços, e categorias de utilizadores com diferentes necessidades.

O interesse em desenvolver tecnologia nacional no domínio das telecomunicações de banda larga, e contribuir para o "estabelecimento de uma base tecnológica que apoie e incentive o desenvolvimento de novos serviços e aplicações, para as emergentes auto-estradas de informação", motivou o projecto de um TERMINAL ATM, com velocidades de transmissão até aos 155Mbit/s.

Um terminal ATM é uma peça fundamental no processo de implementação das Redes ATM. Não só porque é o equipamento que permite o acesso à rede pelo utilizador, mas também porque importa que as suas características fomentem o acesso "fácil" por um grande número de utilizadores à nova tecnologia - potenciando desta forma a disseminação da mesma.

O terminal foi idealizado de modo a ser implementado sobre a plataforma IBM-PC, uma vez que os computadores da arquitectura Intel x86, constituem actualmente uma plataforma bem implantada, e acessível quer a PMEs quer a utilizadores pessoais, oferecendo um desempenho elevado e dispondo de uma grande base de aplicações instalada.

Na sua essência o sistema é constituído por uma placa adaptadora (a inserir no computador) e respectivo *software* de controlo, oferecendo às aplicações acesso transparente à rede ATM. O núcleo funcional da placa adaptadora é um Circuito Integrado de Aplicação Específica (CIAE). As suas funções básicas são segmentar a informação a transmitir em células ATM, e no sentido inverso, reconstruir a informação no seu formato original. Um terminal ATM que suporte uma interface com a rede à velocidade de 155Mbit/s, dispõe de menos de 3μs para processar cada célula recebida ou a transmitir, e do potencial para movimentar dados a uma taxa superior a 40Mbyte/s. Estas especificações são demasiado exigentes para serem suportadas sem esforço, pelas unidades de processamento das plataformas que se pretendem atingir, e justificam o recurso a *hardware* dedicado para implementação do terminal.

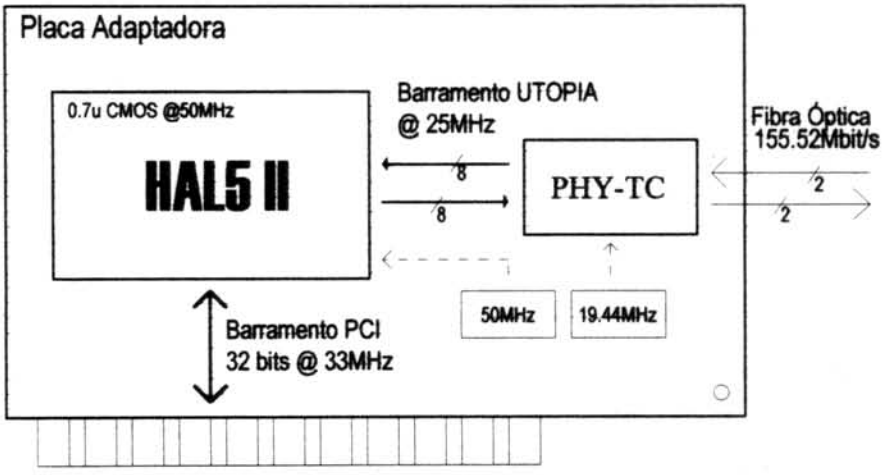


Figura 1.1: A Placa Adaptadora

O circuito integrado que constitui o núcleo da placa adaptadora foi projectado no âmbito deste Curso de Mestrado, e adoptada para a sua denominação a sigla HAL5 II, como acrónimo de "*Hardware for ATM Adaptation Layer 5*". O circuito não foi fabricado, mas procedeu-se à sua simulação funcional, e avaliação das suas características físicas, concluindo um passo importante para a obtenção de um futuro protótipo.

O Modelo de Referência Protocolar (MRP) da RDIS-BL*[23] permite situar a funcionalidade implementada pelo HAL5.

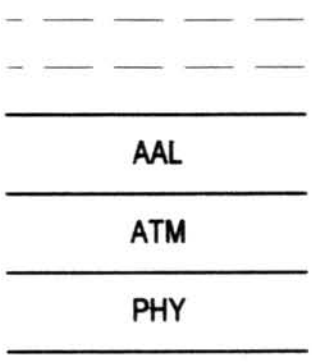


Figura 1.2: MRP da RDIS-BL

A Fig. 1.2 representa de forma simplificada as camadas inferiores do MRP da RDIS-BL. O HAL5 implementa as funções das camadas ATM e AAL (ATM *Adaptation Layer*). As funções do nível físico (camada PHY) estão a cargo do segundo circuito integrado presente na placa adaptadora. As funções a desempenhar pelo HAL5 incluem:

- A manutenção da informação relativa aos canais estabelecidos quer para transmissão quer para recepção: tipo de canal, largura de banda atribuída (na transmissão).
- Fragmentar os pacotes de dados a transmitir, em células ATM. No cabeçalho de cada célula é inserida informação que permite identificar o canal a que pertence a célula. As células são transmitidas de acordo com a taxa de transferência definida para o respectivo canal, e *multiplexadas* num único fluxo de células que é enviado para o nível físico.
- *Desmultiplexar* o fluxo de células recebidas do nível físico e reconstruir os pacotes de dados originais. A informação contida no cabeçalho da célula permite identificar a que canal esta pertence.

* O MRP da RDIS-BL segue o modelo de referência OSI. No entanto, a relação entre as camadas inferiores de ambos os modelos não está ainda completamente definida e clarificada.

- Implementação dos protocolos de comunicação das camadas AAL e ATM. Por exemplo, o cálculo e inserção de um código CRC nos pacotes a transmitir, e verificação do mesmo código de protecção nos pacotes recebidos, é realizado em tempo real pelo circuito - libertando o CPU do sistema hospedeiro desta tarefa pesada.
- Gestão da memória utilizada para armazenamento dos pacotes de informação. A utilização de uma interface PCI permite a utilização da memória principal do sistema hospedeiro para armazenar os pacotes de informação. A gestão da memória é desempenhada pelo circuito sem intervenção do CPU do sistema hospedeiro.

A arquitectura do circuito permite-lhe desempenhar estas tarefas e suportar velocidades até 155Mbit/s simultaneamente na transmissão e na recepção.

A interface entre o nível físico e os níveis superiores, ou seja entre os dois CIs, segue o padrão de barramento UTOPIA (Universal Test and Operations PHY Interface for ATM), definido pelo Fórum ATM[2]. O barramento UTOPIA é uma interface bidireccional, paralela e síncrona, para transferência de células ATM. Ao aderir a este padrão de interface e delegando as funções de acesso ao meio físico para circuitos integrados dedicados a essa função (existentes no mercado), permite-se a utilização de componentes do nível físico de características variadas. A implementação do terminal ATM é desta forma independente do meio físico que suporta a rede, seja ele óptico (155Mbit/s) ou não (26.5Mbit/s, por exemplo).

A interface entre a placa adaptadora e o sistema hospedeiro, segue a norma PCI (*Peripheral Component Interconnect*). Uma primeira consequência desta opção é a vantagem de se poder utilizar a placa adaptadora noutras plataformas que não a IBM-PC, bastando para tal portar o *software* de controlo. Acresce que a utilização de uma interface PCI tem fortes implicações - positivas - na própria arquitectura da placa adaptadora e do terminal ATM. Pelas suas características - grande largura de banda e baixa latência - o barramento PCI torna possível a utilização da memória principal do sistema hospedeiro para armazenamento dos pacotes de dados, eliminando a necessidade de incluir memória local à placa adaptadora para esse fim. Além das vantagens obtidas ao nível arquitectural (ver capítulo 3) promove-se uma grande redução dos custos de implementação da placa adaptadora. O HAL5 leva este

conceito ao extremo colocando as próprias estruturas de dados de controle na memória principal do sistema hospedeiro.

O HAL5 é fruto de um percurso que se iniciou na disciplina de Seminário/Projecto da Licenciatura em Eng. Electrotécnica e de Computadores, no ano lectivo de 93/94, onde se desenvolveu uma primeira versão do circuito[12]. Nessa versão apenas foram implementadas as funções de recepção e reconstrução da informação. O armazenamento dos pacotes de dados era efectuado em memória dedicada ao integrado, solução que implicava a utilização de memória estática - dispendiosa - em quantidades apreciáveis. As interfaces ao nível físico e ao sistema hospedeiro utilizavam um protocolo dedicado, o que limitava a utilização universal do circuito.

O HAL5 II apresenta uma solução de circuito único para a implementação das operações associadas às camadas AAL e ATM nos dois sentidos de transmissão, efectuando todas as transferências de dados de e para o sistema hospedeiro, sem intervenção do processador central através de uma interface padrão (PCI, Bus Master). O circuito integrado permite a implementação de terminais de acesso à rede ATM, com velocidades até 155Mbit/s*.

A fabricação do circuito não foi uma das metas propostas no âmbito deste curso de mestrado. Deste facto resultou uma menor pressão em certas fases do projecto, e algumas simplificações destinadas a enquadrar as fases principais do projecto no quadro temporal disponível. Assim, colocou-se um maior ênfase na concepção da arquitectura do CI e na sua validação. O projecto do HAL5 implicou, além da análise do sistema e das especificações a atingir, a realização das seguintes tarefas:

i) A modelização e descrição do circuito e da sua arquitectura numa linguagem de descrição de *hardware* - Verilog HDL - independente da tecnologia de silício.
ii) O recurso a ferramentas de síntese de alto nível (CADENCE Synergy Synthesis Tool), para mapear a descrição do circuito em componentes da tecnologia utilizada (portas lógicas, elementos de memória, etc).
iii) A simulação funcional dos componentes do circuito integrado.

* Velocidade agregada de 310MBit/s

iv) A utilização de ferramentas de CAD para *Layout* semi-automático (CADENCE CellEnsemble3).

Este documento está estruturado em 6 capítulos. O próximo capítulo é dedicado ao tema das redes ATM, e ao Modo de Transferência Assíncrono. São introduzidos os conceitos técnicos básicos necessários à compreensão da funcionalidade do HAL5.

O capítulo III apresenta o HAL5 II. A arquitectura adoptada, as estruturas de dados de suporte e o funcionamento do CI, são explicados e justificados. O IV capítulo trata dos aspectos associados à implementação do circuito. O capítulo V aborda aspectos relacionados com o projecto digital do HAL5.

O último capítulo conclui com uma apreciação global do trabalho e enquadramento dos resultados.

Ao longo do texto usou-se o *itálico* para os termos em língua estrangeira, bem como para um ou outro neologismo de utilização mais comum na linguagem técnica.

Capítulo 2

Redes ATM

2.1 Aplicações e Serviços de Banda Larga

A universalização na utilização das redes de comunicação é responsável pelo forte investimento verificado actualmente, em tecnologias de redes de comunicação mais velozes e eficientes. Na procura de retirar o maior proveito das estruturas de rede existentes, lançou-se mão de aplicações cada vez mais sofisticadas, muitas vezes baseadas no uso de recursos de som e imagem como as videoconferências, que exigem canais de comunicação rápidos.

A RDIS-BL suporta - com uma gestão eficiente da largura de banda - serviços de transmissão de dados, voz (som), imagem (fotografia e vídeo), bem como aplicações *multimedia* que podem combinar, por exemplo, voz, dados e imagem. As suas características fazem dela uma boa solução tanto para utilizadores particulares como para o meio empresarial e industrial.

Por exemplo, a utilização da videoconferência para comunicação entre pessoas, permite a redução de custos e tempos, ao evitar a deslocação física dos intervenientes. Este processo de comunicação ainda não se encontra acessível em larga escala ao público em geral e às empresas. A utilização da RDIS-BL pode melhorar consideravelmente este cenário, transformando a videoconferência numa ferramenta de uso comum, uma vez que permite o emprego de imagem de alta qualidade e o estabelecimento da comunicação entre os potenciais utilizadores através de interfaces padrão.

Outro característica importante da RDIS-BL é a sua adequação para fornecer com uma boa relação custo/benefício, canais de alta velocidade (*backbone*) para a interligação de

redes locais, com uma gestão flexível da largura de banda. O utilizador particular poderá dispor de serviços que lhe permitem receber informação nos mais variados formatos, como texto, gráficos, som, imagens e filmes; aceder a informação sobre estâncias de férias, eventos culturais, comércio e serviços de vídeo interactivos.

A tabela 2.1 apresenta alguns dos Serviços e Aplicações de Banda Larga [17][22] suportados pela RDIS-BL.

Serviços	Aplicações
VideoTelefonia	Comunicação de voz, imagem e fotografia entre dois pontos. * Tele-Educação * Tele-Comércio * Tele-Publicidade * Tele-Medicina
VideoConferência	Comunicação multi-ponto de voz, imagem, e fotografia. * Tele-Educação * Tele-Medicina * Conferência (empresas)
Transmissão de informação audio e video	* Transporte de sinal de televisão * Video-on-demand
Serviço digital de transmissão de dados a alta velocidade	* Comunicação a alta velocidade – Interligação de LANs – Interligação de MANs * Transmissão de informação vídeo, audio, etc. * CAD interactivo multiponto. * CAM interactivo multiponto
TeleControlo de alta velocidade	Controlo em tempo-real Telemetria Alarmes
TeleFax de alta velocidade	Transferência de texto, imagens, desenho, etc
Serviço de comunicação de imagens de alta resolução	* Industria * Medicina
Video on Demand	* Recreação * Formação e educação à distância
VideoTexto de Banda Larga	VideoTexto com imagem em movimento Tele-*Software* Tele-Comércio Tele-Publicidade Acesso a notícias

Tabela 2.1: Aplicações e serviços de Banda Larga

A necessidade de satisfazer os requisitos de largura de banda dos novos serviços e aplicações foi apenas um dos factores envolvidos no desenvolvimento da RDIS-BL. Outros factores que influenciaram o seu desenvolvimento foram[20]:

- O aparecimento de tecnologias de transmissão, comutação e processamento de sinal de alta velocidade, que permitem taxas de transmissão na ordem do Gbit/s.
- O aumento da capacidade de processamento da informação disponível ao utilizador.
- O desenvolvimento verificado nas aplicações de *software* nas áreas da informática e telecomunicações.
- A necessidade integrar diferentes tipos de serviços e redes de comunicação numa única rede universal, com o objectivo de reduzir custos na área do planeamento, desenvolvimento, implementação e manutenção.
- A necessidade de satisfazer de forma flexível os requisitos do utilizador, em termos de largura de banda, qualidade do serviço, etc.

Ao contrário da redes pré-RDIS, na sua maioria redes dedicadas (por exemplo, telefone ou dados) com largura de banda e flexibilidade limitadas, a RDIS-BL foi concebida como uma rede universal com suporte para diferentes tipos de aplicações e categorias de utilizadores, no que representa um grande salto qualitativo, em relação ao que são hoje a maioria das redes de comunicação utilizadas. A RDIS-BL pretende constituir *a* rede de comunicação do futuro.

2.2 A Tecnologia ATM

De todas as tecnologias de redes de alta velocidade (ATM, FDDI, FastEthernet, 100VG-AnyLAN) a tecnologia ATM foi considerada a única capaz de permitir a transmissão maciça de informação e viabilizar a operação das "auto-estradas da informação". As suas

características permitem-lhe transportar diversos tipos de informação com a mesma eficiência, e foi projectada para trabalhar com dados em que o tempo é um factor crítico e os atrasos inaceitáveis. Está por isso habilitada a tratar de igual modo, dados, voz e imagem, e a integrá-los num único canal de comunicação de dados. A sua flexibilidade permite-lhe estabelecer serviços com diferentes prioridades e características. Outro atributo importante da tecnologia ATM é a sua escalabilidade em termos de distância e velocidade, permitindo que as redes ATM cresçam em velocidade (25, 155, 620 Mbit/s) e dimensão, de acordo com o aumento da sua procura.

Pelo menos cinco factores fundamentam o 'salto' para a tecnologia ATM:

1. Apesar da evolução representada pela RDIS, a mesma é insuficiente para os serviços ditos de banda larga.
2. A integração de voz, dados e aplicações de banda larga de forma economicamente eficiente requer a flexibilidade e largura de banda propiciadas pela tecnologia ATM
3. Para garantir diferentes qualidades de serviço para todos os diferentes tipos de dados a circular na rede, é necessário uma rede que reconheça intrinsecamente o conceito de qualidade de serviço: ATM
4. A rede pública de dados será a médio e longo prazo uma rede ATM.
5. O desenvolvimento de aplicações de emulação de LANs sobre ATM que permite salvaguardar o investimento realizado em aplicações desenvolvidas sobre redes Ethernet e Token Ring.

A iminente convergência de meios, aplicações e tecnologia faz prever que a curto prazo a tecnologia ATM atingirá uma importância económica assinalável. A procura de produtos e serviços ATM está a acelerar em direcção à massa crítica: em 1994 o valor do mercado global ATM excedeu os 100 milhões de dólares e as previsões apontam para a marca dos 2000 milhões de dólares em 1998 [4]!

2.3 Aspectos Técnicos Básicos

2.3.1 As Células ATM

Os atributos e potencialidades da tecnologia ATM assentam no seu elemento basilar: a célula ATM.

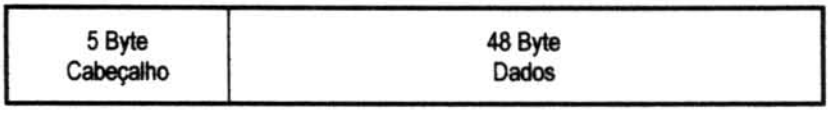


Figura 2.1: Célula ATM

No modo transferência assíncrono toda a informação a transferir é segmentada em blocos de dimensão fixa denominados células. As células ATM são constituídas por um cabeçalho de 5 byte que transporta informação destinada aos protocolos de rede ATM, e um campo de dados de 48 byte disponível para transportar dados do utilizador.

O ATM funciona na base da comutação de circuitos (ver 2.3.2). As células provenientes de diferentes comunicações individuais são *multiplexadas* e transportadas num único fluxo de dados. Parte da informação contida no cabeçalho permite identificar a célula e o canal de comunicação a que esta pertence.

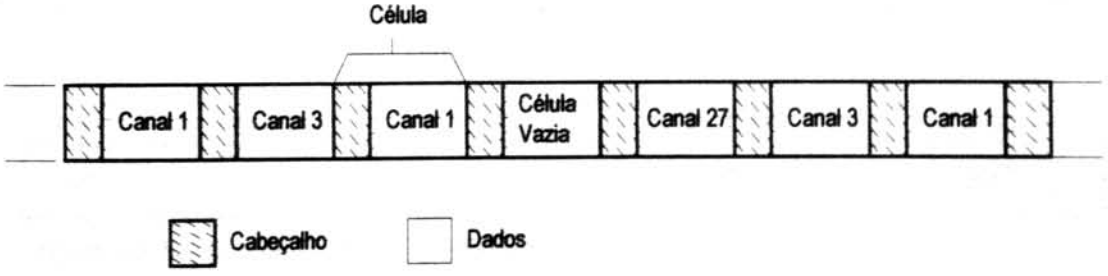


Figura 2.2: Modo de Transferência Assíncrono

O termo «assíncrono» na designação da tecnologia descreve o facto de na transmissão de diferentes canais *multiplexados*, as células pertencentes a um mesmo canal apresentarem um

padrão irregular, reflexo da sua geração em função do débito da aplicação que está a utilizar o canal de comunicação (Fig. 2.2). A flexibilidade e eficiência na gestão da largura de banda é assim superior aos modos de transferência síncronos, onde de uma forma rígida é atribuída uma fatia numa moldura temporal a cada canal, independentemente das características do seu débito de informação e de a mesma ser ou não utilizada.

A largura de banda atribuída a cada conexão é determinada em função das suas necessidades particulares e adaptável dinamicamente, permitindo uma gestão global eficiente dos recursos da rede. A *multiplexagem* dos dados ao nível da célula está na base deste desempenho. O processamento das células é independente do que estas contem ou da aplicação que as gerou, o que permite que o mesmo equipamento de rede atenda a serviços com diferentes débitos de informação, desde os mais reduzidos aos mais elevados.

O modo de transferência assíncrono reúne qualidades das técnicas de comutação de circuitos e das técnicas de comutação de pacotes. Depois de estabelecido o canal de comunicação, o *overhead* envolvido no processamento das células é reduzido e a sua pequena dimensão conjugada com as velocidades envolvidas (ver 2.3.3) permitem um atraso pequeno e quase invariável no transporte da informação (essencial no transporte de informação *time-dependent* como o audio e o vídeo). Por outro lado as pequenas células de dados assemelham-se a pacotes de dados a circular na rede, e o seu tratamento 'assíncrono' permite uma gestão eficiente e flexível da taxa de transferência atribuída a cada canal de comunicação.

A informação contida no cabeçalho das células destina aos protocolos de comunicação da camada ATM (ver 2.4.4). Parte dessa informação destina-se ao *routing* das células através da rede. A Fig. 2.3 apresenta o formato da informação contida no cabeçalho.

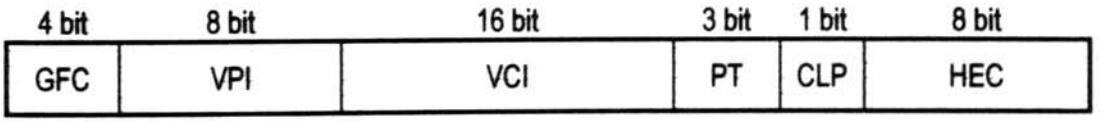


Figura 2.3: Formato do cabeçalho da célula ATM

O formato apresentado é o utilizado na interface utilizador-rede (UNI) e difere do formato utilizado nas interfaces nos nós da rede (NNI). Nos NNIs o campo GFC desaparece

e é ocupado pelo campo VPI estendendo-se este para os 12 bit. A função dos campos que constituem o cabeçalho é a seguinte:

- Generic Flow Protocol: Este mecanismo auxilia o controlo de tráfego na interface utilizador-rede, especialmente em situações temporárias de *overload* que se possam verificar na rede do utilizador. Esta informação está presente em todas as células válidas e o seu valor por omissão, quando o mecanismo não é utilizado, é 0000. Os procedimentos para a sua utilização ainda não estão completamente definidos[21].
- Virtual Path/Circuit Identifier: Informação para *routing* da célula (ver 2.3.2).
- Payload Type: Permite identificar o tipo de dados contido no campo de informação da célula, distinguindo entre dados do utilizador e dados da rede. Para as células que transportam informação do utilizador (PTI=0xx)*: o bit CI (congestion Indication) pode ser alterado por qualquer elemento da rede para indicar uma situação de saturação; o bit AUU (ATM-Layer-user-to-ATM-Layer-user) é utilizado pelo protocolo AAL5 (ver 2.4.5).
- Cell Loss Priority: Permite definir dois tipos de prioridades para o transporte das células. Se CLP for '1' a célula pode ser 'descartada', se o estado da rede assim o recomendar. Se CLP for '0' a célula é considerada de prioridade superior e a rede deve garantir os recursos suficientes para o seu transporte.
- Header Error Control: Este campo não é utilizado pelos protocolos da camada ATM. Trata-se de um código de protecção da informação do cabeçalho, e é mantido pela camada do nível físico. Permite detectar e corrigir erros no cabeçalho da célula. Os erros *single-bit* são corrigidos enquanto os erros multi-bit provocam a eliminação da célula ao nível da camada física.

2.3.2 Os Canais Virtuais

A rede ATM é baseada na comutação de circuitos (*connection oriented*). A função dos campos VCI e VPI, presentes no cabeçalho da célula, consiste em identificar a que canal de

* Nota: nas células em que o PTI é da forma 0xx, o formato do campo PTI é o seguinte: [0][CI][AUU].

comunicação pertence a célula. Os canais são designados virtuais porque, de facto, ao nível físico o que se passa é que as células dos vários canais de comunicação, são *multiplexadas* num único fluxo de células no canal de transmissão. Em circunstâncias normais de funcionamento o ATM garante a integridade da sequência das células, ou seja, não se dá a situação de uma célula de um determinado canal de comunicação, ser ultrapassada por outra célula, do mesmo canal de comunicação, gerada posteriormente.



Figura 2.4: Circuitos Virtuais no ATM

A Fig. 2.4 representa a hierarquia de transmissão do ATM. Por canal de transmissão subentende-se o troço físico entre dois elementos da rede onde se proceda à segmentação e reconstrução das células. As funções de transporte da camada ATM implementam dois níveis de transporte das células: o nível do caminho virtual (*Virtual Path*), e o nível do canal virtual (*Virtual Channel*) [19]:

Virtual Channel: 'Um conceito utilizado para definir o transporte unidirecional de células ATM, associadas por um identificador comum único'. O identificador em questão é o campo VCI no cabeçalho da célula.

Virtual Path: 'Um conceito utilizado para definir o transporte unidirecional de células pertencentes a VC associados por um identificador comum único'. O identificador em questão é o campo VPI no cabeçalho da célula.

Resumindo, um canal de transmissão pode conter vários *Virtual path* e cada um destes agrupa vários *Virtual Channel*. O troço da rede entre dois pontos onde o VCI se mantém

constante, ou seja entre o ponto onde é gerado e o ponto onde é removido ou comutado designa-se *Virtual Channel Link*. De igual forma se define o *Virtual Path Link*. Uma concatenação de VPLs designa-se por *Virtual Channel connection* (VCC) e uma concatenação de VCLs por *Virtual Path connection* (VPC).

A comutação na rede faz-se por comutação de 'canais virtuais' e 'caminhos virtuais'. A comutação VP dá-se no termino dos VPLs e implica o mapeamento do identificador VPI das células de acordo com o destino da conexão. O identificador VCI mantém-se constante. A comutação VC dá-se no termino dos VCLs e consequentemente dos VPLs (uma vez que o canal está associado ao caminho).

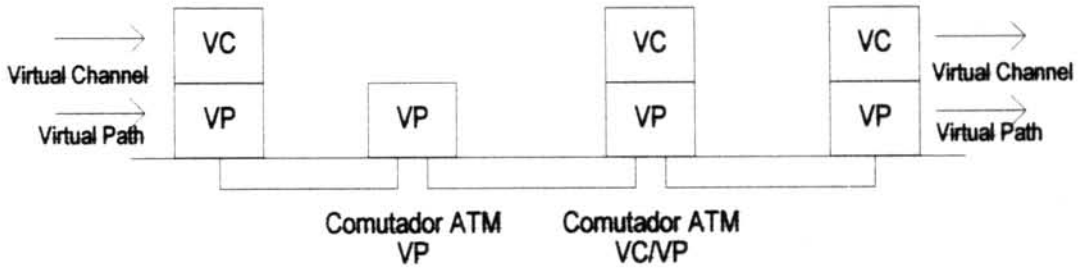


Figura 2.5: Conexões ATM

As conexões VC (VCC) e as conexões VP (VPC) constituem circuitos de transporte de informação na rede, onde todas as células duma mesma conexão seguem o mesmo percurso na rede, e a integridade da sua sequência é preservada. As conexões VPV/VCC permitem estabelecer canais de comunicação, entre utilizadores, entre o utilizador e a rede, e entre redes.

A cada conexão estabelecida na rede é dedicada parte da largura de banda disponível, durante todo o período da sessão de comunicação. Desta forma garante-se que a largura de banda negociada estará sempre disponível quando for necessária. Esta característica eleva o ATM além de tecnologias como o FDDI e Fast Ethernet, as quais também oferecem 'Banda Larga'. As vantagens revelam-se ao nível da gestão da largura de banda, e dos diferentes tipos de tráfego a transportar na rede.

Por exemplo, é possível estabelecer um canal para troca de correio-electrónico e documentos de texto, ao qual é atribuída uma pequena quantidade fixa (ou variável) de largura

de banda, enquanto a outro canal, usado para videoconferência, é atribuída uma largura de banda adequada (3Mbit/s) e que se mantém fixa durante a sessão de comunicação. A largura de banda é dedicada a cada canal, garantindo a qualidade da comunicação independentemente do resto da rede (ao contrário das redes não orientadas à conexão, onde, numa situação de saturação, todos os utilizadores e aplicações são afectados por igual).

A definição de canais e 'caminhos' virtuais é particularmente eficaz. Por exemplo, um VPC estabelecido entre dois utilizadores, fornece-lhes uma via de comunicação cuja organização está inteiramente a seu cargo. Por exemplo para interligação de duas LANs, permitindo-lhes o estabelecimento de canais adequados aos diferentes tipos de tráfego.

Esta facilidade na gestão e atribuição dos recursos da rede, traduz-se no conceito de Qualidade de Serviço (QOS). No ATM é possível indicar, além da largura de banda, a qualidade de serviço pretendida:

ABR: *Available Bit Rate*. A rede tenta providenciar a largura de banda necessária, mas não é garantido que esta esteja sempre disponível, podendo o seu valor variar durante a sessão de comunicação.

UBR: *Unspecified Bit Rate*. A rede não garante qualquer largura de banda. A largura de banda atribuída durante a sessão de comunicação varia em função da largura de banda não utilizada a cada momento.

CBR: *Constant Bit Rate*. A rede garante que a largura de banda atribuída se mantém constante durante toda a sessão de comunicação.

Por exemplo, o serviço do tipo CBR é particularmente adequado para sessões de videoconferência, enquanto para a transferência de grandes ficheiros de texto será suficiente uma qualidade de serviço do tipo ABR.

2.3.3 As Funções de Operação e Manutenção

A Recomendação ITU-T M.60 define manutenção como:

'A combinação de acções técnicas e administrativas, incluindo acções de supervisão, com a intenção de manter, ou repor, um elemento da rede num estado em que possa realizar as funções que lhe estão atribuídas'.

A tabela 2.2 [25] apresenta as principais funções de '*Operation and Maintenance*' (OAM).

Acção	Descrição
Monitorização do Desempenho	O normal funcionamento dos elementos da rede é verificado através de um controlo periódico ou contínuo. Como resultado é gerada informação de monitorização.
Detecção de Falhas e Defeitos	A detecção de falhas, de facto ou prováveis, é realizada por verificação continua ou periódica dos elementos da rede. Como resultado é gerada informação sobre falhas ou defeitos, ou é activado um alarme.
Protecção do Sistema	O efeito provocado pela falha de um elemento da rede é minorado através do reajustamento dos restantes elementos da rede. Como resultado o elemento em falha é excluído do sistema.
Localização de Faltas	Determinação de falhas através de testes, internos ou externos, no caso da informação sobre falhas for insuficiente.
Informação sobre falhas e desempenho	A informação é enviada às entidades administradoras. Fornece também resposta a pedidos de informação.

Tabela 2.2: Funções OAM

Na rede ATM as funções de OAM são realizadas a cinco níveis hierárquicos associados com as camadas ATM e Física do MRP. A cada nível está associado um fluxo de informação OAM.

Nível	Fluxo	
Nível da secção repetidora	F1	Camada Física
Nível da secção Digital	F2	
Nível do Canal de Transmissão	F3	
Nível do 'Virtual Path'	F4	Camada ATM
Nível do 'Virtual Channel'	F5	

Tabela 2.3 Fluxos OAM

Ao nível da camada física estão definidos três fluxos, F1, F2, e F3, para desempenho das funções de OAM (ver figura 2.3). A informação pode ser transportada em células com cabeçalhos reservados (que não são enviadas para a camada ATM) e/ou inserida na transmissão de acordo com o esquema adoptado para a interface física (interface *cell-based* ou SDH, ver 2.3.4).

Os fluxos F4 e F5 referem-se às funções OAM na camada ATM. As funções OAM a este nível incluem (tabela 2.4) [17]:

Nível	Função	Fluxo
'Virtual Path'	Monitorização da disponibilidade do 'Caminho'.	F4
	Monitorização de Desempenho.	
'Virtual Channel'	Monitorização da disponibilidade do Canal.	F5
	Monitorização de Desempenho.	

Tabela 2.4 Funções OAM na camada ATM

As células OAM F4 usam os VCIs reservados 3 e 4. As células OAM F5 são identificadas pelo seu PTI=10x. Ambos os tipos de células contêm no seu campo de dados informação referente a funções OAM.

Os fluxos de dados de OAM são bidireccionais. Por exemplo, a monitorização do estado de uma VPC pode ser realizado através do envio de células de um ponto da rede para outro, que as reflecte para o ponto de origem para análise e obtenção de informação sobre o estado da conexão.

2.3.4 O Meio Físico

O desenvolvimento verificado ao nível da tecnologia óptica para transmissão de sinal, com o aparecimento de equipamento de alto desempenho e economicamente acessível, constitui um dos motores da RDIS-BL e do ATM.

A transmissão óptica de sinal é caracterizada por [17]:

- Grandes distâncias entre repetidores.
- Grande largura de banda, na ordem das várias centenas de Mbit/s.
- Grande flexibilidade e maneabilidade da fibra óptica.
- Imunidade a interferências electromagnéticas.
- Baixa probabilidade de erro na transmissão.
- Inexistência de *crosstalk* entre as fibras ópticas.
- Maior imunidade a acessos ilegais ao meio.

A tecnologia actual permite larguras de banda até 2.5Gbit/s. Contudo, por motivos económicos, a interface típica utilizador-rede está projectada para 155Mbit/s. Está também definida outra interface para taxas de transferência na ordem dos 622Mbit/s. Ainda por motivos económicos, definiram-se outras interfaces (não necessariamente ópticas) com taxas de transferência inferiores - desde 2Mbit/s até 26.5Mbit/s. A utilização de tecnologia com este grau de fiabilidade para a transmissão de sinal, reflecte-se na simplificação da estrutura da rede, por exemplo, na simplificação dos protocolos de comunicação ao nível da correcção de erros.

As interfaces são simétricas em termos de taxa de transferência, ou seja 155Mbit/s no sentido do utilizador para a rede e 155Mbit/s no sentido da rede para o utilizador. A possibilidade de interfaces assimétricas está prevista para utilizadores com tráfego muito superior numa das direcções. Por exemplo, 622Mbit/s no sentido descendente é uma situação plausível para um serviço de distribuição de vários canais de televisão a um utilizador residencial, ao qual, para interacção com o serviço, lhe bastam 155Mbit/s no sentido ascendente.

Os valores exactos das taxas de transferencia definidas para as interfaces com a rede são 155.520Mbit/s e 622.080Mbit/s. Estes dois valores correspondem aos dois valores inferiores utilizados na tecnologia de transporte de dados SDH (*Synchronous Digital Hierarchy*). A SDH é a estrutura preferida para as interfaces RDIS-BL, por estar particularmente adaptada ao transporte de variados tipos de sinal, tal como RDIS ou células ATM. O SDH já é utilizado na implementação de um número considerável de redes (ATM ou não).

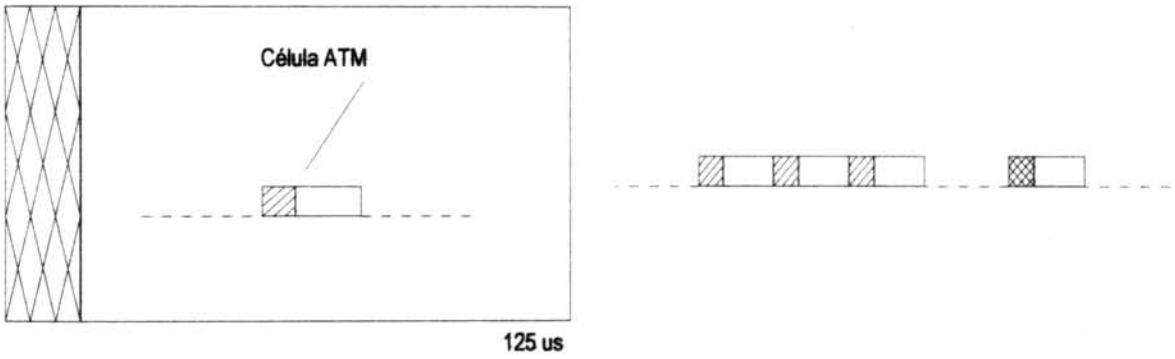


Figura 2.6: Estruturas da Interface ATM

Na SDH a informação a transmitir é estruturada em molduras (*frames*) de 2430 byte, dispostos em 9 linhas e 270 colunas. As molduras são enviadas à frequência de 8KHz (2430 × 8KHz = 155.520Mbit/s). As células ATM são transportadas no 'interior' da moldura utilizando 9 linhas e 260 colunas. As restantes 10 colunas constituem o *overhead* necessário à implementação dos protocolos SDH e do nível físico ATM (funções OAM, etc.).

Uma das suas desvantagens é o *overhead* acima do que seria necessário numa interface estruturada em células ATM. Este facto levou à definição de uma interface completamente *cell-based* (ver figura 2.6). Apesar de não haver nenhuma restrição nesse sentido, adoptaram-se as mesmas taxas de transferência da estrutura SDH.

As taxas de transferência do SDH, descontando o *overhead*, são 149.760Mbit/s e 599.040Mbit/s. Estas são as taxas de transferencia disponíveis para as células de dados do utilizador, células de sinalização, células OAM da camada ATM e superiores. Nas interfaces de estrutura *cell-based*, a camada física insere uma célula própria a cada 26 células. Deste modo (26/27) garante-se o mesmo valor para as taxas de transferência independentemente da estrutura da interface.

2.4 O Terminal ATM

2.4.1 O Modelo de Referência Protocolar da RDIS-BL

Actualmente a organização das várias funções e procedimentos associados aos sistemas de comunicação, é disposta numa estrutura modular por camadas. As funções de cada camada e as interacções entre cada camada são descritas no Modelo de Referencia Protocolar. Este é o conceito incorporado no modelo OSI (*Open System Interconnection*).

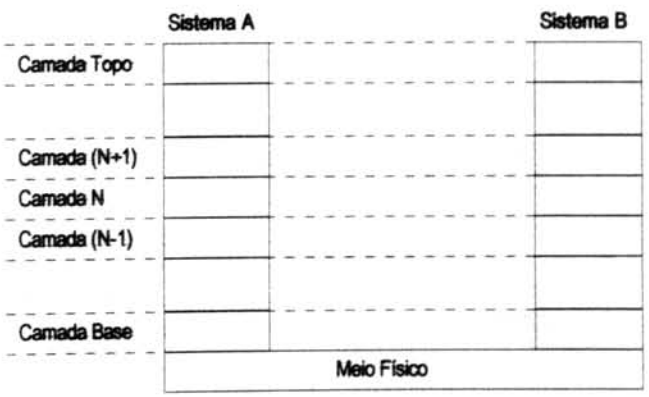


Figura 2.7: O modelo OSI

De acordo com o modelo OSI um 'sistema aberto' é descrito como um conjunto de camadas dispostas numa estrutura vertical. A camada do nível N, constituída por uma ou mais entidades N, só interage com as camadas imediatamente acima ou abaixo. Uma entidade N desempenha funções ao nível da camada N. A comunicação entre camadas N de diferentes sistemas é realizada através dos protocolos de comunicação do nível N. A unidade de informação utilizada pelo protocolo do nível N designa-se por N-PDU (*Packet Data Unit*). Duas entidades N comunicam entre si, através dos serviços fornecidos pelas camadas inferiores adjacentes. Os serviços da camada N são oferecidos à camada (N+1), sendo que o ponto onde os serviços do nível N são fornecidos à camada superior adjacente se designa por N-SAP (*Service Access Point*). As figuras 2.8 e 2.9 descrevem este conceito de serviço de valor acrescentado, associado ao conceito OSI, e a relação entre os vários tipos de dados utilizados por cada nível hierárquico.

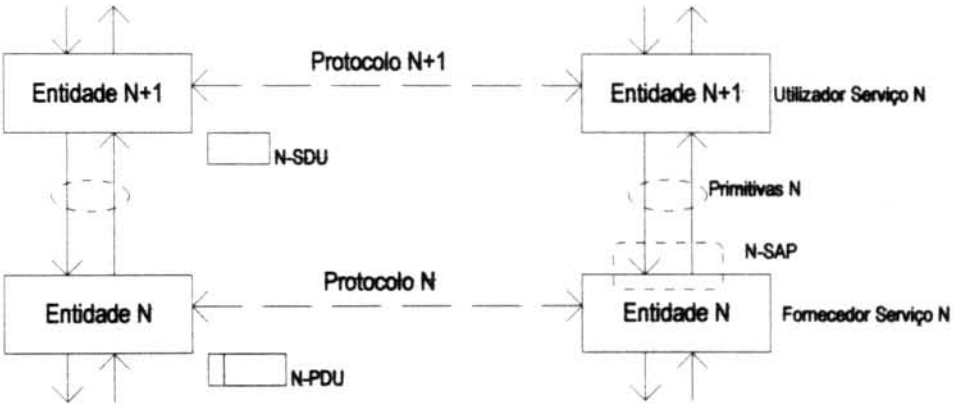


Figura 2.8: Conceito de Serviço do modelo OSI

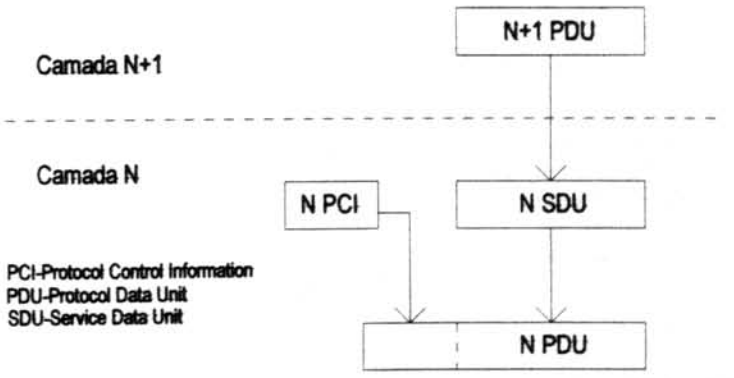


Figura 2.9: Fluxo de informação do modelo OSI

O modelo OSI define sete camadas para a organização das diferentes funções e procedimentos associados aos sistemas de comunicação; desde a camada inferior, associada ao transporte físico da informação, até à camada de topo, que efectivamente oferece os recursos da rede ao utilizador. O modelo de referência protocolar da RDIS-BL segue a abordagem do modelo OSI, contudo, a relação entre as camadas inferiores de ambos os modelos ainda não está completamente clarificada. A figura 2.10 representa a estrutura das camadas inferiores do modelo de referência protocolar, na implementação sobre o Modo de Transferência Assíncrono.

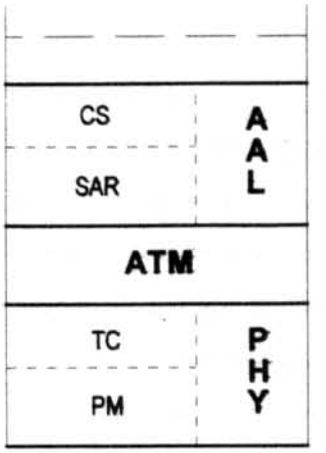


Figura 2.10: O MRP do ATM

A figura 2.11 apresenta de forma simplificada o funcionamento de uma rede ATM, e as funções desempenhadas pelas camadas AAL, ATM e Física.

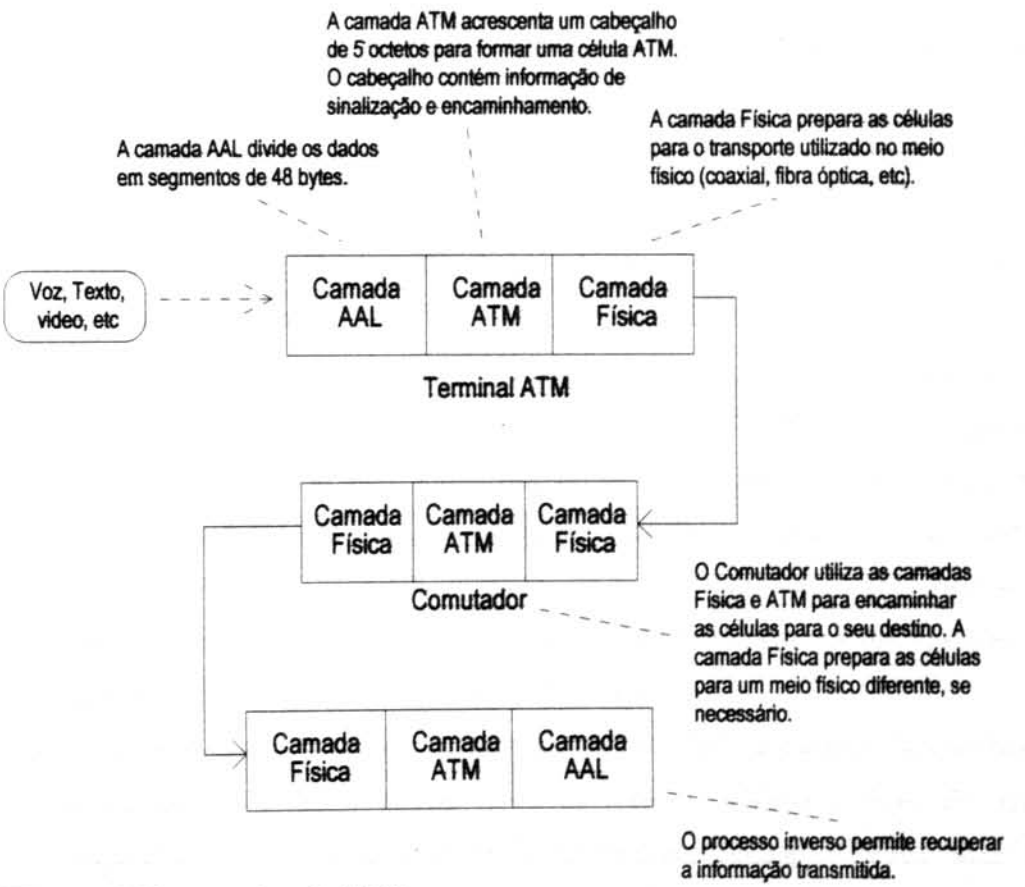


Figura 2.11: Estrutura da rede ATM

2.4.2 A Camada Física

A camada Física está organizada em duas sub-camadas. A sub-camada PM (*Physical Medium*), ao nível mais baixo, desempenha apenas as funções estritamente relacionadas com o meio físico. É responsável pela capacidade de transmissão do fluxo de bit, incluindo o alinhamento e sincronismo do fluxo de dados ao nível do bit. A utilização de códigos de linha e, se necessário, a conversão eléctrico/óptica estão a cargo desta sub-camada. Resumindo, a sua principal função é a geração e recepção de formas de onda (sinais) adequados ao meio físico utilizado, inserção e extracção de informação de sincronismo do fluxo de bit e utilização de códigos de linhas adequados (se necessário).

A sub-camada TC (*Transmission Convergence*) desempenha cinco funções, sendo que a função básica é a geração e recuperação da moldura (*frame*) de transmissão. As restantes funções são:

- Adaptação da moldura de transmissão: esta função consiste em adaptar o fluxo de células à capacidade da estrutura de transporte utilizada na interface. A estrutura de transporte pode ser a própria célula (interface *cell-based*, não é utilizado qualquer envelope), ou um envelope SDH.
- Delineação das células: esta função permite identificar as fronteiras de cada célula no fluxo de dados.
- Implementação do HEC: Este mecanismo de protecção do cabeçalho da célula permite detectar erros no cabeçalho da célula. Os erros do tipo *single-bit* são corrigidos, enquanto as células com erros no cabeçalho do tipo *multi-bit* são eliminadas. Para cada célula a transmitir é calculado o valor do HEC sobre os primeiros 4 byte do cabeçalho, e, na recepção, este campo é utilizado para verificar da correcção do cabeçalho. A função de delineação das células apoia-se neste mecanismo para obter o sincronismo de células.
- Desacoplamento da taxa de transferência das células: Na transmissão esta função é responsável por inserir células vazias de forma a adaptar o fluxo de células à capacidade de transporte da interface. Na recepção elimina as células vazias. Só as células atribuídas são enviadas para a camada ATM.

As funções da camada Física não são desempenhadas pelo HAL5. As motivações para a utilização de outro circuito integrado para o desempenho destas funções incluem:

- Simplificar o HAL5. A inclusão das funções da camada física no HAL5 acarretaria uma complexidade desnecessária ao circuito.
- Universalidade. Ao implementar apenas as camadas ATM e AAL, o HAL5 pode ser utilizado na construção de terminais ATM para diferentes meios físicos (óptico, coaxial, etc.) e para diferentes velocidades de transmissão (até 155Mbit/s).
- Interface UTOPIA. A existência de uma interface padrão para ligação a circuitos que implementem as funções da camada física, permite a utilização de um leque variado de opções entre os circuitos disponíveis no mercado.

2.4.3 A Camada ATM

A camada ATM utiliza os serviços da camada Física. As suas funções são independentes do meio físico utilizado. São 4 as principais funções desta camada.

- Na transmissão, as células provenientes dos vários VPs e VCs são *multiplexadas* num único fluxo de células. O fluxo composto é, normalmente, um fluxo não contínuo. Esta função designa-se por 'multiplexagem de células'. Na recepção, a função de 'desmultiplexagem de células' separa as células recebidas no fluxo composto pelos VPs e VCs apropriados.
- Nos nós de comutação da rede a função de 'mapeamento de VPI e VCI' efectua o encaminhamento adequado dos canais de comunicação. Nos nós de comutação VP o valor do campo VPI em cada célula recebida é mapeado no novo valor para transmissão. Nos nós de comutação VC ambos os valores VCI e VPI são mapeados em novos valores. Esta função de *routing* não está necessariamente presente num equipamento terminal.
- Nos pontos terminais da rede está presente a função de geração/extracção de cabeçalhos. Na transmissão, após receber o campo de dados (48 byte) da camada

AAL, é acrescentado o cabeçalho da célula (à excepção do campo HEC) com a informação apropriada ao encaminhamento da célula (VPI/VCI). Na recepção, o cabeçalho é extraído e só o campo de dados é enviado para a camada superior.

- Nos pontos terminais da rede a função GFC (*General Flow Control*) permite minorar o efeito de situações de saturação no UNI.

As funções desempenhadas pela camada ATM apoiam-se na informação contida no cabeçalho de cada célula (ver 2.3.1), e estão na base da capacidade de estabelecimento dos canais de comunicação no ATM. Uma conexão, na camada ATM, é uma concatenação de vários segmentos (*links*) por forma a obter um meio de comunicação entre dois pontos de acesso.

O campo VPI é utilizado para distinguir entre os diferentes segmentos VP *multiplexados* no fluxo composto de células, enviadas para a (ou recebidas da) camada Física. Os diferentes segmentos VC em cada VPC são identificados pelos seus VCIs. Dois VCs pertencentes a diferentes VPs, na mesma interface, podem ter valores de VCI idênticos, sendo necessário ambos os valores VPI e VCI para identificar o VC em questão.

2.4.4 A Camada AAL

A camada AAL está dividida nas sub-camadas SAR (*Segmentation And Reassembly*) e CS (*Convergence Sub-layer*). A camada AAL situa-se entre a camada ATM e as camadas superiores. A sua função principal é adaptar os serviços fornecidos pela camada ATM às necessidades das camadas superiores, na forma de um valor acrescentado - característico do modelo OSI. Os pacotes de dados (PDUs) das camadas superiores são mapeados no campo de dados das células ATM. O estabelecimento da comunicação entre entidades AAL está na base das funções da camada AAL - de acordo com o modelo OSI.

As funções essenciais da sub-camada SAR são, na transmissão, a segmentação dos PDUs recebidos da camada superior em blocos adaptados à dimensão do campo de informação das células ATM (48 byte) e, na recepção, a reconstrução dos PDUs. As funções da sub-camada CS são dependentes do serviço específico a ser prestado pela camada AAL às

camadas superiores. Em algumas aplicações pode não ser necessária a utilização de qualquer uma destas sub-camadas, situação em que uma ou ambas podem ser nulas.

As funções da sub-camada CS e dos protocolos AAL são definidos em função do tipo de serviço a prestar no ponto de acesso (AAL-SAP). A classificação dos tipos de serviço é feita em função do tipo de conexão, taxa de transferência e dependência temporal.

A tabela 2.5 descreve as 4 classes definidas.

	Classe A	Classe B	Classe C	Classe D
Relação temporal entre origem e destino	Necessária		Não necessária	
Taxa de transferência	Constante	Variável		
Tipo de comunicação	Orientada à conexão			Não orientada à conexão

Tabela 2.5: Classes de Serviço

As diferentes classes definidas prestam-se a diferentes tipos de serviços. Alguns exemplos:

- Classe A: Emulação de circuitos (por exemplo, transporte de um sinal a 2Mbit/s, videoconferência, etc.
- Classe B: Audio e vídeo de qualidade variável.
- Classe C e D: Comunicação de dados, etc.

Estão definidos vários protocolos AAL. Nomeadamente, AAL0, AAL1, AAL2, AAL3/4 e AAL5. Cada protocolo dispõe de camadas SAR e CS específicas. Estes protocolos prestam-se à implementação das classes de serviços definidas. A relação entre os protocolos AAL e as diferentes classes de serviços é a seguinte: AAL1 - Classe A, AAL2 - Classe B, AAL3/4 - Classes C e D. Contudo, esta relação entre os protocolos e as classes de serviços não é imperativa.

O protocolo AAL0 consiste numa camada AAL em que as sub-camadas SAR e CS são transparentes. A camada AAL não implementa nenhuma funcionalidade e o conteúdo das células é enviado directamente para os níveis superiores.

O protocolo AAL5 foi originalmente definido para serviços com taxas de transferências variáveis e sem relação temporal entre a origem e o destino da comunicação. Este protocolo assemelha-se ao protocolo AAL3/4 e foi criado com o intuito de obter um protocolo com um *overhead* reduzido. O protocolo AAL5 pretende implementar uma camada 'AAL simples e eficiente'.

O HAL5 suporta, em *hardware*, os protocolos AAL5 e AAL0.

2.4.4.1 O Protocolo AAL5

O protocolo AAL5 permite implementar uma camada AAL simples e eficiente. Este facto está na base da sua importância crescente e justifica a sua utilização em serviços que não os para que foi pensado inicialmente. Por exemplo, está em estudo a transmissão de vídeo MPEG2 sobre AAL5 quando este tipo de serviço (classe B) imporia, em princípio, a utilização do protocolo AAL2. Acresce que o protocolo AAL5, é o protocolo escolhido para implementar as funções de sinalização, '*frame relay*' e TCP/IP na rede ATM e, potencialmente, o protocolo a utilizar em futuras aplicações de transmissão de dados na rede ATM.

O protocolo AAL5 define ambas as sub-camadas SAR e CS. A sub-camada SAR aceita SDUs com um comprimento múltiplo de 48 byte, da sub-camada CS. Não é acrescentado qualquer campo ou informação adicional ao SDU. Apenas é realizada a segmentação, e no sentido inverso, a reconstrução do SDU. O início e o fim do SAR-PDU é determinado através do bit AUU existente no cabeçalho da célula ATM. Este esquema contraria o princípio de modularidade do MRP mas foi adoptado por uma questão de simplicidade do protocolo.

A sub-camada CS fornece aos níveis superiores a capacidade de transporte de pacotes de dados de comprimento variável até 65536 byte. Esta sub-camada acrescenta ao pacote de dados a transmitir um bloco com informação de controlo do protocolo AAL5.A figura 2.10 descreve o fluxo de dados entre as várias camadas.

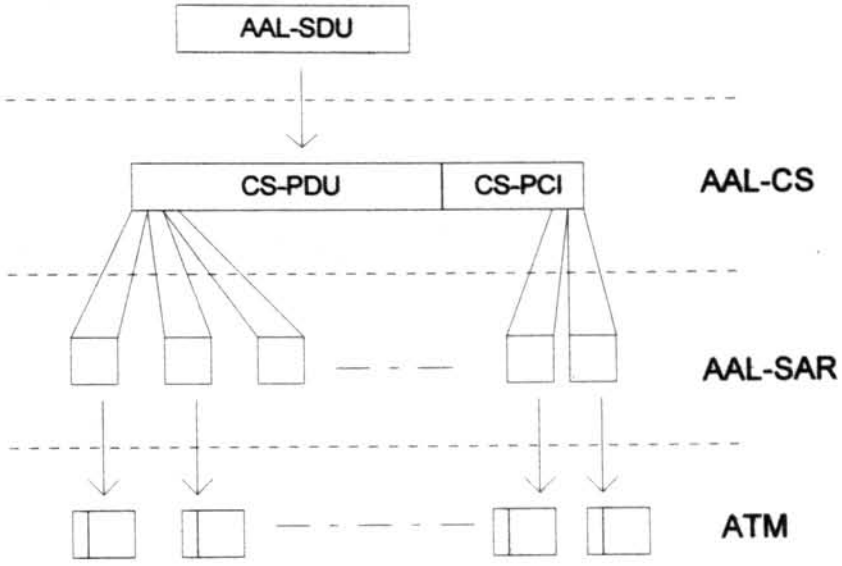


Figura 2.10

No protocolo AAL5, a sub-camada CS acrescenta a seguinte informação ao AAL-SDU:

<48 byte	8 bit	8 bit	16 bit	32 bit
PAD	UU	CPI	Comprimento	CRC32

O campo PAD é de comprimento variável e permite alinhar o CS-PDU a 48 byte. O Campo UU (utilizador-a-utilizador) permite a comunicação de informação adicional ponto-a-ponto. O campo CPI, presentemente, tem apenas a função de alinhar o CS-PDU a 64 bit. Um campo de 16 bit indica o comprimento do pacote transportado no CS-PDU. O campo CRC32 protege toda a informação (dados e informação de protocolo) e permite detectar erros no CS-PDU.

Capítulo 3

Arquitectura do HAL5 II

3.1 Introdução

O HAL5 desempenha as funções da camada ATM, da sub-camada AAL-SAR e assiste o *software* de comunicação no desempenho das funções da sub-camada AAL-CS. A memória principal do sistema hospedeiro é utilizada para o armazenamento dos pacotes de dados (CS-PDU). Em traços largos, a funcionalidade do circuito consiste em:

- Para cada célula recebida do nível físico é consultada uma tabela com a descrição das conexões activas. Esta tabela também se encontra na memória principal e é preenchida pelo *software* de controlo da placa, de acordo com as instruções do *software* que implementa as camadas superiores do sistema de comunicação. A tabela é indexada com o identificador de canal existente no cabeçalho da célula. A informação obtida permite determinar o canal a que pertence a célula e a localização na memória principal do respectivo pacote em reconstrução. O campo de dados da célula é então transferido para a memória principal através da interface PCI, sem qualquer intervenção do CPU. Células com cabeçalhos pré-definidos (células OAM F4 e F5, por exemplo) são automaticamente enviadas para um *buffer* próprio.
- Os pacotes de dados a transmitir são colocados em fila de espera na memória principal do sistema. Os canais de comunicação são servidos de acordo com uma tabela que determina as suas características (taxa de transferência, etc). Os pacotes são fragmentados em blocos de 48 byte aos quais é adicionado o cabeçalho com a

informação de controlo da camada ATM (VPI/VCI, etc). A célula assim formada é então enviada para o circuito que implementa o nível físico.

A escolha da localização dos *buffers* para armazenamento dos pacotes a transmitir e a receber, constitui uma opção de fundo com implicações na arquitectura do CI, no seu desempenho e no custo de implementação do sistema.

Uma das hipóteses consiste em utilizar memória local, dedicada ao circuito. Esta solução apresenta algumas desvantagens (entre elas o aumento dos custos de implementação do sistema), mas é a opção inevitável quando não se dispõe de um barramento de comunicação com o sistema hospedeiro, com uma largura de banda sustentada capaz de suportar o trafego ATM. Os CS-PDUs têm que ser copiados duas vezes (para a memória local e da memória local para a memória principal) o que aumenta a latência dos protocolos da rede. Por outro lado a quantidade limitada (necessariamente) de memória local implica um limite no número de CS-PDUs que podem ser mantidos simultaneamente. Torna-se necessária a utilização de esquemas complexos de gestão de memória para permitir uma aproveitamento óptimo da memória disponível. Por exemplo, o HAL5 I [12] utilizava uma unidade de gestão de memória (MMU), *in-chip*, para permitir o acesso linear à memória local, disposta num modelo de acesso paginado.

Nos sistemas a que se destina o terminal ATM, actualmente, é comum a existência de um barramento PCI, bem como de quantidades de memória em valores apreciáveis. As características do barramento PCI - baixa latência e elevada largura de banda - permitem o acesso rápido e eficaz a memória principal. Nesta situação é viável a utilização da memória principal do sistema para armazenamento dos CS-PDUs (a segunda hipótese). Desta forma remove-se o condicionalismo quanto ao numero de pacotes possíveis de processar em simultâneo, bem como se elimina a necessidade de se proceder a uma cópia extra dos pacotes (da memória local para a memória principal).

A implementação no circuito de uma interface PCI permitiu optar pelo segundo modelo e colher as vantagens que lhe estão associadas. Acresce que a opção por uma interface padrão, largamente utilizada, contribui para a maior universalidade do circuito. Outra das vantagens do barramento PCI é a possibilidade de um elemento colocado no barramento poder aceder à memória principal (ou a outro dispositivo no barramento) sem intervenção do

CPU (*bus master*). Assim, as transferências de e para a memória principal do campo de dados das células (48 byte), efectuam-se a comando do HAL5 sem requerer qualquer envolvimento do CPU do sistema. Este só necessita de proceder a transferências de dimensão reduzida, para aceder a estruturas de dados de controlo, proceder ao estabelecimento das comunicações, fornecer a localização de *buffers* livres na memória (para se proceder à reconstrução dos pacotes), etc.

A memória principal do sistema também é utilizada para manter as principais estruturas de dados de controlo do processo de transmissão e recepção. Evita-se assim a inclusão de qualquer integrado de memória na placa adaptadora. De outro modo seria necessário a utilização de memória estática (SRAM) rápida na ordem dos 64 aos 512Kbyte - aumentando os custos de implementação. Por outro lado consegue-se uma maior acessibilidade a estas estruturas pelo *software* de controlo.

O HAL5 fornece suporte em *hardware* para os protocolos AAL0 e AAL5. O protocolo AAL0 não requer processamento especial uma vez que este protocolo não define qualquer funcionalidade para a camada AAL. As células ATM são colocadas sem alteração na memória principal. De igual modo, não é realizado qualquer processamento nos pacotes a transmitir.

<48 byte	8 bit	8 bit	16 bit	32 bit
PAD	UU	CPI	Comprimento	CRC32

Para o protocolo AAL5, a circuito encarrega-se de incluir em cada pacote, a informação de controlo da camada AAL-CS (ver 2.4.4.1), incluindo o cálculo e inserção do campo de alinhamento (PAD) do pacote, a um múltiplo de 48 byte. O cálculo (na transmissão) e verificação (na recepção) do campo CRC, que protege a integridade dos dados contidos no CS-PDU, também é realizada (*on-the-fly*) pelo HAL5. Este é um ponto importante uma vez que se trataria de uma tarefa 'dispendiosa', em termos de tempo de CPU do sistema central.

Na recepção, o número de canais virtuais abertos só é limitado pela quantidade de memória disponível para as estruturas de controlo (cada canal aberto requer 16 byte de

informação de controlo)*. A estrutura adoptada para a transmissão de dados implementa até 255 filas, em que cada uma pode conter até 64 CS-PDUs para transmissão simultânea. A taxa de transferência atribuída a cada fila pode ser definida com uma granulosidade de 32Kbit/s. Esta solução adapta-se bem a serviços do tipo CBR. Para serviços do tipo VBR permite variar a taxa de transferência entre cada pacote transmitido. Os serviços do tipo ABR podem ser implementados num mecanismo de gestão de largura de banda combinado com o *hardware* de controlo.

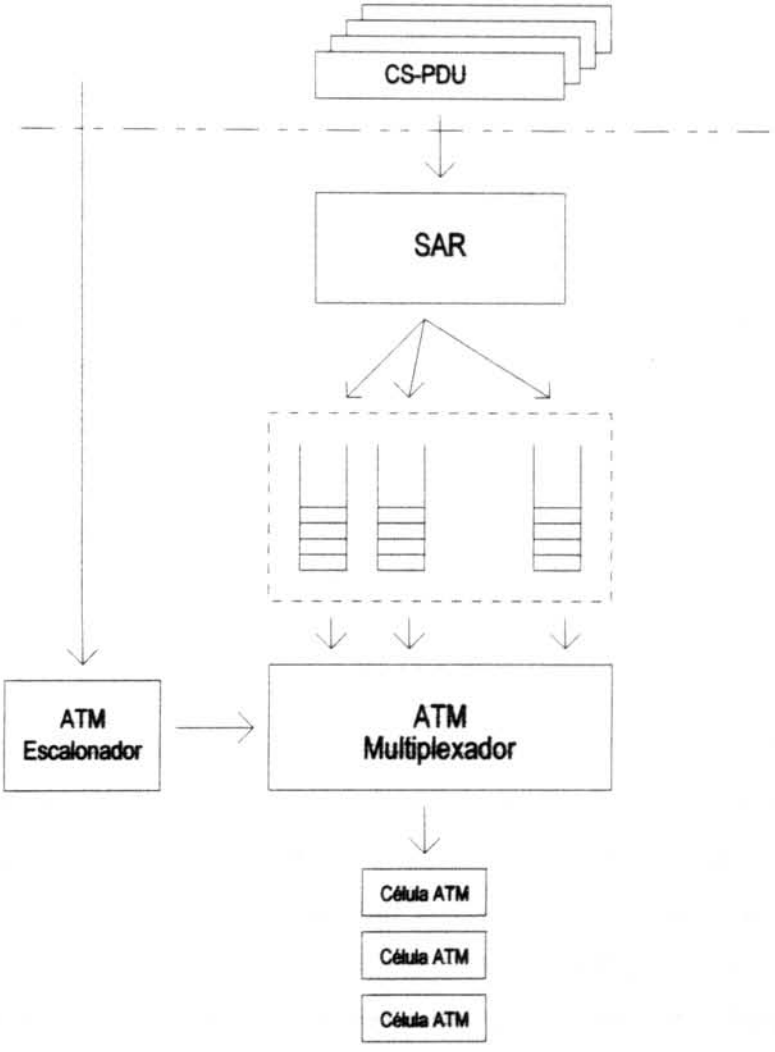


Figura 3.1: Estrutura de Transmissão

* De recordar que a camada ATM define um espaço de endereçamento - na UNI - de dimensão máxima 2^{24}. A dimensão efectiva do espaço de endereçamento é acordado entre o utilizador e a rede.

3.2 A Arquitectura

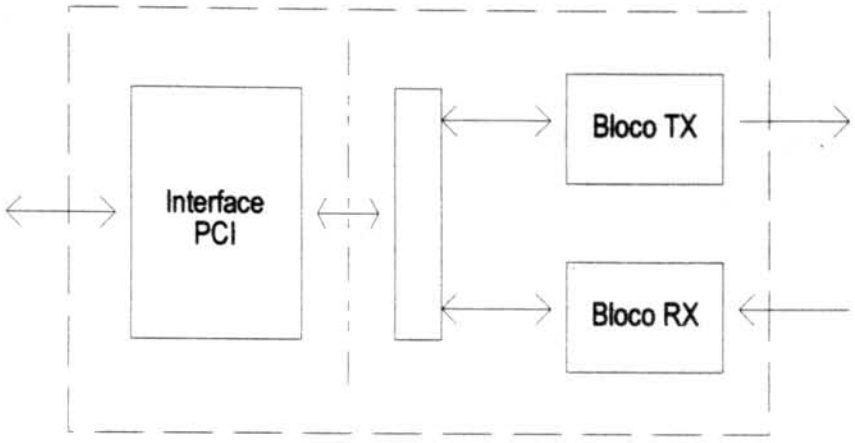


Figura 3.2

A figura 3.2 descreve a arquitectura do circuito no seu nível de abstracção mais elevado. Este primeiro nível de abstracção poderá parecer, numa primeira análise, algo redundante: dada a funcionalidade do circuito esta é de facto uma das partições naturais do 'problema'. A virtude deste esquema deve-se ao facto de revelar uma partição importante na arquitectura do circuito - de um lado o bloco que implementa a interface PCI e do outro os blocos que implementam as funções principais do circuito. Esta partição apresenta vantagens a dois níveis:

i) A modelização, implementação e validação de uma interface PCI, é uma tarefa complexa e dispendiosa. É portanto de todo o interesse evitar esta tarefa, especialmente se já foi realizada por terceiros. Um circuito digital que implemente uma interface PCI é um elemento padrão, e existem disponíveis no mercado modelos em linguagens de descrição de *hardware* (HDL), eficazes, completamente testados e aptos a serem sintetizados. Uma das vantagens da utilização de HDLs é de facto, a possibilidade de dispor de bibliotecas com circuitos padrão para re-utilização, e com isto diminuir o tempo e custo do projecto digital.

ii) A possibilidade de utilizar o circuito em sistemas com diferentes arquitecturas, revela-se mais 'económica'. Por exemplo, para implementar o terminal ATM num

sistema da arquitectura SUN basta substituir o bloco que implementa a interface PCI, por outro que implemente a interface de acesso ao barramento S-BUS. Desta forma garante-se o aproveitamento do esforço aplicado no desenvolvimento do resto do circuito.

Uma das opções estratégicas no projecto do HAL5, foi a decisão de proceder à obtenção exterior, do bloco que implementa a interface PCI. Este componente, descrito em Verilog HDL (ou em outra HDL), já validado e testado, é então sintetizado e incluído no circuito (ver 4.1.6).

A figura 3.3 apresenta a arquitectura do circuito em maior detalhe.

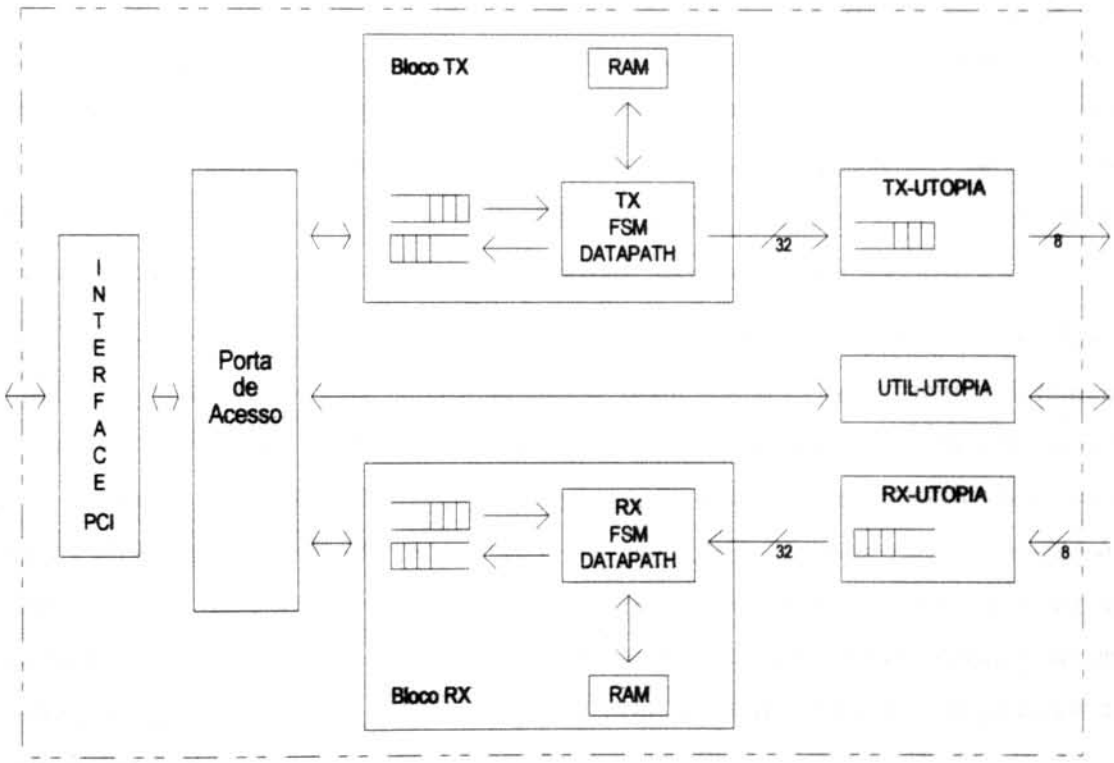


Figura 3.3: Esquema de Blocos

O bloco que implementa a interface PCI opera à velocidade do barramento PCI - 33MHz. O resto do circuito foi projectado para funcionar à velocidade de 50MHz. O bloco designado por 'Porta de Acesso' representa toda a lógica utilizada para a fazer a ligação entre

o bloco que implementa a interface PCI e o resto do circuito. Na terminologia PCI é o que se chama o 'BackEnd PCI'. Além de fazer a ponte com o resto do circuito, ordena os pedidos de acesso à memória principal por parte do bloco de recepção e o do bloco de transmissão.

A arquitectura adoptada para o circuito obriga a que o corpo principal do circuito opere a uma velocidade superior à da rede ATM. A utilização da memória principal do sistema - um recurso partilhado - implica a introdução de parâmetros não quantificáveis, sendo o mais importante o tempo de espera no acesso ao barramento PCI (ver 4.3.1). Este factor pode afectar negativamente o desempenho do terminal, ao dificultar o cumprimento do tempo médio de processamento para cada célula, condição necessária à manutenção dos parâmetros estabelecidos para o bom funcionamento da comunicação (especialmente o CDV: a diferença entre o tempo esperado e o tempo real, de chegada da célula ao seu destino).

Assim, o processamento das células é efectuado a uma velocidade superior à da rede ATM, principalmente para fazer face à variabilidade no tempo de acesso ao barramento PCI e a eventuais situações de excepção no funcionamento do terminal (por exemplo, a exaustão temporária dos *buffers* utilizados para armazenamento dos pacotes de dados). O circuito inclui duas FIFOs, uma para cada sentido da comunicação, para permitir o armazenamento das células e regularizar o fluxo das células entre o circuito e a camada física.

A comunicação com a camada física é realizada através de uma interface UTOPIA a 25MHz, o que estabelece uma pressão constante nesta interface (para 155Mbit/s bastaria operar esta interface a 19.44MHz). O estabelecimento da velocidade da transferência dos dados no nível físico é então realizado pelo integrado que implementa a camada física. Desta forma o HAl5 pode ser utilizado para qualquer velocidade de transmissão no meio físico (não superior a 155Mbit/s) sem necessidade de nenhuma alteração, uma vez que a definição da taxa de transferência é feita em unidades de células transmitidas na camada física, e não em células por segundo (ver 3.4.2), o que também permite uma maior precisão e simplicidade na definição das taxas de transferência.

Os blocos TX-UTOPIA e RX-UTOPIA consistem basicamente nas FIFOs referidas, e na lógica de controlo para implementação dos sinais do barramento UTOPIA. As FIFOs, além de adaptarem a velocidade de processamento das células à velocidade na interface UTOPIA, permitem adaptar os dados à largura dos barramentos de dados internos do circuito, os quais tem uma largura de 32 bit. A FIFO de recepção tem capacidade para 256x32

bit e a FIFO de transmissão para 128x32 bit. Cada célula é armazenada nas FIFOs em 13x32 bit. O quinto byte do cabeçalho da célula é desprezado na recepção e, na transmissão, é preenchido com um valor aleatório. Esta posição corresponde ao campo HEC, que é mantido pela camada física. Assim, na recepção, o cabeçalho da célula já foi verificado e por consequência o seu valor é redundante (só são recebidas células sem erros no cabeçalho). Na transmissão o cálculo do HEC é realizado ao nível da camada física pelo e inserido no quinto byte do cabeçalho.

Os blocos TX e RX (Fig. 3.3) tem uma constituição praticamente idêntica. São constituídos, cada um, por uma única máquina de estado, e pelo respectivo *datapath*. Ambos os blocos contém uma RAM de pequenas dimensões (256x38bit e 32x32bit respectivamente) para manter a informação de controlo que não pode ser mantida na memória principal do sistema, por necessitar de ser acedida com elevada frequência e rapidez. No caso do bloco TX a RAM mantém os valores parciais do cálculo do campo CRC, dos CS-PDUs em transmissão. No caso do bloco RX a RAM constitui uma *cache* para optimizar o acesso a algumas estruturas de dados relacionadas com o processo de recepção e, também, para diminuir a influencia do tempo de acesso ao barramento PCI no desempenho do circuito.

Cada um dos blocos contém ainda duas FIFOs para aceder à memória principal. Uma delas serve para colocar os comandos de leitura e de escrita (neste caso os dados são incluídos no comando) à memória principal. A outra serve para receber os dados provenientes de comandos de leitura. A utilização destas FIFOs também permite diminuir a dependência do desempenho do circuito, dos acessos à memória principal. Por exemplo, uma das situações comuns no funcionamento dos blocos é a actualização das estruturas de dados na memória principal. A utilização de FIFOs permite, de forma simples, que o comando de actualização (escrita) dos dados seja 'despachado' e que o processamento da próxima célula comece em simultâneo.

Estas FIFOs são de menores dimensões comparadas com as FIFOs utilizadas nos blocos UTOPIA. Ao todo, a capacidade de memória incluída no circuito sobre a forma de FIFOs e RAMs é inferior a 3Kbyte, o que na tecnologia utilizada representa uma área de silício de cerca de $10mm^2$.

3.3 As interfaces com o exterior

3.3.1 Interface PCI

O padrão PCI define um barramento local de elevada eficiência, para processadores e controladores periféricos em ambientes de alta velocidade. Foi adoptado por vários dos intervenientes de maior destaque na indústria, tais com a Motorola e a Intel, para a implementação dos seus processadores e computadores. Estes predicados levaram à opção pela sua utilização na implementação do terminal ATM. A própria arquitectura do HAL5 alicerça-se no alto desempenho oferecido pelo barramento PCI.

A interface PCI adoptada para o HAL5 tem uma largura de 32 bit a 33MHz, a qual é a configuração mais utilizada actualmente. A largura de banda máxima (teórica) que lhe está associada é de 132MByte/s, o que é suficiente para suportar uma ligação ATM a 155Mbit/s. Se considerarmos valores aproximados, serão necessários cerca de 20MByte/s para transferir os dados para a memória principal, mais 5MByte/s para os acessos às estruturas de dados de controle (de recordar que o HAL5 utiliza a memória principal para manter esta informação). Assim, a comunicação a 155Mbit/s em ambos os sentidos simultaneamente exigiria, aproximadamente, uma largura de banda de cerca de 50Mbit/s no acesso à memória principal. Este valor ainda é inferior a 50% da largura de banda disponível no barramento. E há a considerar que não será plausível a manutenção de uma comunicação bi-direcional a estas velocidades senão por curtos períodos de tempo e esporadicamente: não é só uma questão de transferir informação; normalmente realiza-se algum processamento sobre a mesma, e a maioria dos CPUs dos sistemas a que se destina o terminal não está habilitado a lidar com esta quantidade de informação - e ainda dispor de tempo para outras tarefas.

Outro factor importante é o tempo de latência no acesso ao barramento, e consequentemente à memória principal do sistema. Para manter uma interface com a rede ATM a uma velocidade de 155Mbit/s, é necessário processar cada célula, em média, num tempo não superior a 2.8us (53*8/155e6). A arquitectura adoptada para o HAL5, depende fortemente do desempenho do barramento PCI e requer, para um funcionamento correcto, que o tempo de latência no barramento seja inferior a 1us.

Uma das características do barramento PCI é o reduzido tempo de espera (latência) no acesso ao meio. E, dispondo de um esquema eficiente de arbitragem, permite estabelecer políticas de acesso partilhado eficientes. Nomeadamente através da configuração de diversos parâmetros associados aos dispositivos e placas adaptadoras ligadas ao barramento. Assim, para o bom funcionamento do HAL5, é necessário que todos os dispositivos no barramento com capacidade '*bus master*' (capacidade para iniciar transferências no barramento), tenham configurados os seus '*Latency Timers*' com valores inferiores a 20 (representando o número de *clocks* PCI, que um dispositivo *master* pode utilizar para *bursts* no barramento. Ver 5.1.6). Desta forma será possível garantir o acesso ao barramento em tempo inferior a 1us, e permitir manter a integridade da ligação ATM a 155Mbit/s.

No entanto, não é possível garantir a invariabilidade deste parâmetro, uma vez que este depende de vários factores não controláveis (se o barramento está a ser usado, por quem, etc). Frequentemente, o tempo de espera para aceder ao barramento será inferior a 0.5us, e normalmente não ultrapassará 1us, mas, em certas situações, este tempo pode assumir valores mais elevados [28].

Para atenuar a dependência deste parâmetro, e permitir manter a integridade da interface ATM, em termos de qualidade de serviço, o circuito processa as células a uma velocidade superior à da rede ATM. Duas FIFOs, incluídas no circuito, permitem armazenar até 10 células no sentido da transmissão e 20 células no sentido da recepção.

A dimensão das FIFOs foi escolhida de forma a permitir a *bufferização* de dados equivalente a 30us. Este é o valor recomendado [28] para fazer face às situações do tipo 'pior cenário' no acesso ao barramento PCI. A FIFO de recepção tem uma capacidade superior, uma vez que os efeitos de quebra de integridade da interface são mais negativos no caso da recepção, em que há lugar à perda de células, do que no caso da transmissão, em que há lugar a perda da integridade temporal da comunicação. O desempenho do circuito beneficiaria com o aumento da dimensão de ambas as FIFOs, por lhe permitir maiores tempos de recuperação, de situações em que a latência do barramento atinja valores elevados. No entanto é necessário ter em conta o custo associado ao seu dimensionamento: FIFOs maiores implicam uma maior área do circuito e consequentemente um maior custo de fabricação do CI.

A utilização de duas RAMs internas, de pequenas dimensões, para optimizar o acesso à memória principal, também permite atenuar a dependência do circuito, do tempo de acesso

ao barramento PCI. No bloco TX, a RAM permite manter *in-chip* a informação de controlo do processo que necessita de ser acedida com uma frequência elevada. No bloco RX, a RAM utilizada permite reduzir o número de acessos à memória principal, ao condensar vários acessos num único acesso em bloco. Em ambos os casos o objectivo é o de reduzir o número de acessos à memória principal.

A dependência do funcionamento do circuito, do tempo de acesso ao barramento PCI, foi analisada através de uma simulação por *software*, em que se modelizou o funcionamento do circuito em função da latência no barramento, e da velocidades de transmissão de dados (Ver capítulo 5).

3.3.2 Interface UTOPIA

A interface UTOPIA (*Universal Test and Operations PHY Interface for ATM*) é um padrão que define um protocolo de comunicação de barramento, que pretende facilitar a interligação e integração de equipamento da camada física com outro sistemas.

A especificação do HAL5 procurou seguir as normas definidas pelas instâncias que 'regem' o ATM e, a opção pela utilização deste protocolo de barramento para comunicação com o(s) dispositivo(s) que implementam a camada física, surge naturalmente. A quase totalidade dos dispositivos do nível físico no mercado segue esta norma de comunicação e, a sua não utilização constituiria um *handicap* forte na arquitectura do HAL5. As vantagens da utilização de uma interface padronizada, incluem a facilidade de utilizar componentes do nível físico com características variadas. A implementação do terminal torna-se independente do meio físico que suporta a rede, seja ele óptico ou não (UTP3, por exemplo).

Este protocolo, na sua simplicidade, é extremamente eficiente. Consiste em dois barramentos de comunicação de 8 bit cada (Rx e Tx) que incluem um sinal de relógio para sincronizar o fluxo de dados, e um barramento assíncrono (designado por utilitário) para aceder a registos e informação interna do dispositivo da camada física. O sincronismo dos barramentos de dados pode ser realizado ao byte ou 'à célula'. Normalmente o dispositivo da camada física contém duas FIFOs ligadas ao barramento UTOPIA que permitem *bufferizar* um número reduzido de células, em ambos os sentidos.

3.4 O Fluxo dos Dados

3.4.1 Recepção

As células são recebidas através do barramento UTOPIA e armazenadas numa FIFO com capacidade para 20 células; a FIFO faz a adaptação da largura dos dados de 8 para 32 bit (largura dos barramentos internos do HAL5). O cabeçalho da célula fornece a indicação do canal virtual a que esta pertence(VPI/VCI), o qual é utilizado para indexar a tabela de conexões activas, e identificar a célula. A informação contida na tabela permite determinar o *buffer* adequado para armazenar o campo de dados da célula. À medida que as células são armazenadas e os pacotes reconstruídos, é calculado o valor parcial do CRC. Após a recepção da última célula do pacote é assim possível verificar a integridade do mesmo (AAL5).

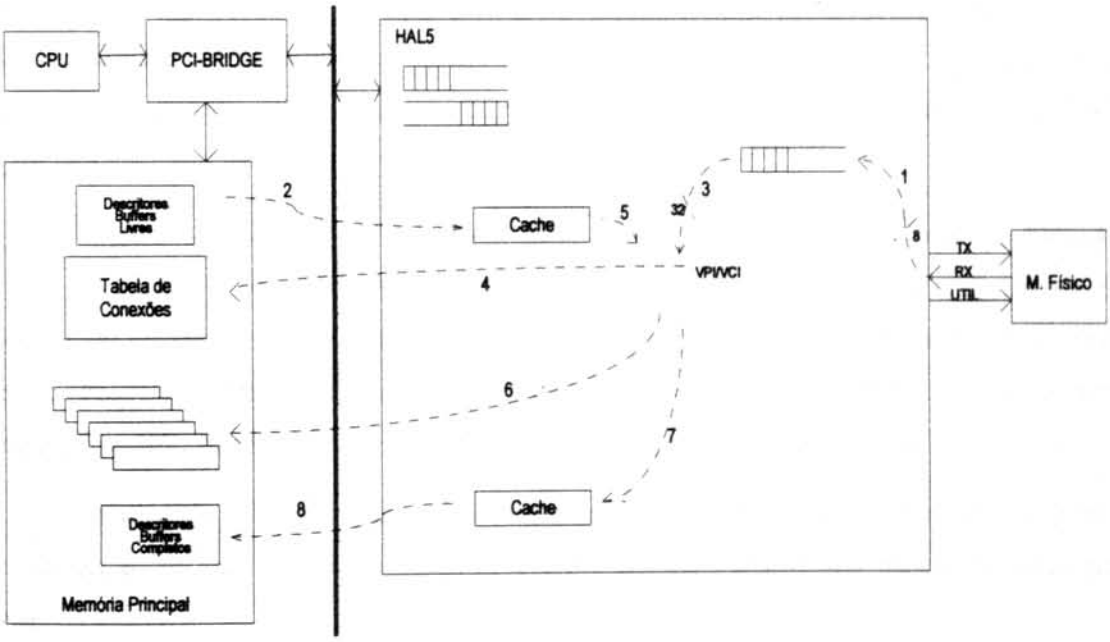


Figura 3.4: Fluxo dos Dados, Recepção

A figura 3.4 representa o fluxo dos dados associado à recepção.

Na memória principal existem três estruturas de dados. A fila circular de 'Descritores de *Buffers* Livres' contém apontadores para os *buffers* a utilizar no processo de reconstrução dos pacotes. A 'Tabela de Conexões' mantém informação sobre os canais virtuais abertos para recepção de dados. Indica o estado do processo de reconstrução, o protocolo utilizado, etc. A fila circular de 'Descritores de *Buffers* Completos' serve para devolver os *buffers* à medida que vão sendo preenchidos. Uma RAM *in-chip* permite optimizar o acesso à fila de 'Descritores de *Buffers* Livres' e à fila de 'Descritores de *Buffers* Completos'.

O processo de recepção começa depois de o *software* de controlo proceder à inicialização das estruturas de controlo do processo. Esta operação inclui o preenchimento da fila de 'Descritores de *Buffers* Livres'. Existem dois tipos de *buffers* de tamanho configurável: os *buffers* 'grandes' (1 a 16Kbyte) e os *buffers* pequenos (64 a 512 byte).

A utilização de dois tamanhos distintos fixos tem o objectivo de, por um lado, evitar esquemas complexos de gestão de memória, e por outro permitir uma melhor utilização da mesma. O primeiro *buffer* utilizado na reconstrução de um pacote é sempre do tipo 'pequeno'. Os *buffers* subsequentes serão do tipo 'grande'. O desperdício de memória na reconstrução de pacotes pequenos é assim menor, do que se utilizasse um único tamanho de *buffer*. Por outro lado a reconstrução de pacotes grandes é realizada eficazmente com o recurso a *buffers* 'grandes'.

O processo de recepção prossegue com a análise do cabeçalho da célula. O cabeçalho é lido da FIFO de recepção (à excepção do HEC, que já foi eliminado), e a 'Tabela de Conexões' é indexada utilizando o endereço VPI/VCI. A informação da tabela é recebida através do barramento PCI e permite identificar a célula: se pertence a um canal aberto para recepção, estado da reconstrução do pacote, protocolo utilizado, valor parcial do CRC, etc.

Se a célula for a primeira célula de um pacote, ou se o *buffer* em utilização para a reconstrução do pacote estiver completo, a RAM interna é consultada para obter um *buffer* livre.

O endereço do *buffer* na memória principal é utilizado para proceder à transferência dos dados da célula para a memória principal. Em simultâneo é actualizado o valor parcial do CRC do pacote.

Após a transferência, se o *buffer* estiver completo, o seu descritor é colocado na fila de 'Descritores Completos' para permitir o seu processamento pelo *software* de controlo. Na conclusão de cada pacote é gerada uma interrupção para informar o *software* de controlo, de que a fila 'Descritores Completos' deverá ser processada.

O *software* de controlo processa a fila 'Descritores Completos' e recolhe os descritores dos *buffers* que constituem o pacote (CS-PDU), para os fornecer ao *software* de comunicações.

3.4.2 Transmissão

Os pacotes a transmitir encontram-se na memória principal em *buffers* de tamanho variável. A sua transmissão inicia-se após a sua colocação numa das 255 filas de transmissão disponíveis. A cada fila de transmissão está associada uma taxa de transferência definida pelo utilizador com uma granulosidade de aproximadamente 32Kbps.

Na memória principal existem apenas duas estruturas de dados. A 'Tabela de Largura de Banda' e a 'Tabela das Filas de Transmissão'. A primeira determina, célula a célula, qual a próxima fila de transmissão a ser servida. Desta forma a taxa de transferência é definida em unidades de célula transmitidas no meio físico. A vantagem é a de poder utilizar o circuito em interfaces de diferentes velocidades, sem necessitar de nenhuma adaptação especial. Mais uma vez, o acesso a esta estrutura é optimizado através da utilização de uma pequena *cache in-chip*.

A segunda tabela permite definir até 255 filas circulares para transmissão de pacotes. Cada uma das filas pode conter até 64 pacotes para transmissão, ou melhor, os descritores dos *buffers* que contém os pacotes. A manutenção dos apontadores de acesso às filas é realizado numa RAM interna ao circuito. Esta RAM permite, também, manter os valores dos CRCs parciais dos pacotes em transmissão.

O processo de transmissão pode começar após a inicialização da 'Tabela de Largura de Banda' pelo *software* de controlo. Este pode modificar a tabela dinamicamente durante o funcionamento do processo de transmissão, e desta forma alterar as taxas de transferência

associadas a cada fila *on-the-fly*. Após a colocação dos descritores dos *buffers* a transmitir, o processo de transmissão inicia-se automaticamente.

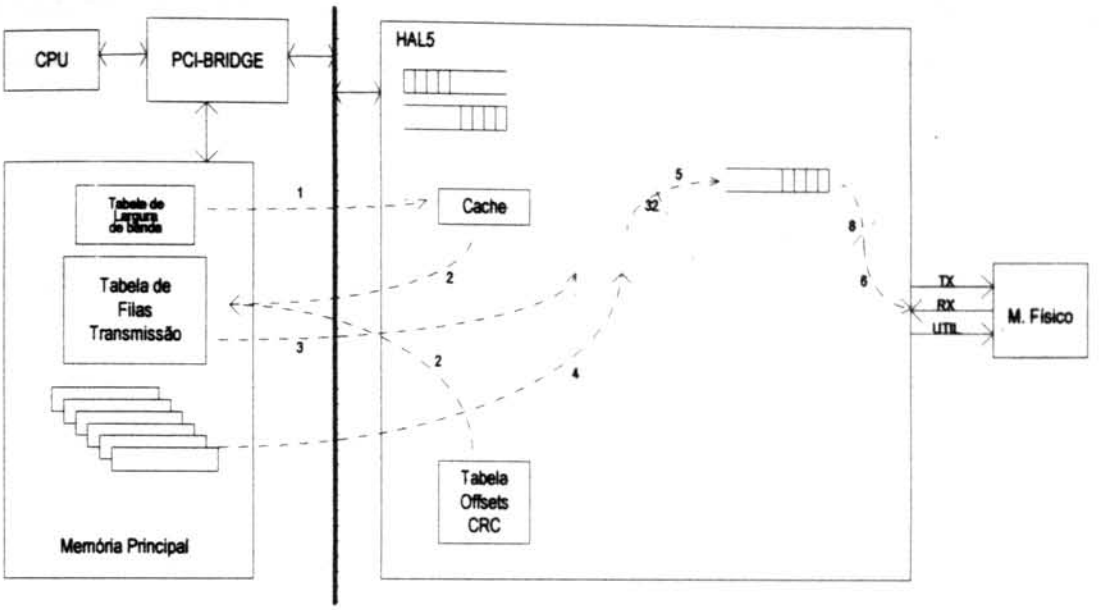


Figura 3.5: Fluxo dos Dados, Transmissão

A 'Tabela de Largura de Banda' é lida pelo HAL para determinar qual a próxima fila de transmissão a servir. De seguida, utilizando a informação contida na RAM interna ('Tabela Offsets+CRC') endereça a fila adequada e obtém a informação relativa ao pacote a processar. A informação obtida inclui, o cabeçalho a inserir na célula a transmitir, o endereço do buffer na memória, o comprimento do pacote, o protocolo a utilizar, etc.

O cabeçalho a inserir na célula ATM é gerado pelo HAL5 e o campo de dados é lido do *buffer* na memória principal através do barramento PCI. Simultaneamente, o valor parcial do CRC é actualizado. O valor parcial do CRC de cada pacote em transmissão é mantido na RAM interna do circuito.

A célula é então enviada para a FIFO de transmissão. Nesta altura o cabeçalho apenas contém quatro bytes, uma vez que não foi incluído o campo HEC. À saída da FIFO, o bloco TXUTOPIA insere um byte nessa posição, o qual será re-escrito pelo dispositivo da camada física com o valor correcto do HEC.

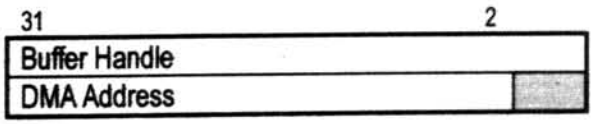
Após a conclusão do envio de um pacote, a informação de controlo do protocolo AAL é automaticamente inserida no fim do CS-PDU (apenas para o protocolo AAL5). O valor final do CRC, calculado pelo HAL5, é incluído nesta informação de controlo. O cálculo da dimensão do campo PAD, que tem por função alinhar a dimensão do CS-PDU a um múltiplo de 48 byte, também é realizado pelo HAL5 e inserido automaticamente.

3.5 As Estruturas de Dados de Controlo

3.5.1 Recepção

- Filas de Descritores de *buffers* livres (FBLRXp & FBLRXg)

Estas duas filas circulares permitem fornecer o HAL5 com os *buffers* necessários ao processo de reconstrução dos CS-PDUs. Uma das filas circulares destina-se a *buffers* grandes, e a outra a *buffers* pequenos. Cada elemento das fila ocupa 8 byte e identifica um *buffer*. Cada fila tem capacidade para 1024 elementos (8Kb).



Buffer handle - endereço virtual do *buffer*.
DMA Address - endereço físico do *buffer*.

Ao funcionamento destas estruturas estão associados os seguintes registos internos:

FBLRXg_BASE[31:13] - endereço base da fila G - múltiplo de 8Kbyte.
FBLRXg_HEAD[12:5] - *offset* do último elemento usado pelo HAL5.

FBLRXg_TAIL[12:5] - *offset* do último elemento fornecido pelo *software* de controlo.

FBLRXp_BASE[31:13] - endereço base da fila P - múltiplo de 8Kbyte.

FBLRXp_HEAD[12:5] - *offset* do último elemento usado pelo HAL5.

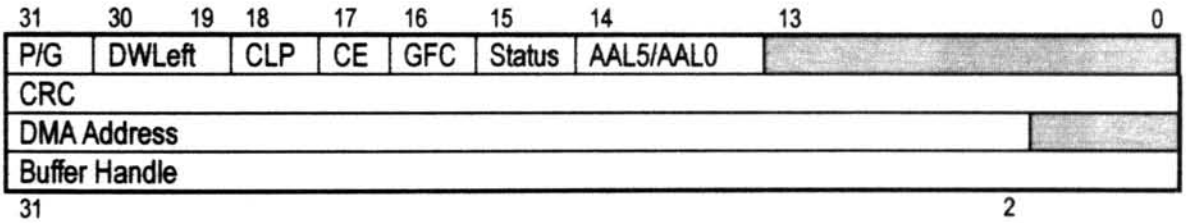
FBLRXp_TAIL[12:5] - *offset* do último elemento fornecido pelo *software* de controlo.

CONFIG[1:0] - tamanho dos *buffers* P: 00=64byte,01=128,10=256,11=512

CONFIG[3:2] - tamanho dos *buffers* G: 00=1Kb,01=2Kb,10=8Kb,11=16Kb

- Tabela de Conexões (TCRX)

Cada elemento desta tabela define um canal de comunicação (VPI/VCI). O número de bit utilizados para a indexar é configurável por *software*: entre 8 a 16 bit do VCI, e entre 0 e 8 bit do VPI. Cada elemento ocupa 16 byte e a dimensão da tabela é definida pelo número de bit utilizados para a indexar.



DWLeft - Indica o número de *dwords* livres no *buffer*. Se igual a 0 indica que não há nenhum *buffer* associado a este canal.

P/G - Indica o tipo de *buffer* em utilização: pequeno ou grande.

CLP - Indica se recebeu alguma célula com o campo do cabeçalho CLP=1.

CE - Indica se recebeu alguma célula com o campo do cabeçalho CE=1.

GFC - Indica se recebeu alguma célula com o campo do cabeçalho GFC!=0.

open - Indica se o canal está a receber dados.

AAL5/AAL0 - Indica o protocolo utilizado no canal de comunicação.

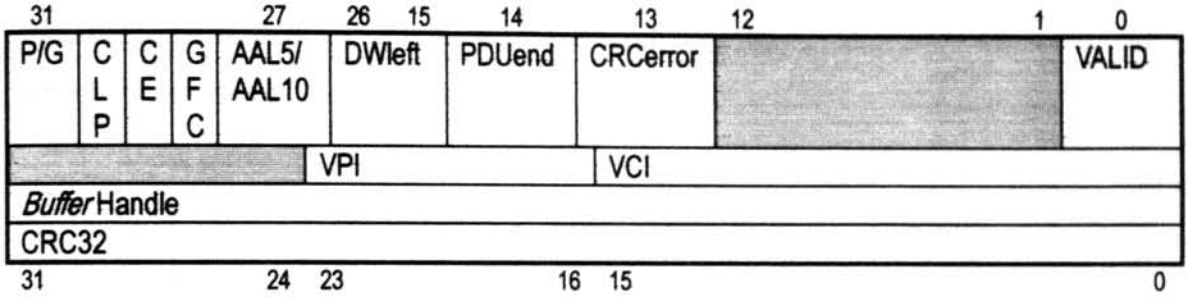
Ao funcionamento desta estrutura estão associados os seguintes registos internos:

TCRX[31:12] - endereço base da tabela (múltiplo de: $2^{(x+y-6)}$Kbyte).

TCRX_MASK[31:0] - define o número de bit a utilizar para indexar a tabela.

- Filas de Descritores de *buffers* completos (FBCRX)

Esta fila circular tem capacidade para 512 elementos, e é utilizada para devolver os *buffers* completos. Cada entrada (16 byte) descreve um *buffer*.



CRC32 - Contém o valor parcial/final do CRC.

VPI/VCI - Identifica o canal virtual a que pertencem o pacote.

Buffer Handle - Identifica o *buffer* devolvido.

P/G - Identifica o tipo de *buffer*: grande ou pequeno

CLP - Indica se recebeu alguma célula com o campo do cabeçalho CLP=1.

CE - Indica se recebeu alguma célula com o campo do cabeçalho CE=1.

GFC - Indica se recebeu alguma célula com o campo do cabeçalho GFC!=0.

AAL5/AAL0 - Indica o protocolo na reconstrução do pacote.

DWleft - Indica o espaço por utilizar no *buffer*.

PDUend - Indica se este *buffer* constitui o fim do pacote.

CRCerror - Indica se foi detectado um erro no pacote - CRC.

VALID - Indica se esta entrada na fila contém dados válidos

Ao funcionamento desta estrutura estão associados os seguintes registos internos:

FBCRX_BASE[31:13] - endereço base da fila - múltiplo de 8Kbyte.

FBCRX_HEAD[12:4] - *offset* do último elemento da fila processado pelo *software* de controlo.

FBCRX_TAIL[12:4] - *offset* do último elemento colocado na fila pelo HAL5.

- Fila de Raw Cells (FRCRX)

Para guardar as células que não fazem parte de pacotes a reconstruir (OAM F5, RM cells, lost cells, etc), o HAL5 utiliza *buffers* do tipo grande, com os quais cria uma lista ligada. Cada célula (13 dwords) é guardada em 16 *dwords*. Em cada *buffer* a última *dword* é utilizada para colocar um apontador para o próximo *buffer*.

A esta estrutura estão associados os seguintes registos internos:

FRCRX_BH[31:0] - descritor do *buffer*, na lista, que está a ser utilizado pelo HAL5 para armazenar as células.

FRCRX_DMA[31:2] - endereço físico do *buffer*

FRCRX_CLEFT[9:0] - espaço livre no *buffer* em unidades 16 *dwords*.

Os registos internos a seguir descritos referem-se ao funcionamento geral do processo de recepção:

CONFIG[4] - Se 0 suspende o processo de recepção.

CONFIG[22] - Se 0 as células inválidas (GFC!=0) são eliminadas e o contador ICC não é incrementado; Se 1 as células inválidas são colocadas na 'Fila de Raw Cells' e o contador é incrementado.

CONFIG[20] - Se 1 o valor do GFC da célula é ignorado.

CONFIG[18] - Se 0 elimina as células cujo VPI/VCI não se encontre na 'Tabela de Conexões' ou em que o canal não se encontre aberto; Se 1 as células são colocadas na 'Fila de Raw Cells'.

CONFIG[23] - Se 0 elimina as células RM (PTI=11x); Se 1 as células são colocadas na 'Fila de Raw Cells'.

CONFIG[13:12] - Se 0 não é gerada uma interrupção no fim da reconstrução de cada PDU; Se 1 gera interrupção de imediato; Se 2 aguarda 250us ; Se 3 aguarda 500us

CONFIG[12] - Se 1 gera interrupção quando completa cada *buffer* da 'Fila de Raw Cells'.

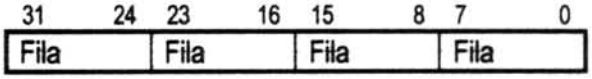
VEC[31:0] - Conta as células eliminadas por erro no VPI/VCI.

ICC[31:0] - Conta as células inválidas colocadas 'Fila de Raw Cells'.

3.5.2 Transmissão

- Tabela de Largura de Banda (TLB)

Esta tabela contém 4096 elementos (4Kb), onde cada entrada determina qual a próxima fila de transmissão a servir. Cada posição representa uma taxa de transferência de cerca de 36.6Kbit/s (149.76e6/4096).



O valor FFh em qualquer posição da tabela indica que o largura de banda respectiva não está a ser utilizada, e o HAL envia automaticamente uma célula vazia.

A esta estrutura só está associado um registo interno:

TLB_BASE[31:12] - endereço base da tabela (múltiplo de 4Kb)

- Tabela de Filas de Transmissão (TFT)

Esta tabela contém até 255 filas de transmissão. Cada elemento da tabela é uma fila circular de 64 posições. Cada posição da fila circular é composta por 16 byte, que identificam um *buffer* a transmitir.

31 16	15 2	1	0
Comprimento		AAL5/AAL0	VALID
DMA			
CELL HEADER			
CS-UU | CPI	LENGTH		

Comprimento - Indica a dimensão do *buffer*.

DMA - Aponta para o endereço do *buffer* transmitir.

CELL HEADER - Contem o cabeçalho a inserir na célula.

CS-UU, CPI, LENGTH - Informação de controlo do protocolo AAL5 a inserir no fim do pacote.

AAL5/AAL0 - Indica o protocolo de comunicação.

VALID - Indica se esta entrada na fila contém dados válidos

A esta estrutura está associado o seguinte registo interno:

TC_BASE[31:18] - endereço base da "tabela" (múltiplo de 256Kb)

- Tabela de *Offsets* das Filas

Esta tabela é mantida numa RAM interna do circuito, e permite manter os offsets para acesso às filas circulares de transmissão, bem como o CRC parcial do pacote em transmissão em cada uma das filas.

O processo de transmissão pode ser suspenso através do seguinte registo interno:

CONFIG[19] - Se 0 suspende o processo de transmissão.

Capítulo 4

Componentes do HAL5 II

4.1 Interface UTOPIA

A interface UTOPIA com o nível físico é implementada por dois blocos de constituição absolutamente simétrica. Ambos contém uma FIFO, de capacidades diferentes, que além de permitir *bufferizar* as células a transmitir/receber, efectua a adaptação entre a largura do barramento interno do HAL e a largura do barramento UTOPIA.

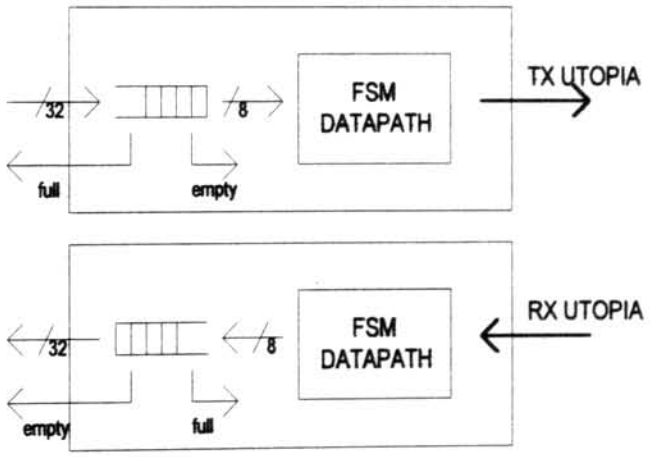


Figura 4.1: Blocos UTOPIA

O núcleo de cada um dos blocos UTOPIA (Fig. 4.1) é formado por uma máquina de estados que controla a transferência das células, de acordo com o protocolo do barramento e com o estado das FIFOs (cheia, vazia). Ao procederem à transferência das células, operam

sobre o 5º byte do cabeçalho (HEC). Na recepção este é eliminado e, na transmissão, é inserido um *place-holder* na posição do 5º byte (ver 3.2).

A figura 4.2 representa o barramento UTOPIA no sentido da transmissão. Os sinais assinalados com '(o)' são pinos de saída do HAL5. O sinal TxClk sincroniza as transferências de dados no barramento; todos os sinais são *sampled* no flanco ascendente de TxClc, e todos os sinais são gerados no flanco ascendente de TxClk.

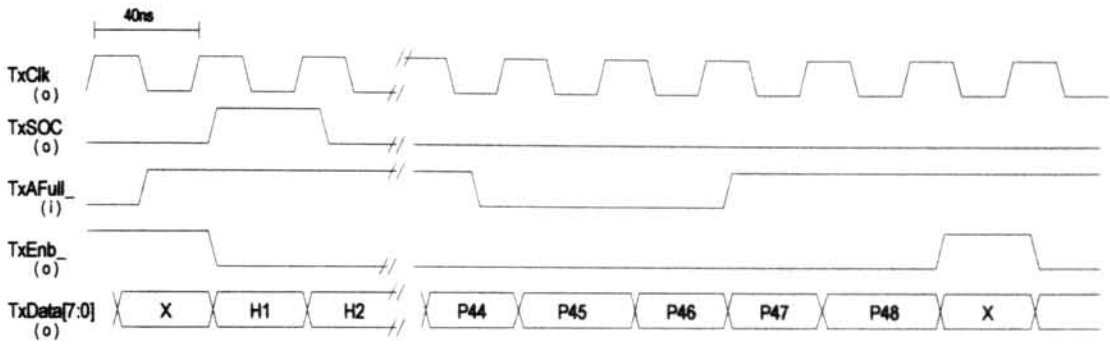


Figura 4.2: TxUTOPIA

O sinal TxEnb_ habilita a transferência de dados em TxData[7:0]. Sempre que TxEnb_ estiver activo o receptor da informação capta os dados no flanco ascendente de TxClk. O sinal TxAFull_ permite ao receptor (o CI do nível físico) indicar que não está em condições de receber mais dados. O sinal TxSOC só é activado durante a transferência do primeiro byte de cada célula, com o intuito de assinalar o início de cada célula.

A figura 4.3 representa o barramento UTOPIA no sentido da recepção. O seu funcionamento é em tudo simétrico ao do barramento de transmissão.

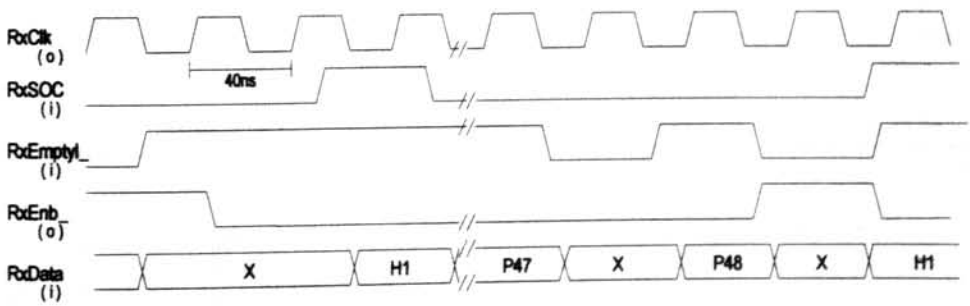


Figura 4.3: RxUTOPIA

Dada a relativa simplicidade destes dois blocos, a sua modelização e síntese não necessitou de cuidados especiais. Os resultados obtidos foram os seguintes:

```
TX-UTOPIA
*************************************************************

Timing Report
*************************************************************
Time unit:   1ns.      Precision:  10ps

************  Top Long Critical Path  **********************
Arrival     Inc  Fanout  Load   Cell(pin to pin)
 ------------ Long Critical Path 1 ----------------------
R(  0.00)   0.00 <-- clock      LIBDFFR(CK)
R(  1.52)   1.52    3    0.15   LIBDFFR(CK=>QZ)
F(  1.77)   0.25    2    0.11   LIBNA2(B=>Y)
R(  2.01)   0.24    1    0.05   LIBNA3(B=>Y)
F(  2.45)   0.45    6    0.32   LIBNA2(B=>Y)
R(  2.83)   0.38    2    0.07   LIBOAI22(B1=>Y)
F(  3.24)   0.41    1    0.05   LIBOAI21(A1=>Y)
R(  3.41)   0.17    1    0.03   LIBNA2(A=>Y)
R(  3.41)   0.00                LIBDFFR(D)
    3.41 ns

Area Report
*************************************************************
                                        Area$          Area$
          Cell          Count           each           total
-------------------------------------------------------------
          INV2              1         273.60          273.60
         AOI21              1         520.60          520.60
         AOI22              1         600.40          600.40
           NO4              1         608.00          608.00
           NA4              1         608.00          608.00
           OR3              1         653.60          653.60
           NO2              2         364.80          729.60
           NA3              3         524.40         1573.20
         OAI21              3         528.20         1584.60
           INV              7         273.60         1915.20
           XOR              3         836.00         2508.00
         OAI22              5         623.20         3116.00
          DFFS              2        1900.00         3800.00
           NA2             11         380.00         4180.00
          DFFR             10        2097.60        20976.00
=============================================================
         Total             52                       43646.80

finite state machine at "hdlFilesDir/txutopia.v", 35:
        States: 3
        Encoding: default
```

```
RX-UTOPIA
****************************************************************

Timing Report
****************************************************************
Time unit:   1ns.      Precision:  10ps

************  Top Long Critical Path  **********************
Arrival      Inc  Fanout  Load   Cell(pin to pin)
------------ Long Critical Path 1 -------------------------
R(  0.00)    0.00 <-- clock       LIBDFFR(CK)
F(  1.46)    1.46    6    0.30   LIBDFFR(CK=>QZ)
R(  1.92)    0.46    5    0.27   LIBNA2(A=>Y)
F(  2.37)    0.45    1    0.05   LIBOAI21(B1=>Y)
R(  2.59)    0.22    2    0.10   LIBNA2(B=>Y)
F(  3.22)    0.62    2    0.15   LIBAOI21(B1=>Y)
R(  3.33)    0.11    2    0.10   LIBINV2(A=>Y)
F(  3.72)    0.39    1    0.05   LIBAOI21(B1=>Y)
R(  4.02)    0.29    1    0.03   LIBOAI22(B1=>Y)
R(  4.02)    0.00                LIBDFFS(D)
    4.02 ns

Area Report
****************************************************************
```

Cell	Count	Area$ each	Area$ total
LOG2	1	311.60	311.60
INV2	2	273.60	547.20
OR3	1	653.60	653.60
MUX21	1	889.20	889.20
OAI222	1	946.20	946.20
AOI222	1	988.00	988.00
AN2	2	509.20	1018.40
NO4	2	608.00	1216.00
NO2	4	364.80	1459.20
XOR	2	836.00	1672.00
OAI22	3	623.20	1869.60
OR2	4	524.40	2097.60
OAI221	3	836.00	2508.00
NA4	5	608.00	3040.00
INV	12	273.60	3283.20
AOI21	7	520.60	3644.20
NA3	9	524.40	4719.60
AOI22	8	600.40	4803.20
OAI21	10	528.20	5282.00
NA2	23	380.00	8740.00
DFFS	9	1900.00	17100.00
DFFR	10	2097.60	20976.00
Total	120		87764.80

```
finite state machine at "hdlFilesDir/rxutopia.v", 51:
        States: 12
        Encoding: default
finite state machine at "hdlFilesDir/rxutopia.v", 35:
        States: 2
        Encoding: default
```

A máquina de estados que controla o processo de transmissão necessitou de 3 estados e 52 *standard-cells* para implementar a funcionalidade desejada.

O bloco que controla o barramento de recepção foi implementado com 2 máquinas de estados, uma com 2 estados e outra com 12 estados, utilizando na totalidade de 120 *standard-cells*. A justificação para este valor (em comparação com o bloco de transmissão) encontra-se na necessidade de tratar as situações de ***overrun*** que ocorrem na recepção, quando a FIFO se encontra cheia. Nesta situação o procedimento consiste em substituir a última célula introduzida na FIFO pela nova célula.

Em ambos os casos se obtiveram atrasos de propagação (3.41ns e 4.02ns) que não colocam problemas na operação dos blocos à velocidade desejada (T=20ns).

4.2 CRC32

Este subloco permite calcular o CRC32 dos CS-PDUs (AAL5). Esta função é necessária tanto para a transmissão como para a recepção. O método de cálculo implementado produz um CRC parcial a cada 32 bit, até atingir o resultado final sobre a totalidade do CS-PDU. A partir do circuito clássico para cálculo do código bit a bit derivou-se um bloco combinacional para efectuar o cálculo do CRC em blocos de 32 bit.

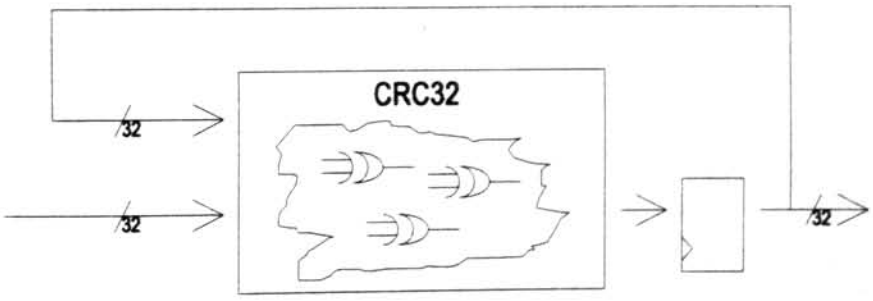


Figura 4.4: Sub-Bloco CRC32.

O bloco combinacional consiste em 32 funções lógicas XOR, que geram o novo valor parcial do CRC a partir do (seu) valor anterior e dos 32 bit de dados do CS-PDU. Dada a

complexidade das funções implementadas procurou-se obter uma solução que minimizasse o número de células utilizadas, e que simultaneamente permitisse o cálculo de cada valor parcial em um ciclo de relógio (T=20ns). Desta forma é possível calcular o CRC de forma transparente, em simultâneo com a transferência das células do CS-PDU para o bloco TX-UTOPIA.

A solução óptima em termos de número de células forneceu um tempo de propagação mais do que adequado. O bloco foi sintetizado em 635 células e, contra todas as expectativas, o tempo de propagação no *path* mais longo é de 4.67ns. De facto, tendo em conta que se optimizou para a área e não para a velocidade, e a natureza da função implementada, o tempo obtido constituiu uma afirmação da capacidade de optimização da ferramenta de síntese.

```
CRC32
*********************************************

Timing Report
*********************************************
Time unit:   1ns.      Precision:  10ps

************  Top Long Critical Path  *********************
Arrival     Inc  Fanout  Load   Cell(pin to pin)
-------------- Long Critical Path 1 ----------------------
F(  0.00)   0.00   13    0.63   Input_pin
R(  0.65)   0.65   12    0.60   LIBINV(A=>Y)
F(  1.23)   0.58    3    0.11   LIBOAI22(B2=>Y)
F(  1.93)   0.70    1    0.05   LIBXOR(A=>Y)
F(  2.62)   0.69    1    0.03   LIBXOR(A=>Y)
F(  3.31)   0.69    1    0.03   LIBXOR(B=>Y)
F(  4.00)   0.70    1    0.05   LIBXOR(B=>Y)
F(  4.67)   0.67    1    0.00   LIBXOR(A=>Y)
F(  4.67)   0.00                Output_pin
    4.67ns

Area Report
*************************************************************
                                        Area$          Area$
          Cell          Count          each           total
-------------------------------------------------------------
       LIBINV2              5        273.60         1368.00
      LIBBUF39              2        972.80         1945.60
        LIBINV             67        273.60        18331.20
      LIBOAI22            174        623.20       108436.80
        LIBXOR            387        836.00       323532.00
=============================================================
         Total            635                     453613.60
```

4.3 Bloco TX

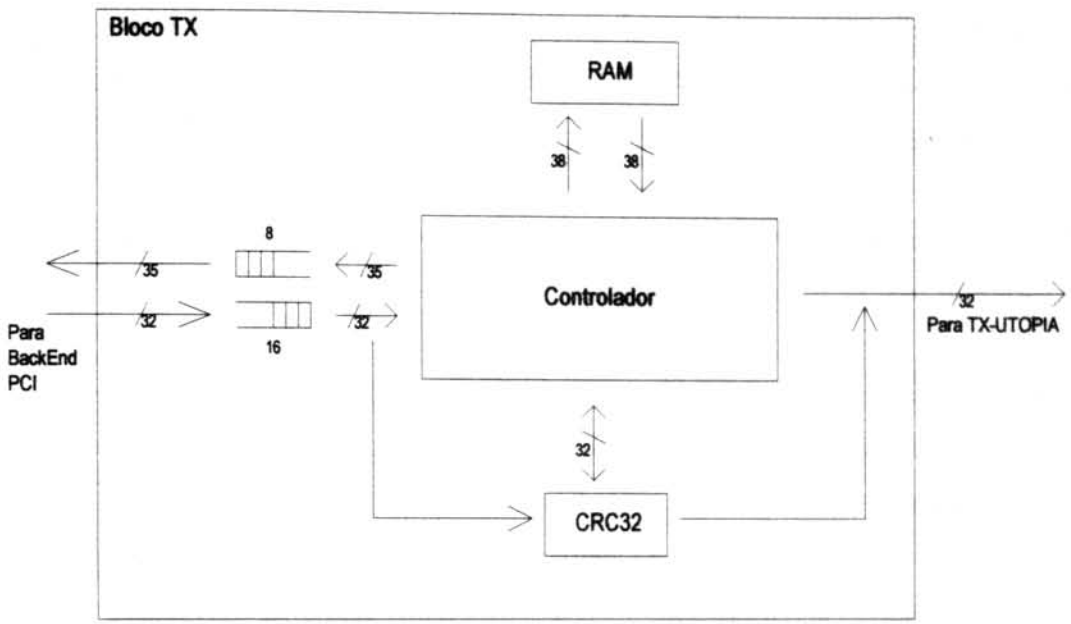


Figura 4.4: Bloco TX

O bloco que desempenha as funções de transmissão é constituído por quatro sub-blocos: uma RAM, duas FIFOs, e um sub-bloco principal, que controla/implementa as operações de transmissão.

A figura 4.4 apresenta a estrutura interna do bloco TX.

O componente principal foi completamente sintetizado, e consiste numa máquina de estados, vários registos de controlo, contadores, etc. É o componente que faz o escalonamento da transmissão, processa as células, e mantém as estruturas de dados de controlo. Em suma, todo o fluxo de dados descrito na secção 3.4.2 é por ele gerido.

O funcionamento do controlador obedece ao fluxograma descrito na figura 4.5. As principais operações por ele realizadas no decorrer desse fluxo são:

i) O acesso às estruturas de dados de controlo e aos CS-PDUs a transmtir, que estão colocados na memória principal. Esta operação é realizada utilizando duas FIFOS. Uma FIFO de 8x35bit, para colocar os pedidos de leitura e de escrita (neste caso

além do endereço, e do número de *double words* a aceder, os dados a escrever na memória também são colocados na FIFO), e outra de 16x32bit para receber os resultados das operações de leitura. A FIFO de retorno dos dados de operações de leitura é maior uma vez que tem que ter pelo menos capacidade para acolher 12x32 *double words* (o corpo da célula a transmitir), de modo a evitar situações em que a FIFO fique cheia. O que poderia afectar o desempenho do bloco de recepção: qualquer pedido de acesso à memória pelo bloco de recepção só pode ser atendido depois de os dados do processo de transmissão estarem todos colocados na FIFO de leitura. Esta situação dá-se simetricamente para ambos os processos, de recepção e de transmissão.

ii) A geração dos endereços na memória principal no acesso às estruturas de dados de controlo e aos *buffers* onde estão colocados os CS-PDUs a transmitir. Os endereços de acesso às estruturas de controlo, são calculados através dos valores presentes nos registos internos (acessíveis ao *software* de controlo pelo barramento PCI), na forma base, *offset*. Nos acessos aos *buffers* dos CS-PDU é ainda necessário proceder ao alinhamento a 32 bit dos dados, uma vez que os *buffers* utilizados para armazenar os CS-PDU a transmitir podem estar alinhados apenas ao byte.

iii) O acesso à memória RAM para leitura dos valores parciais do CRC para cada pacote, e do apontador de *offset* para cada fila de transmissão. As operações de leitura e escrita realizam-se através de dois barramentos de dados distintos.

iv) A escrita de dados para o bloco TX-UTOPIA através de uma FIFO (que se encontra à entrada do bloco TX-UTOPIA). Estes dados podem ser de duas origens: gerados pelo controlador (o cabeçalho das células, por exemplo, ou mesmo a totalidade da células no caso de se dar o caso de ser enviada uma célula *idle*); ou provenientes da memória principal aquando da transferência do corpo da célula. No segundo caso, após a disponibilidade da célula na FIFO de leitura da memória principal, a transferência é feita directamente de FIFO a FIFO.

v) A gestão do cálculo do CRC, actualizando os valores parciais de cada CS-PDU, com o cálculo realizado pelo componente CRC32. O cálculo do novo valor parcial é realizado transparentemente com a transferência do corpo da célula para o bloco TX-UTOPIA. As 12 *double words* que compõem o corpo da célula são capturadas

pelo componente CRC sobre a coordenação do controlador, de modo a calcular o novo valor parcial do CRC.

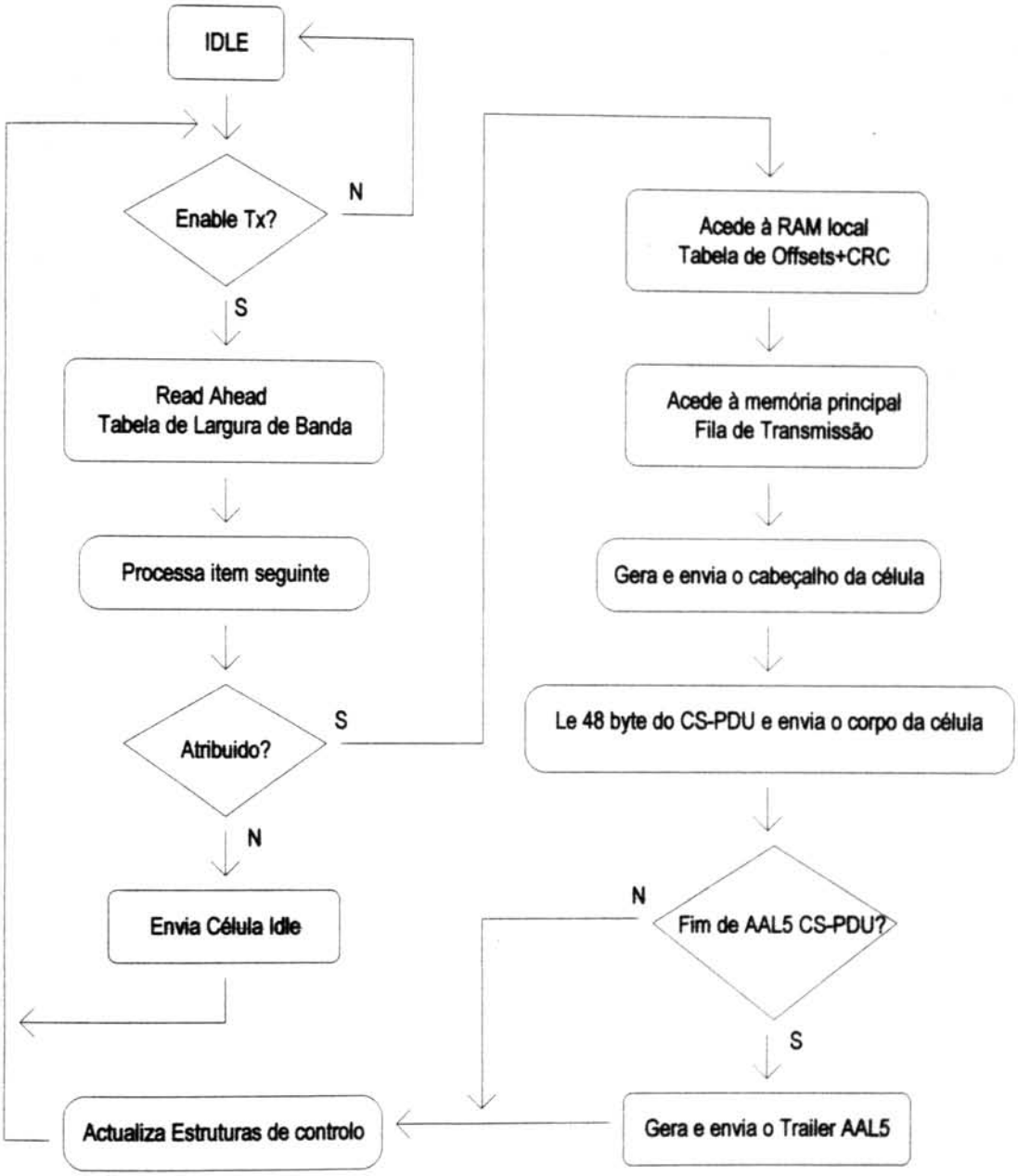


Figura 4.5: Fluxograma TX.

A relativa simplicidade do algoritmo implementado, o número elevado de registos de controlo, em parte pela necessidade de efectuar o alinhamento dos *buffers* de transmissão

(estes podem estar alinhados ao byte), a síntese deste componente requereu a realização de algumas iterações até se atingir uma boa solução em termos de área, e que permitisse cumprir com as especificações de velocidade.

Uma das opções que influenciou positivamente o resultado foi a adopção do método de codificação de estados 'one-hot'. Este método é particularmente promissor em situações em que o número de registos a controlar é elevado, o que veio de facto a verificar-se.

Assim, foi possível implementar este componente numa máquina de estados com 44 estados, e utilizando apenas 2883 *standard-cells*. Incluindo o bloco do CRC (635 células) foram necessárias no total 3518 *standard-cells* para implementar o bloco TX.

O tempo de propagação no *path* mais longo é de 15.27ns, o que permite que o componente funcione à velocidade desejada (50MHz) com uma folga aceitável.

```
TX
****************************************************

Timing Report
****************************************************
Time unit:   1ns.      Precision:  10ps

************  Top Long Critical Paths  *************
Arrival      Inc  Fanout  Load   Cell(pin to pin)
------------ Long Critical Path 1 ------------------
R(  0.00)    0.00 <-- clock      LIBDFF(CK)
R(  1.25)    1.25    6    0.33   LIBDFF(CK=>Q)
F(  3.15)    1.90    1    0.06   LIBFADD2(A1=>COUT)
F(  3.72)    0.56    1    0.06   LIBFADD2(CIN=>COUT)
F(  4.37)    0.65    3    0.17   LIBFADD2(CIN=>COUT)
R(  4.67)    0.30    3    0.17   LIBNA2(B=>Y)
F(  4.84)    0.17    1    0.05   LIBINV(A=>Y)
R(  5.14)    0.30    3    0.17   LIBNA2(B=>Y)
F(  5.31)    0.17    1    0.05   LIBINV(A=>Y)
R(  5.61)    0.30    3    0.17   LIBNA2(B=>Y)
F(  5.77)    0.17    1    0.05   LIBINV(A=>Y)
R(  6.08)    0.30    3    0.17   LIBNA2(B=>Y)
F(  6.24)    0.17    1    0.05   LIBINV(A=>Y)
R(  6.54)    0.30    3    0.17   LIBNA2(B=>Y)
F(  6.71)    0.17    1    0.05   LIBINV(A=>Y)
R(  7.01)    0.30    3    0.17   LIBNA2(B=>Y)
F(  7.18)    0.17    1    0.05   LIBINV(A=>Y)
..............
F( 10.92)    0.17    1    0.05   LIBINV(A=>Y)
R( 11.23)    0.30    3    0.17   LIBNA2(B=>Y)
F( 11.39)    0.17    1    0.05   LIBINV(A=>Y)
R( 11.70)    0.30    3    0.17   LIBNA2(B=>Y)
F( 11.86)    0.17    1    0.05   LIBINV(A=>Y)
R( 12.16)    0.30    3    0.17   LIBNA2(B=>Y)
```

```
F( 12.33)    0.17     1     0.05    LIBINV(A=>Y)
R( 12.63)    0.30     3     0.17    LIBNA2(B=>Y)
F( 12.80)    0.17     1     0.05    LIBINV(A=>Y)
R( 13.10)    0.30     3     0.17    LIBNA2(B=>Y)
F( 13.27)    0.17     1     0.05    LIBINV(A=>Y)
R( 13.57)    0.30     3     0.17    LIBNA2(B=>Y)
F( 13.73)    0.17     1     0.05    LIBINV(A=>Y)
R( 14.04)    0.30     3     0.17    LIBNA2(B=>Y)
F( 14.20)    0.17     1     0.05    LIBINV(A=>Y)
R( 14.51)    0.30     3     0.17    LIBNA2(B=>Y)
F( 14.67)    0.17     1     0.05    LIBINV(A=>Y)
R( 14.97)    0.30     1     0.17    LIBNA2(B=>Y)
R( 15.27)    0.00                   LIBDFF(D)
   15.27ns
```

Area Report

****

Cell	Count	Area$ each	Area$ total
LIBLOG2	1	311.60	311.60
LIBNA2D	1	501.60	501.60
LIBINV4	1	501.60	501.60
LIBINV6	1	744.80	744.80
LIBOR4	3	874.00	2622.00
LIBAN4	4	760.00	3040.00
LIBINV2	14	273.60	3830.40
LIBBUF39	6	972.80	5836.80
LIBAN3	13	653.60	8496.80
LIBAN2	18	509.20	9165.60
LIBNO3S	32	558.60	17875.20
LIBNO4	32	608.00	19456.00
LIBFADD2	7	5312.40	37186.80
LIBINV	186	273.60	50889.60
LIBNO2	144	364.80	52531.20
LIBOR3	82	653.60	53595.20
LIBOR2	162	524.40	84952.80
LIBNA4	144	608.00	87552.00
LIBNA3	211	524.40	110648.40
LIBNA2	1429	380.00	543020.00
LIBDFF	392	1755.60	688195.20
Total	2883		1780953.60

Sub-modules	
crc32	635
ramtx	
reqfifo	
retfifo	

```
Block Summary:
   finite state machine at "hdlFilesDir/tx.v", 68:
        States: 44
        Encoding: one hot
```

4.4 Bloco RX

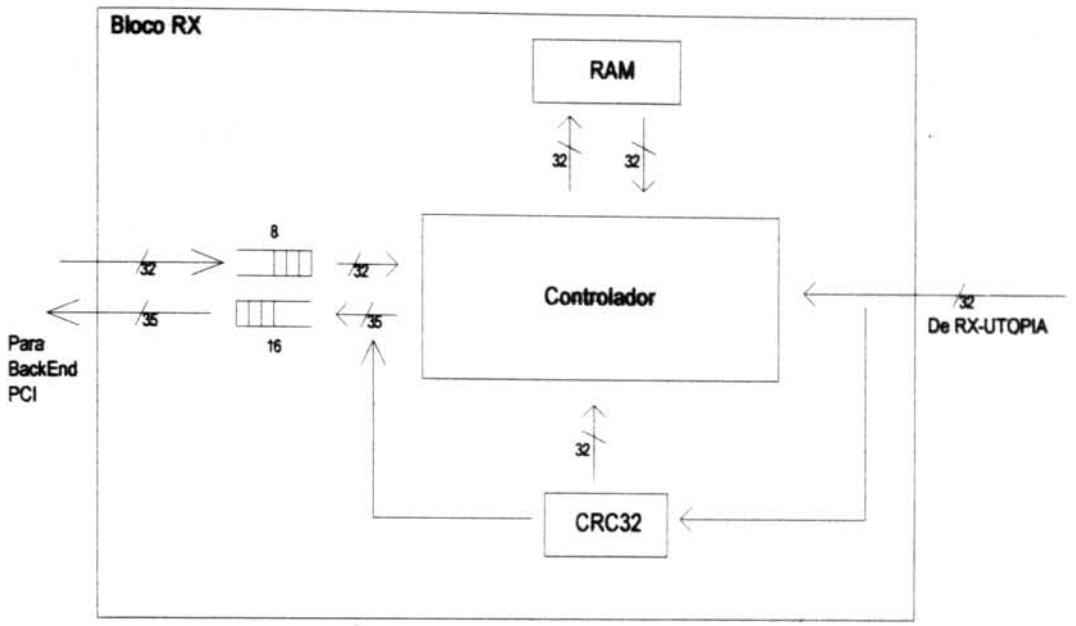


Figura 4.6: Bloco RX

O bloco que desempenha as funções de recepção tem uma constituição em tudo idêntica ao Bloco TX. É constituído por quatro sub-blocos: uma RAM, duas FIFOs, e um sub-bloco principal, que controla/implementa as operações de recepção. A figura 4.6 apresenta a sua estrutura interna.

O componente principal foi de igual modo completamente sintetizado, e modelizado de forma semelhante ao bloco TX. Contudo, a sua dimensão é superior (quase o dobro); em virtude das operações de *cache* implementadas, no acesso às estruturas de dados na memória principal (ver 3.4.1).

O funcionamento do controlador obedece ao fluxograma descrito na figura 4.7. As principais operações por ele realizadas no decorrer desse fluxo são:

i) O acesso às estruturas de dados de controlo e o armazenamento das células que compõem os CS-PDUs recebidos, na memória principal. Esta operação é realizada utilizando duas FIFOS. Uma FIFO de 16x35bit, para colocar os pedidos de leitura

e de escrita, e outra de 8x32bit para receber os resultados das operações de leitura. A FIFO de retorno dos dados de operações de leitura é maior de modo a albergar 12x32 *double words* (o corpo da célula a armazenar na memória principal).

ii) A geração dos endereços na memória principal no acesso às estruturas de dados de controlo e aos *buffers* onde são reconstruídos os CS-PDUs. Os endereços de acesso às estruturas de controlo, são calculados através dos valores presentes nos registos internos (acessíveis ao *software* de controlo pelo barramento PCI), na forma base, *offset*.

iii) O acesso à memória RAM para armazenamento temporário de dados das estruturas de controlo do processo.

iv) A leitura das células do bloco RX-UTOPIA através de uma FIFO (que se encontra à saída do bloco RX-UTOPIA). A transferência do corpo da célula é feita directamente de FIFO a FIFO.

v) A gestão do cálculo do CRC, actualizando os valores parciais de cada CS-PDU, com o cálculo realizado pelo componente CRC32. O cálculo do novo valor parcial é realizado transparentemente com a transferência do corpo da célula para a memória principal.

vi) A gestão da *cache/buffer* utilizado para optimizar o acesso às estruturas de dados. A memória RAM armazena até 4 descritores de *buffers* grandes, e 4 descritores de *buffers* pequenos, para utilização no processo de reconstrução dos CS-PDUs. Sempre que não se encontrar nenhum *buffer* adequado na memória RAM, a memória principal é acedida para obter novos *buffers* da 'Fila de *Buffers* Livres'. Esta solução reduz o número de acessos à memória principal, para obtenção de *buffers* livres, em cerca de 20%. De igual modo, a actualização da 'Fila de pacotes completos', na conclusão de qualquer CS-PDU, também é optimizada através da consolidação de até 4 actualizações num único acesso à memória principal. O número de acessos à memória principal para este efeito é reduzido em cerca de 25%.

vii) A actualização de registos internos com informação sobre o processo de recepção (células com endereço errado, etc).

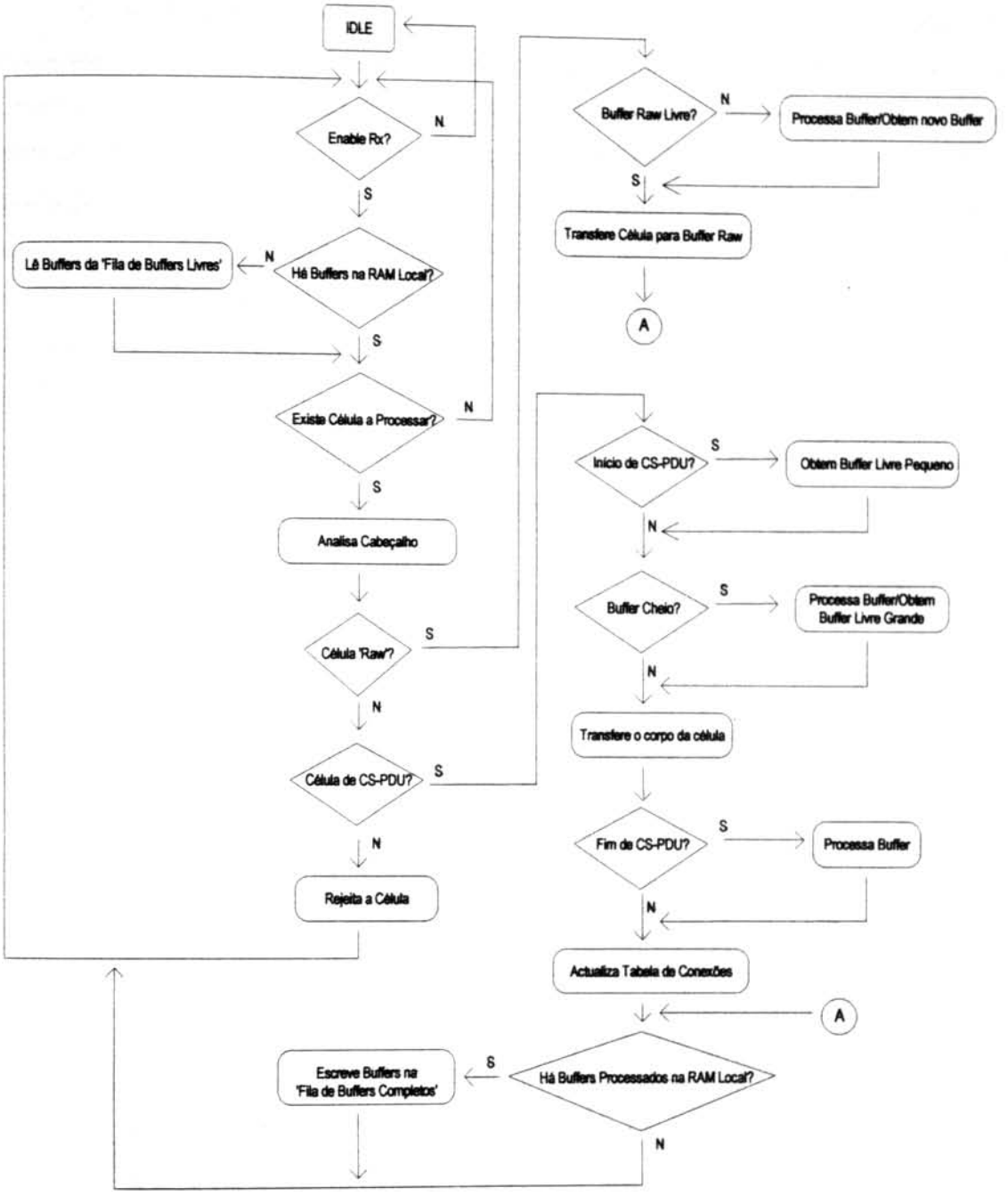


Figura 4.7: Fluxograma RX.

Este componente foi implementado numa máquina de estados com 81 estados. Utilizou-se o método 'one-hot' para codificar os estados uma vez que o número de registos a controlar era grande. A dimensão da funcionalidade deste componente suscitou a necessidade de controlar o processo de síntese em parâmetros, que não foi necessário controlar na síntese de outros componentes do HAL5.

Os problemas apresentaram-se ao nível dos tempos de propagação. Para os reduzir foi necessário especificar como deveriam ser sintetizadas certas estruturas, sendo o caso mais flagrante o dos blocos somadores. Em que foi necessário optar por uma estrutura do tipo *carry look ahead* de modo a reduzir o tempo de propagação no caminho mais longo. Esta mudança foi motivada pela constatação de que nos caminhos mais longos se encontravam blocos somadores. Outra estratégia adoptada foi a de orientar a ferramenta no sentido de evitar certas células da biblioteca, que introduziam atrasos mais elevados; neste caso a ferramenta tratou de a partir das equações lógicas, decompor a função a evitar e sintetizá-la a partir de outros elementos.

O componente de recepção foi sintetizado em 6067 *standard-cells*. Incluindo o bloco do CRC (635 células) foram necessárias no total 6702 *standard-cells* para implementar o bloco RX.

```
RX
****************************************************

Timing Report
****************************************************
Time unit:   1ns.      Precision:  10ps

************  Top Long Critical Paths  *************
Arrival      Inc  Fanout  Load   Cell(pin to pin)
------------ Long Critical Path 1 ------------------
R(  0.00)   0.00 <-- clock      LIBDFF(CK)
R(  1.25)   1.25    6    0.33   LIBDFF(CK=>Q)
F(  3.15)   1.90    1    0.06   LIBFADD2(A1=>COUT)
F(  3.72)   0.56    1    0.06   LIBFADD2(CIN=>COUT)
F(  4.37)   0.65    3    0.17   LIBFADD2(CIN=>COUT)
R(  4.67)   0.30    3    0.17   LIBNA2(B=>Y)
F(  4.84)   0.17    1    0.05   LIBINV(A=>Y)
R(  5.14)   0.30    3    0.17   LIBNA2(B=>Y)
F(  5.31)   0.17    1    0.05   LIBINV(A=>Y)
R(  5.61)   0.30    3    0.17   LIBNA2(B=>Y)
F(  5.77)   0.17    1    0.05   LIBINV(A=>Y)
R(  6.08)   0.30    3    0.17   LIBNA2(B=>Y)
F(  6.24)   0.17    1    0.05   LIBINV(A=>Y)
R(  6.54)   0.30    3    0.17   LIBNA2(B=>Y)
F(  6.71)   0.17    1    0.05   LIBINV(A=>Y)
R(  7.01)   0.30    3    0.17   LIBNA2(B=>Y)
F(  7.18)   0.17    1    0.05   LIBINV(A=>Y)
R(  7.48)   0.30    3    0.17   LIBNA2(B=>Y)
F(  7.65)   0.17    1    0.05   LIBINV(A=>Y)
R(  7.95)   0.30    3    0.17   LIBNA2(B=>Y)
F(  8.11)   0.17    1    0.05   LIBINV(A=>Y)
R(  8.42)   0.30    3    0.17   LIBNA2(B=>Y)
F(  8.58)   0.17    1    0.05   LIBINV(A=>Y)
R(  8.89)   0.30    3    0.17   LIBNA2(B=>Y)
```

```
F(  9.05)    0.17     1     0.05    LIBINV(A=>Y)
.....................
F( 15.19)    0.21     2     0.11    LIBINV(A=>Y)
R( 15.51)    0.33     2     0.11    LIBNA3(A=>Y)
F( 15.78)    0.26     1     0.06    LIBNA3(C=>Y)
R( 16.04)    0.27     1     0.03    LIBNA4(B=>Y)
F( 17.01)    0.30     1     0.17    LIBNA2(B=>Y)
R( 17.01)    0.00                   LIBDFF(D)
   17.01ns
```

Area Report

****

Cell	Count	Area$ each	Area$ total
LIBLOG2	1	311.60	311.60
LIBNA2D	3	501.60	1504.80
LIBINV4	3	501.60	1504.80
LIBINV6	3	744.80	2234.40
LIBOR4	4	874.00	3496.00
LIBMUX21	15	889.20	13338.00
LIBOAI222	7	946.20	6623.40
LIBAOI222	3	988.00	2964.00
LIBAN4	17	760.00	12920.00
LIBAN2	24	509.20	12220.80
LIBXOR	5	836.00	4180.00
LIBINV2	14	273.60	3830.40
LIBOAI221	4	836.00	3344.00
LIBBUF39	6	972.80	5836.80
LIBAN3	13	653.60	8496.80
LIBAN2	18	509.20	9156.60
LIBNO3S	52	558.60	29047.20
LIBNO4	53	608.00	32224.00
LIBFADD2	12	5312.40	63748.80
LIBINV	406	273.60	111081.60
LIBNO2	254	364.80	92659.20
LIBOR3	51	653.60	33333.60
LIBOR2	270	524.40	141588.00
LIBNA4	207	608.00	125856.00
LIBNA3	527	524.40	276358.80
LIBNA2	3481	380.00	928580.00
LIBDFF	614	1755.60	1077938.40
Total	6067		3004378.00

Sub-modules	
crc32	635
ramrx	
reqfifo	
retfifo	

```
Block Summary:
   finite state machine at "hdlFilesDir/rx.v", 96:
        States: 81
        Encoding: one hot
```

4.5 Interface PCI

4.5.1 *BackEnd* PCI

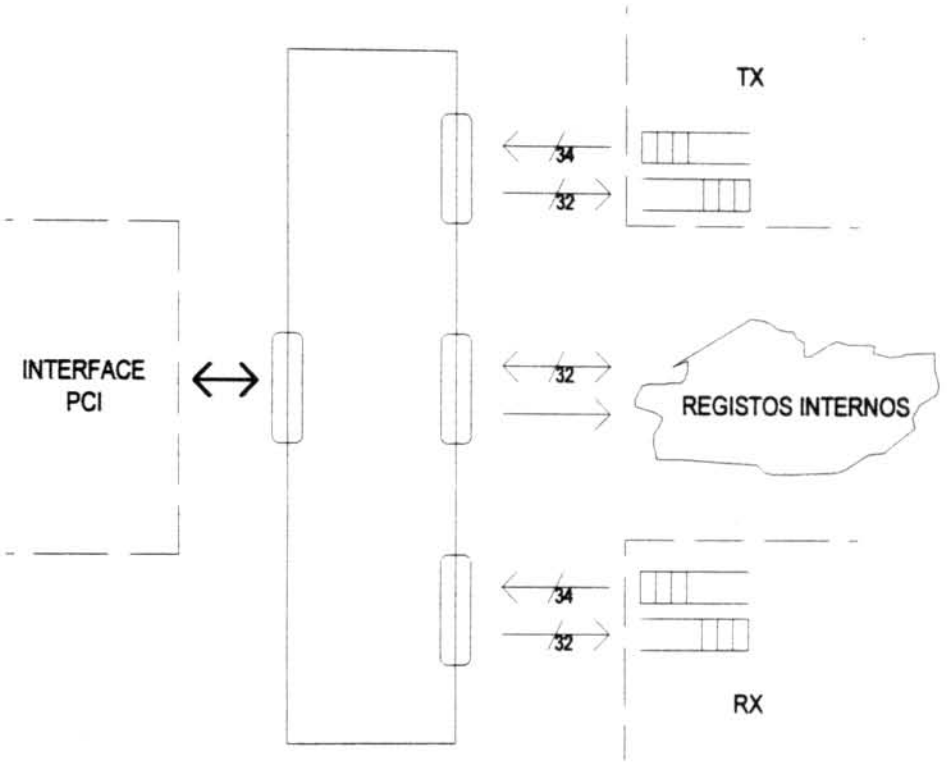


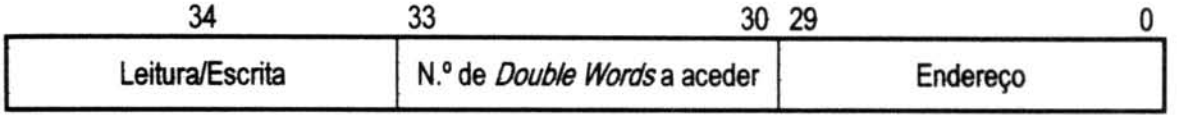
Figura 4.8: Ligação à interface PCI

Este componente desempenha a tarefa de fazer a ligação entre a interface PCI e o resto do circuito.

Os pedidos de acesso à memória por parte dos blocos TX e RX são atendidos com prioridades iguais, *mutliplexados* e colocados à interface PCI para que esta efectue o acesso o acesso à memória. Outra função deste componente é o de permitir o acesso aos registos internos do circuito. O *software* de controlo pode aceder a estes registos através da interface PCI, para proceder à configuração do HAL5, e também para aceder ao barramento utilitário (UTOPIA).

Um dos atributos deste componente consiste em fazer a ponte entre as duas velocidades de funcionamento existentes no circuito: 33MHz na interface PCI (este componente obtém o seu sinal de relógio do próprio barramento PCI), e 50MHZ no corpo principal do HAL5. A utilização de FIFOs assíncronas entre este componente e os blocos TX e RX, foi um dos métodos para atingir este objectivo. Além disso recorreu-se a um pequeno circuito assíncrono, que permitiu reduzir as possibilidades de meta-estabilidade nos pontos de contacto entre sinais de relógios diferentes.

Como se disse, os pedidos de acesso à memória por parte dos dois blocos são atendidos com igual prioridade. Estes, para efectuar um acesso à memória 'introduzem' o pedido na sua FIFO, utilizando o seguinte formato:



e, no caso de operações de escrita, as palavras a escrever na memória são colocadas imediatamente a seguir, utilizando os 32 bit menos significativos. Todos os acessos de leitura e escrita, quer por parte do bloco TX quer por parte do bloco RX, são realizados em palavras completas de 32 bit.

O pedido de acesso é apresentado à interface PCI e, o resultado dos pedidos de leitura é encaminhado para a FIFO do bloco (TX ou RX) que efectuou o acesso.

Sem incluir o pequeno circuito assíncrono que a seguir se descreve, os resultados da síntese deste componente foram:

```
BPCI
****************************************************

Timing Report
****************************************************
Time unit:   1ns.      Precision:  10ps

************  Top Long Critical Paths  *************
Arrival     Inc  Fanout  Load   Cell(pin to pin)
------------ Long Critical Path 1 -----------------
R(  0.00)   0.00 <-- clock       LIBDFF(CK)
.............
```

```
.............
F( 11.39)   0.17    1    0.05   LIBINV(A=>Y)
R( 11.70)   0.30    3    0.17   LIBNA2(B=>Y)
F( 11.96)   0.26    1    0.06   LIBNA3(C=>Y)
R( 12.23)   0.27    1    0.03   LIBNA4(B=>Y)
R( 12.23)   0.00                LIBDFF(D)
   12:23ns

Area Report
**********************************************************
```

Cell	Count	Area$ each	Area$ total
LOG2	1	311.60	311.60
INV2	3	273.60	820.80
OR3	2	653.60	1307.20
AOI222	1	988.00	889.20
AN2	3	509.20	1527.60
NO4	3	608.00	1824.00
NO2	7	364.80	2553.60
XOR	3	836.00	2508.00
OAI22	4	623.20	2492.80
OR2	9	524.40	4719.60
OAI221	1	836.00	836.00
NA4	8	608.00	4864.00
INV	16	273.60	4377.60
AOI21	7	520.60	3644.20
NA3	24	524.40	12585.60
AOI22	8	600.40	4803.20
OAI21	10	528.20	5282.00
NA2	43	380.00	16340.00
DFFS	19	1900.00	36100.00
DFFR	16	2097.60	33561.60
Total	188		141348.60

```
finite state machine at "hdlFilesDir/bpci.v", 27:
        States: 9
        Encoding: default
finite state machine at "hdlFilesDir/bpci.v", 71:
        States: 3
        Encoding: default
combinational block at "hdlFilesDir/bpci.v", 98: combinational
logic
```

Na comunicação entre as duas partes do circuito a funcionar a diferentes velocidades, dá-se a possibilidade de se proporcionarem situações de meta-estabilidade. As probabilidades de ocorrência são reduzidas, no entanto, optou-se por tentar eliminar o mais possível esse cenário. Grande parte do problema ficou resolvido com a utilização de FIFOs para a comunicação de dados entre os dois blocos. Estas, com duas portas de acesso (leitura e escrita) permitem acessos assíncronos, simultâneos ou não. O problema firmou-se nos sinais de estado das FIFOs: *full* e *empty*.

Estes sinais reflectem o estado da FIFO e são activados em função das operações de escrita e leitura. As máquinas de estado que controlam efectuam o acesso às FIFOs, tomam os sinais como suas entradas e, naturalmente, estes devem respeitar os tempos de *hold* e *setup* dos elementos de memória que mantém o estado presente da máquina.

Pela natureza das FIFOs o problema não se punha aquando da activação destes sinais (valor lógico 1), em que a transição do sinal respeita os tempos de *hold* e *setup*, mas sim na situação de uma transição do sinal para um valor lógico 0.

Procurou-se então uma forma de sincronizar estes sinais com o(s) sinal(is) de relógio. A utilização de um circuito síncrono, como por exemplo dois ou três flip-flops em série, podia resolver o problema, mas implicava a introdução de um atraso no efeito do sinal, que impediria uma acção atempada pelo bloco que estivesse a aceder à FIFO.

A solução escolhida consistiu na utilização de um pequeno circuito sequencial assíncrono no modo fundamental. Colocada de forma simplificada, a funcionalidade do circuito é semelhante à de um flip-flop *edge-triggered* em que o pino de *reset* assíncrono está ligado ao pino de entrada de dados. O circuito foi implementado com duas *latch* SR e 6 portas lógicas NOR.

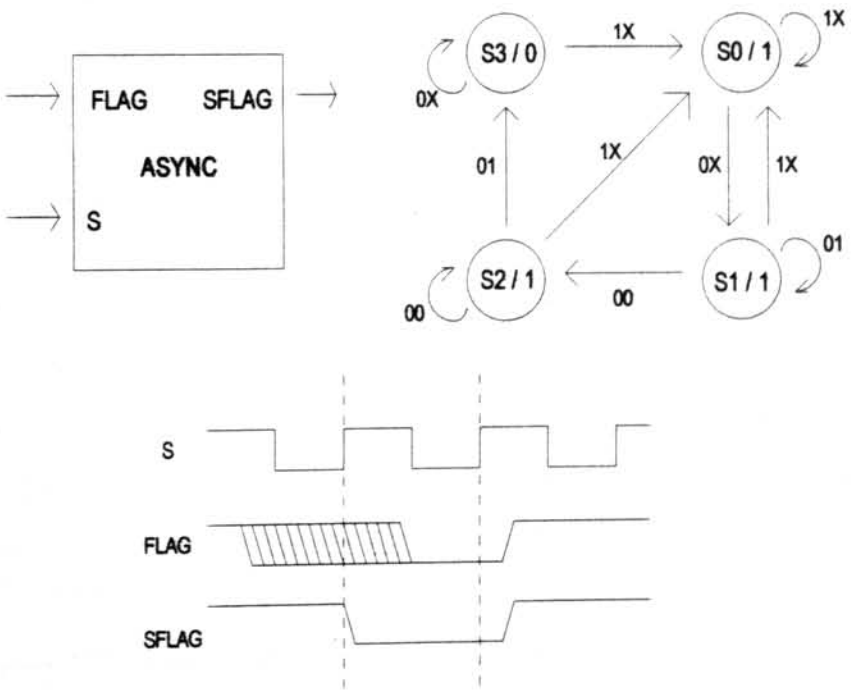


Figura 4.9: Circuito Assíncrono.

4.5.2 Bloco PCI

A estratégia para a implementação da interface PCI no HAL5, passou pela opção de obter o componente já desenvolvido testado e validado por terceiros (ver 3.2). Esta opção foi tomada quando a fabricação do HAL5 estava programada (e possível) para curto prazo, enquadrada no âmbito desta tese. A decisão de manter a linha de pensamento que levou aquela opção, apesar de a fabricação deixar de entrar no plano do projecto, prende-se com o desejo de ter presente neste trabalho, o tipo de soluções a considerar no projecto digital em situações reais (de fabricação).

Assim, uma vez que a obtenção desse componente não figuraria no âmbito desta tese, desenvolveu-se uma implementação da interface PCI, de modo a permitir uma visão completa do circuito. Contudo, dada a dimensão da completa especificação e validação necessária a este tipo de componente, optou-se por uma implementação extremamente simples. Apenas foram implementados os aspectos básicos ao funcionamento da interface. A sua simulação limitou-se a um mínimo de situações particulares, uma vez que para a completa validação e simulação do componente seria necessário a modelização do próprio barramento PCI e de outros dispositivos a ele ligados.

O funcionamento do HAL5 requer que ele assuma no barramento, quer o papel de *master* (dispositivo com capacidade para iniciar transações), quer o papel de *target* (dispositivo com capacidade de resposta a pedidos de transações que lhe são destinadas). No papel de *target* aceita acessos de dois tipos: acessos ao espaço de configuração (para permitir a leitura e escrita nos registos de configuração da interface PCI) e acessos ao espaço de I/O (para permitir a leitura e escrita nos seus registos internos de configuração). No desempenho do papel de *master*, o único tipo de transações iniciadas pelo HAL5 são de acesso ao espaço da memória principal.

O bloco PCI implementa apenas, como *master*, os acessos à memória, e como *target* os acessos ao espaço de configuração - associando portanto os registos internos de configuração do HAL5 com os do espaço de configuração da interface PCI (uma solução aceitável mesmo na norma PCI, dentro de certos parâmetros [28]).

A figura 4.10 apresenta a estrutura deste componente.

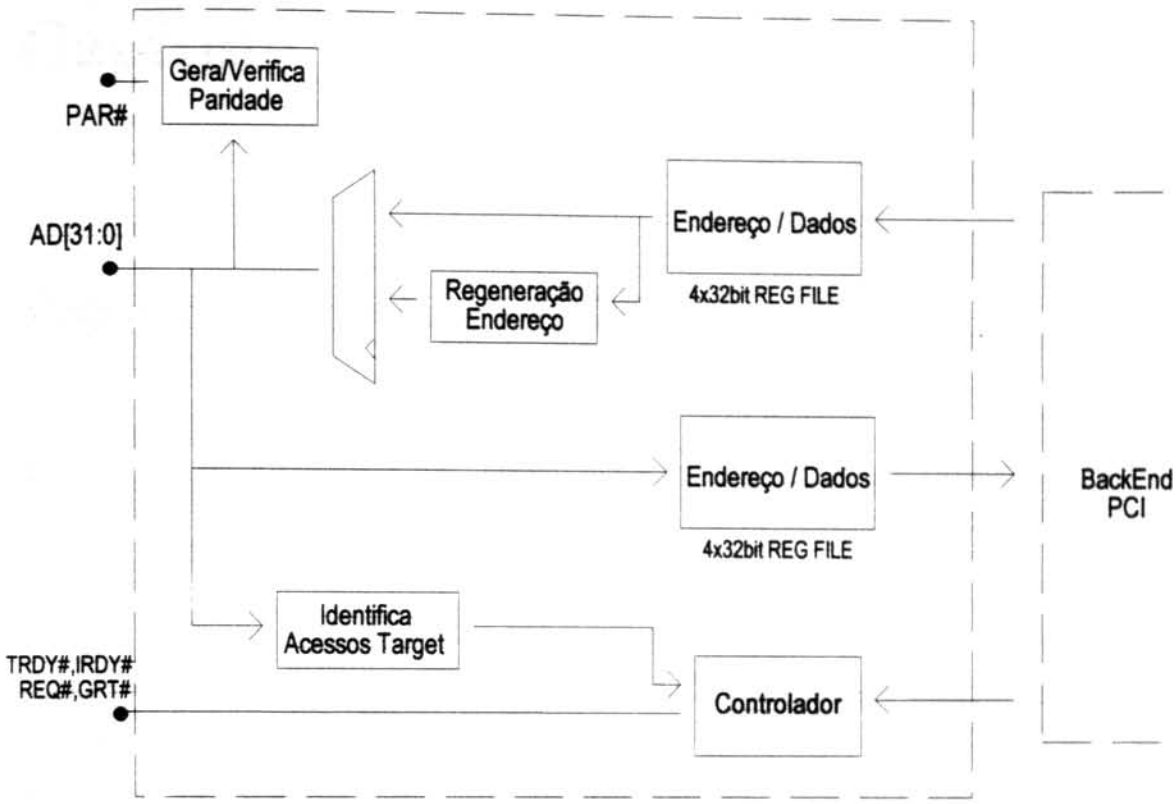


Figura 4.10: Bloco PCI

O sub-bloco que controla o funcionamento do componente é baseado uma versão simplificada e adaptada da implementação apresentada na 'Especificação do Barramento PCI, Versão 2.0' ([28], PCI *Special Interest Group*), para as máquinas de estado *Master/Target*.

A implementação deste componente implicou uma maior componente de captura esquemática para a sua criação, uma vez que pela natureza de alguns dos sinais do barramento PCI, é necessário 'registá-los' e o processo de síntese não facilita a colocação adequado dos registos. Na construção deste componente foram utilizadas cerca 2000 *standard-cells*, contudo, deve-se notar que esta foi uma implementação simplificada da interface.

Capítulo 5

Projecto Digital do HAL5 II

5.1 Projecto do CIAE

A implementação digital do HAL5 recorreu a um processo *semi-custom*, baseado em *standard-cells* da ATMEL/ES2. Nesta tecnologia a obtenção das máscaras para a fabricação do circuito envolve 3 fases distintas. O processo inicia-se com a adopção de uma biblioteca de *standard-cells* - da 'European Silicon Structures' (ES2), no caso presente - que contém as células base para a construção do CI. A biblioteca disponibiliza elementos básicos como, portas lógicas AND, OR, XOR, inversores, flip-flops, latches, somadores, pinos de I/O, etc. O *layout* físico interno de cada um destes elementos já vem definido de origem e para vários processos (1.5, 1.0, 0.7u), estando disponível ao utilizador, a caracterização lógica, eléctrica e física (física, no sentido de geometria exterior, pontos de acesso, etc.) de cada um dos elementos. Associada à biblioteca está também um gerador de *macro-cells*, que permite construir automaticamente, ROMs, RAMs, FIFOs, DPRAMs e multiplicadores.

Após a construção do circuito a partir dos elementos existentes na biblioteca, a fase seguinte consiste no 'Place' das *macro-cells* e das *standard-cells* que constituem o circuito. Esta fase consiste basicamente em definir o posicionamento das *macro-cells* e as regiões para a colocação das *standard-cells*, onde estas são colocadas em linhas paralelas horizontais (o comprimento das diferentes *standard-cells* da biblioteca ES2 é variável, mas a altura é constante a todas). Entre as linhas de *standard-cells* e entre as várias regiões definidas, localizam-se os canais para *routing*. É nesses canais que durante a terceira fase do processo o *Router* coloca as

linhas que interligam as células do circuito, as quais são exclusivamente realizadas em metal (o processo utilizado permite dois níveis de metal)*.

O processo descrito parte da *netlist* do circuito (obtida a partir do esquemático de topo, e que contém todos os componentes que formam o circuito, incluindo os *pads* de I/O) para o *layout* físico do circuito (*layout* que inclui o *routing* das *nets*, o posicionamento das células, mas não o *layout* interno destas). Este processo desenrola-se duma forma interactiva (quanto se deseje) e iterativa, com o recurso a ferramentas de CAD para 'Place&Route' (da Cadence, no caso presente), que, utilizando algoritmos avançados permitem conduzir o processo no sentido de se orientar a soluções obtidas de modo a satisfazer os requisitos de área, e velocidade (procurando obter uma disposição das células que minimize o comprimento das *nets* que as ligam).

A utilização de um processo *semi-custom* para a implementação do CI permite manter, em grande parte, as vantagens associadas à utilização dos métodos do tipo *custom-silicon*, nomeadamente, velocidades elevadas, áreas reduzidas e grande integração de funcionalidades. Permite fazê-lo sem incorrer nos elevados custos de desenvolvimento, prototipagem e teste, e longos tempos de desenvolvimento dos métodos *full-custom*, ou pelo menos, diminuindo em larga escala o custo e tempo de desenvolvimento associados.

Outra possibilidade seria a utilização de processos do tipo *field-programmable*. Em comparação com os processos do tipo *mask-programmable* (*full* ou *semi-custom*), estes permitem custos e tempos de desenvolvimento muito menores, mas em contrapartida seria difícil enquadrar um circuito com a elevada complexidade, elevada integração, e velocidades de operação, do HAL5, nos parâmetros de velocidade e dimensão característicos dos componentes desta tecnologia (convém notar que esta situação está em rápida evolução. Assim, a Xilinx já dispõe de FPGAs que permitem implementar interfaces com o barramento PCI a 33MHz).

O desenvolvimento de um ASIC é um processo complexo e envolve várias áreas de conhecimento, desde o nível de definição da arquitectura do CI até à obtenção do seu *layout*. Na realização deste trabalho procedeu-se a uma selecção nas várias tarefas e passos envolvidos

* Algumas *nets* são realizadas por *abutting*, ou seja, uma *net* entre duas células adjacentes pode ser realizada por simples contacto. Para permitir isto as células dispõem de pontos de acessos laterais, além dos pontos normais de acesso para o canal de *routing*.

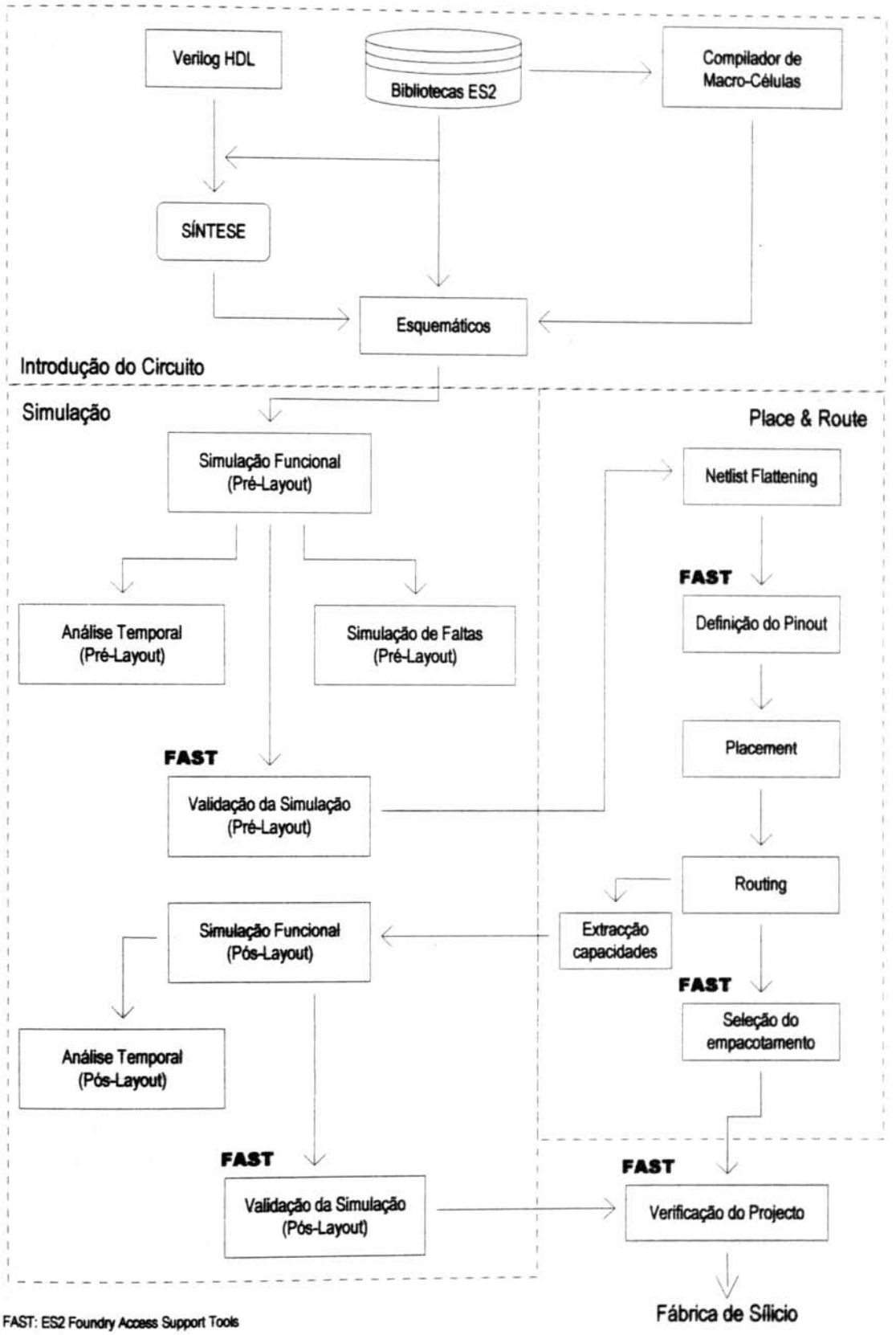
no processo de desenvolvimento, por forma a permitir incluir os aspectos mais 'importantes' dentro do quadro temporal disponível. O facto de a fabricação do circuito não constituir (pelo menos no âmbito desta tese) uma das metas do trabalho, motivou a colocação de um maior ênfase nas tarefas de concepção e síntese do circuito e, na obtenção do *layout* apenas com o intuito de obter uma informação genérica das características físicas do CI.

A figura 5.1 apresenta o fluxo de desenvolvimento associado ao '*ES2 Design Kit*' - a integração da biblioteca de células da ES2 nas ferramentas de CAD '*Cadence Design Framework 2*'.

A primeira fase do fluxo representado, consiste na introdução do circuito e na produção do esquemático de topo que contém a estrutura hierárquica do circuito. A realização desta tarefa passa pela compilação de *macro-cells* (RAMs, etc.), síntese de circuitos a partir de descrições realizadas em HDLs (ao nível comportamental, RTL, lógico, ou misto), e pela própria captura esquemática utilizando um editor gráfico.

A simulação do circuito é realizada em duas fases distintas: antes e depois do *layout*. Antes de se proceder ao *layout* do circuito, e em particular ao *routing*, não é possível saber (apenas estimar) o comprimento de cada *net* e o respectivo atraso de propagação. A simulação entra em linha de conta apenas com os atrasos intrínsecos de cada célula, e com os atrasos relativos à comutação das capacidades de entrada e saída das células associadas a cada *net*. A repetição das simulações após a obtenção das capacidades das linhas (simulação *loaded*), e a inclusão dessa informação no cálculo dos tempos de propagação é necessária para verificar se o circuito ainda se encontra dentro dos parâmetros de funcionamento especificados.

Além da 'simulação funcional' do circuito - realizada hierarquicamente, isto é, simulando cada componente isoladamente até à simulação global do circuito - procede-se à 'simulação de faltas' e à 'análise temporal' do circuito. A análise temporal do circuito 'percorre' cada *path* do circuito para determinar os tempos máximo e mínimo entre quaisquer dois pontos (também permite detectar violações nos tempos de *hold* e *setup*). A análise temporal é uma tarefa opcional, particularmente se o circuito foi realizado através de síntese, uma vez que nesse caso, este tipo de informação temporal é obtido como *by-product* da operação de síntese.



FAST: ES2 Foundry Access Support Tools

Figura 5.1: ES2 'Cadence Design Kit': *Design Flow*

A 'simulação de faltas' avalia a qualidade dos estímulos utilizados para o teste do circuito, no sentido de permitirem detectar falhas físicas no circuito. O circuito é simulado na presença de faltas e, a razão entre o número de faltas detectadas e o número de faltas simuladas, permite calcular a 'cobertura de faltas' do teste - uma aproximação da 'cobertura de defeitos'. Este é um parâmetro importante porque influencia grandemente a qualidade do produto final fabricado. O nível de circuitos fabricados com defeitos (e que atingem o utilizador) é dado por [1]: $DL = 1 - Y^{1-d}$, onde Y é o *yield* do processo de fabrico, ou seja a probabilidade do o produto fabricado estar livre de defeitos, e d a cobertura de defeitos do teste utilizado para verificar o circuito. A simulação de faltas permite realizar a geração de testes que garantam um nível de defeitos adequado.

A terceira actividade representada na figura 5.1 consiste no 'Place&Route' do circuito. Esta operação é realizada iterativamente até se obter um resultado final satisfatório (em termos de área, por exemplo). A participação do utilizador no processo pode ir até ao ponto de permitir o *routing* manual de parte das *nets*, sendo um bom exemplo o caso das linhas de alimentação e de distribuição do sinal de relógio (crítica).

As tarefas principais no Place&Route são:

i) *netlist flattening*: a estrutura hierárquica do circuito é aplanada a um único nível. Os algoritmos de *placement* não utilizam este tipo de informação estrutural (hierarquia) na distribuição das células; operam com um 'mar' de células e guiam-se pelo objectivo de diminuir, globalmente, as distancias de interligação das células.
ii) colocação das *macro-cells*.
iii) colocação das *standard-cells* e dos pinos de I/O.
iv) criação dos canais de *routing*.
v) *routing* global; os percursos de todas as *nets* existentes no circuito são 'distribuídos' pelos canais de *routing* por forma a equilibrar o nível de utilização dos mesmos.
vi) *routing* detalhado; a geometria de cada *net* em cada canal é determinada.
vii) extracção das capacidades; a capacidade de cada *net* é calculada em função do seu comprimento. Esta informação é utilizada para as simulações pós-*layout*.

Ao longo do fluxo representado na figura 5.1 encontra-se ainda, um conjunto de operações associadas na denominação FAST (ES2 *Foundry Access Support Tools*). A sua finalidade é verificar que as simulações realizadas, e certos aspectos do próprio CI, são compatíveis com os requisitos de verificação pós-fabricação na 'fábrica' ES2.

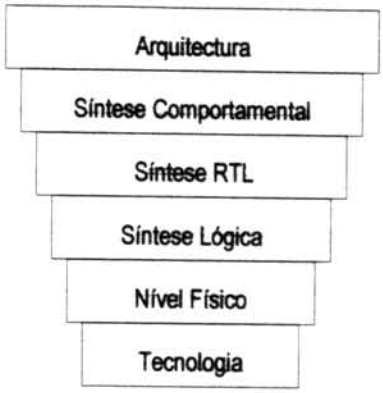


Figura 5.2: Especificação do sistema

Este trabalho centrou-se largamente no capítulo de especificação do sistema digital. O circuito foi na sua (quase) totalidade descrito em Verilog HDL, e sintetizado (*High Level Synthesis*).

A linguagem Verilog foi criada inicialmente como uma ferramenta de simulação/verificação. Posteriormente, o seu leque de utilização foi alargado com a sua integração em várias ferramentas de análise de sistemas, como ferramentas de 'simulação de faltas' e 'analise temporal', e como base para ferramentas de síntese (*Synergy*). De facto, as ferramentas utilizadas na realização deste trabalho, para a simulação, análise e síntese do circuito, basearam-se exclusivamente nesta linguagem.

Uma das qualidades da linguagem utilizada, é a possibilidade que esta oferece de descrever um sistema digital a vários níveis de abstracção. Desta forma permite acompanhar todas as fases de especificação do sistema, com uma descrição do circuito no nível de abstracção mais adequado a cada fase. Assim, num primeiro ciclo de concepção do sistema, é possível trabalhar com descrições do tipo *behavioural* que captam a essência do funcionamento do circuito, sem entrar nos detalhes da sua implementação; e, nas fases de implementação, descer a níveis de representação 'inferiores' que descrevam a estrutura do circuito.

Naturalmente a linguagem permite a utilização de estruturas hierárquicas, de forma a permitir a modelização de circuitos complexos, bem como de construções lógicas avançadas (if...then...else, while, wait, tasks, functions, for, etc). As semelhanças com qualquer linguagem de programação de *software* são de tal forma, que se torna difícil estabelecer uma fronteira definida entre *software/hardware*.

A possibilidade de, a partir de uma linguagem de descrição de *hardware*, especificar (programar), simular (*debug*), e sintetizar (compilar) um circuito digital, como se de *software* se tratasse, mostrou-se extremamente positiva.

Os benefícios fizeram-se sentir particularmente ao nível do esforço de especificação do circuito, e da sua validação funcional/certificação. No entanto, como contraponto, é de destacar que a qualidade do resultado final é fortemente dependente do desempenho das ferramentas de síntese utilizadas, bem como da maior ou menor aplicação do utilizador, em orientar o seu (das ferramentas) esforço na direcção adequada. E, também, a partir de certa dimensão, demonstra-se difícil estabelecer uma relação fácil e directa entre a descrição dos circuitos e a sua implementação, o que por vezes limita a interpretação das simulações (da implementação), à análise dos sinais de I/O de topo, do circuito (ou sub-circuito).

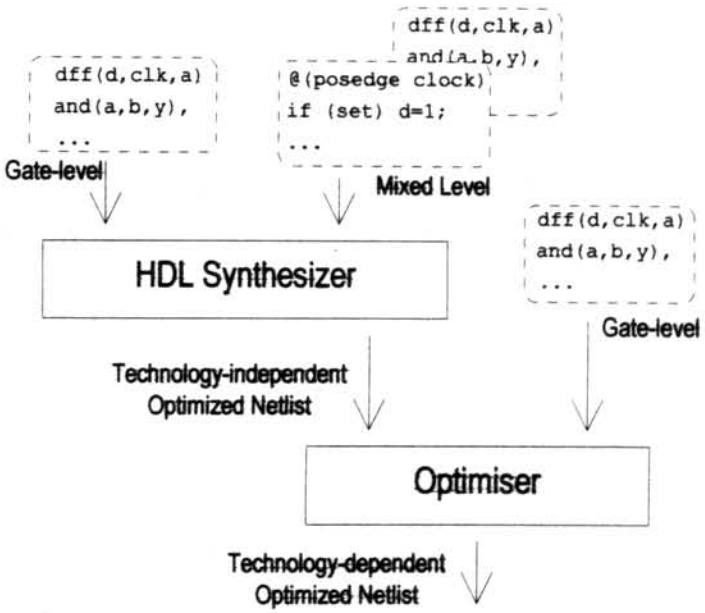


Figura 5.3: ***Synergy Synthesis Tool***

A ferramenta de síntese utilizada é composta pelo *'HDL Synthesiser'* e pelo *'Optimiser'* (figura 5.3). O primeiro sintetiza *netlists*, optimizadas independentemente da tecnologia de silício utilizada, a partir de descrições do nível lógico, RTL, ou misto. O *'Optimiser'* efectua o mapeamento da *netlist* nos componentes existentes na tecnologia de silício utilizada e realiza nova optimização, desta vez *technology-dependent*. As optimizações podem ter como *'goal'* a diminuição da área ou a diminuição dos atrasos na propagação dos sinais. De igual modo, é possível influenciar todo o processo e o resultado obtido, especificando um conjunto de condições/requisitos, que vão desde o tipo de estrutura a utilizar para implementar somadores, ao tipo de codificação a utilizar na codificação de estados em FSMs, ao nível de partilha de recursos, etc.

A experiência adquirida no desenrolar do trabalho demonstrou, infelizmente, que mesmo para elementos de dimensões médias, a síntese a partir de descrições comportamentais implica a utilização de recursos apreciáveis em termos de tempo de CPU e de memória necessária. De tal modo que para permitir efectuar a síntese de certos componentes foi necessário descer o nível de abstracção utilizado para descrever certos componente (o que por outro lado, permitiu manter uma maior proximidade entre a descrição do circuito, e a sua implementação sintetizada, facilitando a análise do mesmo).

Em particular, foi necessário representar explicitamente as descrições das máquinas de estado.

Uma das facilidades da HDL utilizada é a de poder descrever o funcionamento de um determinado circuito síncrono, através de uma sequência de passos, sincronizados com o sinal de relógio. A partir desta descrição (dita implícita) a ferramenta de síntese é capaz de extrair quais os estados necessários à implementação do circuito, gerar as funções de transição de estado, e das saídas. A listagem seguinte apresenta um excerto modelizado por este método:

```
if ((ncells>0)||(!f_full))
   begin
      if (f_full) ncells<=ncells-1;
      else ncells<=4;
      f_wr<=1;
      @(posedge clock)
        f_wr<=0;
      wait(valid && !u_empty);
      @(posedge clock)
        f_wr<=1;
```

```
        @(posedge clock)
          f_wr<=0;
        wait(valid && !u_empty);
        @(posedge clock)
          f_wr<=0;
        repeat(48) F_WRITE; // F_WRITE Task
    end
@(posedge clock) begin
```

Este estilo de modelização facilita a descrição do circuito porque, em parte, é possível fazê-lo duma forma mais sintética e rápida, sendo mais evidente qual a funcionalidade implementada pelo circuito. A representação explícita do circuito tornou-se mais complexa por implicar a definição de cada estado e todas as condições de transição de estado. Um exemplo:

```
case (STATE)
.......
2:if (fullREQ) state<=2;
     else begin
           dataREQ<={5'b00000,TLB_BASE,
                     cacheTLBoff[11:2],2'b00};
           wrREQ<=1;
           state<=3;
          end
   3:begin
      wrREQ<=0;
      cacheTLB<=dataRET;
      cacheTLBoff[11:2]<=cacheTLBoff[11:2]+1;
      state<=4;
     end
   4:if (queue==255) begin
                      cacheTLBoff[1:0]<=cacheTLBoff[1:0]+1;
                      state<=5;
                     end
     else begin
           addRAM<=queue;
           rdRAM<=1;
           state<=9;
          end
// SendIdle
   5:if (fullFIFO) state<=5;
     else begin
           dataFIFO<=32'b00000001000000000000000000000000;
           wrFIFO<=1;
           state<=6;
          end
   6:begin
      wrFIFO<=0;dataFIFO<=0;counter<=1;
      state<=7;
```

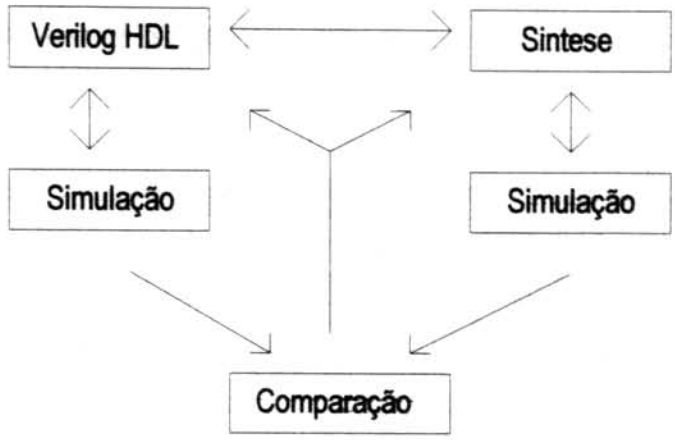


Figura 5.4: Fluxo de desenvolvimento

A figura 5.4 descreve o processo iterativo de síntese, simulação e comparação associado a este método de desenvolvimento. O processo desenvolve-se de forma a garantir a equivalência de funcionamento entre ambas as representações do circuito, e de forma a aproximar o resultado final dos valores desejados em termos de área e velocidade.

Em primeiro lugar as simulações funcionais realizaram-se sobre as descrições de alto nível dos componentes do circuito. Após a constatação da correcção dos mesmos, e que se cumprem com as especificações definidas, o passo seguinte consiste em sintetizar o componente. Uma nova simulação(ões) permite constatar se a implementação sobre os elementos da biblioteca utilizada para a síntese, ainda cumpre a especificação do circuito (sub-circuito). Por outro lado permite avaliar da qualidade do circuito produzido pela síntese e, se necessário, realizar novas iterações.

A simulação funcional de certos elementos mais complexos revelou-se por vezes tarefa difícil, em virtude de a partir de certa dimensão não ser trivial estabelecer um paralelo entre a descrição de alto nível e o esquemático obtido da síntese. No entanto, dois factores contribuíram para facilitar a realização desta tarefa. Por uma lado, a realização das simulações hierarquicamente a partir dos blocos de base em direcção ao topo, permitiu ir abstraindo da complexidade interna dos blocos. O método comum para a simulação de circuitos.

A característica particular do HAL5 que também facilitou (à excepção da interface PCI) as simulações, foi o facto de a maioria das interfaces entre os blocos se realizar utilizando FIFOs. Desta forma tornou-se mais fácil a aplicação de estímulos (células ATM, resultados de acessos a estruturas de dados), através destas portas padrão.

5.2 Validação da Arquitectura

A arquitectura adoptada para o HAL5 depende fortemente do desempenho do barramento PCI. A opção de utilizar a memória principal do sistema para colocar os CS-PDUs a transmitir/recebidos, e mais ainda, para colocar as próprias estruturas de dados, contém algumas contrapartidas. A necessidade de aceder com elevada frequência à memória principal afecta o desempenho do circuito em sintonia com a maior ou menor disponibilidade do barramento.

Esta disponibilidade traduz-se no tempo de latência do barramento, ou seja o tempo de espera até que seja atribuído o direito para o utilizar. Para que seja possível manter uma ligação ATM a 310Mbits (velocidade agregada) é necessário que este tempo de latência seja inferior a 1us, valor que foi estimado a partir da informação do tempo médio disponível para processar cada célula ATM (2.8us) e das imposições colocadas pela arquitectura do circuito.

A validade das opções tomadas na definição da arquitectura do HAL5 implica a necessidade de, alguma forma, verificar o comportamento do circuito em funcionamento. Uma simulação funcional para este fim, seria complexa de realizar pela necessidade de modelar uma série de parâmetros (tráfego no barramento PCI, etc) de forma a emular uma situação 'real' plausível. O que pode ser feito até certo grau. O ideal seria então dispor de um protótipo (real) para poder analisar o desempenho do circuito e proceder às revisões necessárias para produzir a versão final do circuito.

A abordagem seguida foi então a de simular um modelo abstracto do circuito em *software*, entrando em linha de conta com os parâmetros que regem o seu funcionamento. Sendo que os parâmetros mais importantes considerados são: o tempo de latência no barramento PCI, a taxa de ocupação do barramento PCI, e as taxas de transferência de recepção e de transmissão.

O programa desenvolvido em linguagem Pascal, leva em linha de conta a estrutura dos fluxos de recepção e transmissão no circuito, bem como os processos e velocidades associados a cada um deles (figura 5.5). Permite observar o número de células perdidas/ms na

recepção por *overflow* da FIFO, e o número de células/ms que são enviadas com atraso na transmissão, por não se conseguir manter a taxa de transferência estabelecida.

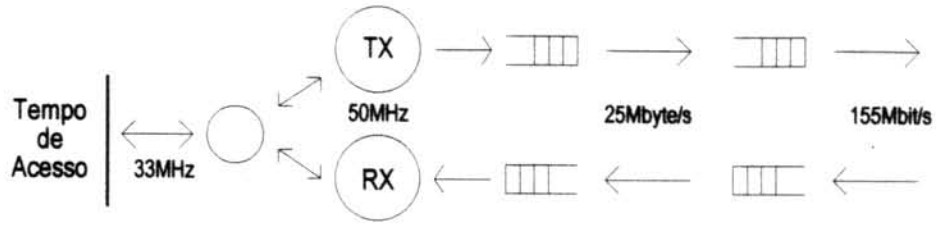


Figura 5.5: Modelo

Os gráficos seguintes (5.6, 5.7, 5.8) apresentam os resultados obtidos para diversos tempos de latência no barramento, entre 0.7 e 1.0us, e para três situações pertinentes. O primeiro cenário corresponde a uma largura de banda de 155Mbit/s simultaneamente em ambos os sentidos. Os outros dois casos apresentam situações assimétricas, com uma maior largura de banda no sentido de recepção (situação vulgar). Em todos os casos se assumiu uma taxa de ocupação do barramento PCI de 80% (quando o barramento está *idle*, o tempo de espera para aceder ao barramento é desprezável).

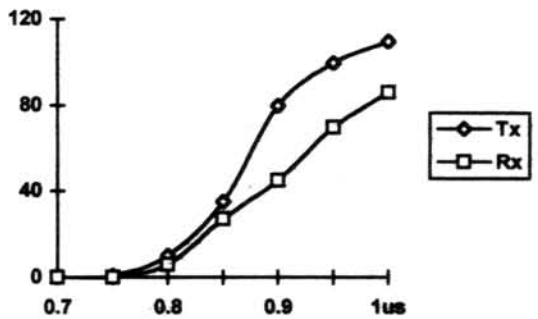


Figura 5.6: TX155/RX155

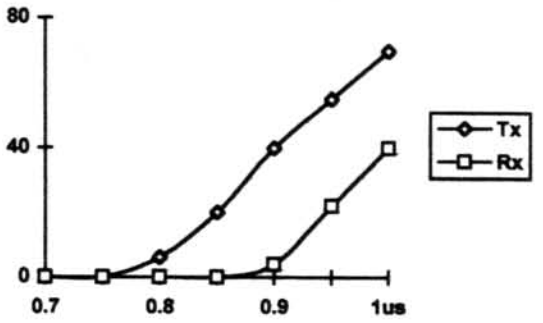


Figura 5.7: TX100/RX155

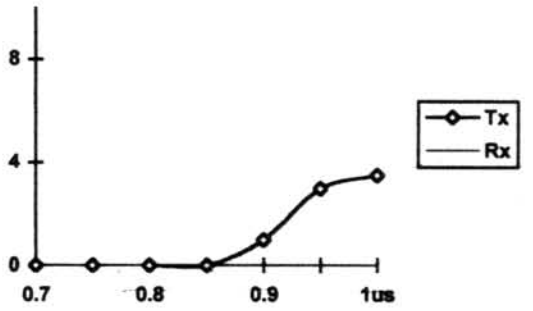


Figura 5.8: TX56/RX100

Os resultados obtidos confirmam a necessidade de um tempo de latência no acesso ao barramento PCI inferior a 1us. É também possível de constatar que para ligações à velocidade de 155Mbit/s simultânemente em ambos os sentidos, este tempo deve ser relativamente bastante inferior, na ordem dos 0.75ns. Caso contrário o circuito não poderá manter a integridade da ligação.

A configuração do barramento de PCI em termos de desempenho é uma questões com duas faces. É possível limitar o tempo de latência, mas ao mesmo tempo limita-se a largura de banda máxima no barramento. Por outro lado para dispor de maior largura de banda é necessário aumentar os tempos de latência.

Um valor para este parâmetro na ordem de 1us é um valor de compromisso aceitável para o desempenho do barramento. No entanto, à medida que este valor é diminuído (pela

configuração dos elementos que utilizam o barramento, e pelo dispositivo que arbitra o acesso ao barramento), a largura de banda diminui de forma a afectar negativamente a capacidade do barramento.

Uma hipótese para permitir um tempo de latência menos exigente, seria o de utilizar um integrado de memória estática a colocar na mesma placa adaptadora que o HAL5, onde este poderia colocar as estruturas de dados de controlo. A opção pela não utilização desta hipótese, prendeu-se com os seguintes motivos:

i) A utilização de memória local encarece a implementação do terminal ATM.
ii) A situação de uma ligação a 155Mbit/s bidireccional durante muito tempo não será comum.
iii) Os protocolos de comunicação das camadas superiores (a AAL) dispõem de processos de recuperação da comunicação.

O objectivo principal no desenvolvimento de um sistema com o menor custo de implementação possível, de forma a permitir o acesso dos utilizadores, em larga escala, ao tipo de tecnologia de comunicação que este circuito oferece.

Capítulo 6

Conclusão

O trabalho que aqui se apresentou conclui o caminho iniciado na disciplina de Seminário/Projecto, da Licenciatura em Eng. Electrotécnica e de Computadores, onde se desenvolveu a primeira versão do HAL5.

O sistema agora proposto prima por apontar a uma solução simples e eficiente na implementação de um terminal ATM. Num único circuito integrado (com cerca de 50mm^2 de área) implementa-se a funcionalidade das camadas ATM e AAL, permitindo a construção de terminais de acesso à rede ATM acessíveis ao grande público - daí a opção de não utilizar memória local. Procurou-se de igual modo seguir as principais normas estabelecidas no universo das telecomunicações e da micro-electrónica - ao optar por padrões como a interface PCI e a interface UTOPIA.

De facto, apesar de já na época do projecto da primeira versão do HAL5, se sentir a necessidade de desenvolver tecnologia que suportasse a adopção generalizada da tecnologia ATM, o cenário de lá até ao presente caracterizou-se por um constante adiar da mudança, sempre na expectativa de estar para breve a reunião das condições mais adequadas.

A situação deverá alterar-se a breve prazo. Começam já a desaparecer alguns entraves, como o elevado custo do equipamento, ou a divergência nas normas utilizadas. O momento que se vive actualmente ao nível das telecomunicações levará forçosamente a uma situação de rotura. O trabalho aqui apresentado, pretende constituir uma pequena parte do esforço que se vem a realizar, no sentido de dotar a sociedade a que pertencemos com a tecnologia necessária à expansão das suas actividades sociais e económicas.

Como se disse algures, um bom exemplo desta necessidade é a InterNet. A largura de banda instalada é cada vez mais utilizada até ao limite. E surgem novos conceitos que deixam

antever alterações importantes, mas que estão dependentes das capacidades de comunicação. Como é o caso do *Network Computer*, vulgo NC, cujo conceito radical de funcionamento depende totalmente da disponibilidade de tecnologias de comunicação com as características do ATM.

Para concluir, apesar de não se ter fabricado o HAL5 II, a especificação do circuito, da sua arquitectura, a avaliação das suas características físicas e duma possível implementação constituem um bom ponto de partida, para a eventual decisão de o fabricar ou no projecto de um outro circuito integrado que implemente as suas funções.

Bibliografia

1 Abramovici, M., Breuer, Melvin A., Friedman, Arthur D., Digital Systems Testing and Testable Design, Computers Science Press, 1990
2 ATM-Forum: ATM UNI-Specification (V3.0), 1993.
3 Bocker: ISDN - The Integrated Services Digital Network - Concepts, Methods, Systems. Springer-Verlag, 1992.
4 Byte Magazine: Playing the ATM Card, 1996.
5 Cadence, Design FrameWork II User Guide.
6 Cadence, Design Planner - Cell3 Ensemble.
7 Cadence, HDL Synthesizer and Optimizer.
8 Cadence, Simulation Environment User Guide.
9 Cadence, VERILOG-XL Interface User Guide.
10 Cadence, VERILOG-XL Reference Manual.
11 Cadence, Waveform Display User Guide.
12 Canas Ferreira, J., Castro, R., de Carvalho, H., Mota, P., Estácio, P.: 'A single chip Implementation of ATM AAL5 Packet Reassemble'. X Systems and Integrated Circuits Design Congress, SICD/DCIS'95, Saragoça, Nov/95.
13 Ercegovac, Milos D., Lang, Tomás: Digital Systems and Hardware/Firmware Algorithms, John Wiley & Sons, 1985.
14 ES2, Process-Independent Library Databook.
15 Fabricius, Eugene D., Introduction to VLSI Design, McGraw-Hill, 1990.
16 Hall, Douglas V., Microprocessors and Interfacing - Programming and Hardware, McGraw-Hill, 1992.
17 Händel, Manfred, Stefan Schröder: ATM Networks - Concepts, Protocols, Applications.
18 ITU-T: COM XVIII-228-E, Geneva, 1984.
19 ITU-T: Recomendação I.113: 'Vocabulary of Terms for Broadband Aspects of ISDN', Rer. 1, Geneva, 1991.
20 ITU-T: Recomendação I.121: 'Broadband Aspects of ISDN".
21 ITU-T: Recomendação I.150: 'B-ISDN ATM Functional Aspects'. Rev. 1, Geneva, 1993.
22 ITU-T: Recomendação I.211: 'B-ISDN Service Aspects'. Rev. 1, Geneva, 1993.
37 ITU-T: Recomendação I.311: 'B-ISDN General Network Aspects'. Geneva, 1993.
23 ITU-T: Recomendação I.321: 'B-ISDN Protocol Reference Model and its Application'. Geneva, 1991.
24 ITU-T: Recomendação I.361: 'B-ISDN ATM Layer Specification'. Rev. 1, Geneva, 1993.
35 ITU-T: Recomendação I.363: 'B-ISDN ATM Adaptation Layer (AAL) Specification', Geneva, 1993.
36 ITU-T: Recomendação I.371: 'Trafic Control and Congestion Control in B-

ISDN', Geneva, 1993.
25 ITU-T: Recomendação I.610: 'B-ISDN Operation and Maintenance Principles and Functions'. Rev. 1, Geneva, 1993.
26 Keiser, G.: Local Area Network, McGraw-Hill,1989.
27 Milenkovic, Milan, Operating Systems - Concepts and Design, McGraw-Hill, 1987.
28 PCI Special Interest Group, PCI Local BUS Specification, Rev. 2.0, 1993.
29 Pucknell, Douglas A., Eshraghian K.: Basic VLSI Design, Silicon Systems Engineering Series, 3rd Ed., 1994.
30 Sandige, Richard S.: Modern Digital Design, McGraw Hill, 1990.
31 Shanley, T., Anderson, D., PCI System Architecture, 3rd Ed., Addison-Wesley Publishing Company.
32 Taub, Herbert, Digital Circuits and Microprocessors, McGraw-Hill,1982.
33 Thomas, Donald E., Moorby, P., The Verilog Hardware Description Language, Kluwer Academic Publishers, 1991.
34 Trembley, Jean-Paul, Sorenson, Paul G., An Introduction to Data Structures with Applications, McGraw-Hill, 1984.

Apêndice A

```
// Verilog HDL for "vparts", "txutopia" "_functional"

module txutopia(clock,rst,
                data,full,wr,
            u_soc,u_enb_,u_full,u_clk,u_data);

input       clock,rst,u_full,wr;
input [31:0]      data;
output            u_soc,u_enb_,u_clk,full;
output      [7:0] u_data;

reg         f_rd,u_soc,u_enb_,u_clk;
reg   [5:0] counter;

txutopia_fifo f1(.reset(rst),.ckin(wr),.ckout(f_rd),.din(data),
                 .full_flag(full),.empty_flag(f_empty),.dout(u_data));

always @(rst)
 if (rst) begin
           assign f_rd=0;
           assign u_enb_=1;
           assign u_clk=0;
           assign u_soc=0;
           assign counter=0;
           disable main1;
          end
 else begin
       deassign f_rd;
       deassign u_enb_;
       deassign u_clk;
       deassign u_soc;
       deassign counter;
      end

always @(posedge clock)
 begin : main1
  if (rst) disable main1;
  u_clk=1;
  if (u_full || f_empty)
   @(posedge clock) begin
                     u_enb_=1;u_clk=0;
                    end
  else begin
        f_rd=(counter!=4);
        @(posedge clock)
        u_clk=0;u_enb_=0;f_rd=0;
        u_soc=(counter==0);
        if (counter<52) counter=counter+1;
        else counter=0;
       end
 end
endmodule
```

```
// Verilog HDL for "vparts", "rxutopia" "_functional"

module rxutopia(clock,rst,
            data,rd,empty,
            u_soc,u_enb_,u_empty,u_clk,u_data);

input       clock,rst,u_empty,u_soc,rd;
output            u_enb_,u_clk,empty;
output      [31:0]      data;
input [7:0] u_data;

reg         f_wr,u_enb_,u_clk,valid;
reg     [2:0]   ncells;

rxutopia_fifo
f1(.reset(rst),.ckin(f_wr),.ckout(rd),.din(u_data),.full_flag(f_full),.em
pty_flag(empty),.dout(data));

always @(rst)
 if (rst) begin
           assign f_wr=0;
           assign u_enb_=1;
           assign u_clk=0;
           assign valid=0;
           assign ncells=4;
           disable main1;
           disable main2;
          end
 else begin
       deassign f_wr;
       deassign u_enb_;
       deassign u_clk;
       deassign valid;
       deassign ncells;
      end

always @(posedge clock)
 begin : main1
  if (rst) disable main1;
  u_enb_=u_empty;u_clk=0;valid=0;
  @(posedge clock)
  u_clk=1;valid=!u_enb_;
 end

task F_WRITE;
 begin
  wait(valid && !u_empty);
  @(posedge clock) f_wr=1;
  @(posedge clock) f_wr=0;
 end
endtask

always @(posedge clock)
 begin : main2
  if (rst) disable main2;
  if (valid && !u_empty && u_soc)
   if ((ncells>0)||(!f_full))
     begin
      if (f_full) ncells=ncells-1;
```

```
      else ncells=4;
      f_wr=1;
      @(posedge clock) f_wr=0;
      repeat(3) F_WRITE;
      wait(valid && !u_empty);
      @(posedge clock) f_wr=0;
      @(posedge clock) f_wr=0;
      repeat(48) F_WRITE;
     end
 end
endmodule

// Verilog HDL for "vparts", "crc32" "_functional"

module crc32(a,b,c) ;
input [31:0]      a,b;
output      [31:0]      c;
reg   [31:0]      c;

always @(a or b)
 c=xor32(a,b);

function [31:0] xor32;
 input [31:0] d,r;
 begin
  xor32[31]=d[23] ^ d[24] ^ d[30] ^ d[27] ^ d[28] ^ d[29] ^ d[25] ^ d[9]
^ d[8] ^ d[31] ^ d[15] ^ d[11] ^ d[5] ^ r[28] ^ r[24] ^ r[23] ^ r[29] ^
r[25] ^ r[27] ^ r[9] ^ r[15] ^ r[8] ^ r[11] ^ r[5] ^ r[31] ^ r[30];
  xor32[30]=d[22] ^ d[8] ^ d[23] ^ d[26] ^ d[30] ^ d[27] ^ d[29] ^ d[24]
^ d[14] ^ d[7] ^ d[28] ^ d[10] ^ d[4] ^ r[27] ^ r[23] ^ r[22] ^ r[14] ^
r[24] ^ r[7] ^ r[26] ^ r[8] ^ r[4] ^ r[29] ^ r[30] ^ r[28] ^ r[10];
  xor32[29]=d[21] ^ d[25] ^ d[23] ^ d[13] ^ d[22] ^ d[26] ^ d[29] ^ d[7]
^ d[31] ^ d[6] ^ d[27] ^ d[28] ^ d[9] ^ d[3] ^ r[6] ^ r[23] ^ r[21] ^
r[22] ^ r[25] ^ r[31] ^ r[26] ^ r[7] ^ r[13] ^ r[28] ^ r[3] ^ r[29] ^
r[27] ^ r[9];
  xor32[28]=d[20] ^ d[12] ^ d[22] ^ d[21] ^ d[24] ^ d[6] ^ d[25] ^ d[28]
^ d[27] ^ d[30] ^ d[5] ^ d[26] ^ d[8] ^ d[2] ^ r[24] ^ r[5] ^ r[20] ^
r[26] ^ r[6] ^ r[21] ^ r[2] ^ r[25] ^ r[22] ^ r[27] ^ r[28] ^ r[30] ^
r[12] ^ r[8];
  xor32[27]=d[19] ^ d[20] ^ d[11] ^ d[27] ^ d[21] ^ d[23] ^ d[24] ^ d[26]
^ d[29] ^ d[5] ^ d[4] ^ d[25] ^ d[7] ^ d[1] ^ r[21] ^ r[23] ^ r[19] ^
r[7] ^ r[27] ^ r[20] ^ r[24] ^ r[25] ^ r[5] ^ r[26] ^ r[29] ^ r[4] ^ r[1]
^ r[11];
  xor32[26]=d[18] ^ d[19] ^ d[22] ^ d[4] ^ d[26] ^ d[28] ^ d[31] ^ d[23]
^ d[20] ^ d[24] ^ d[10] ^ d[3] ^ d[25] ^ d[6] ^ d[0] ^ r[22] ^ r[3] ^
r[6] ^ r[18] ^ r[23] ^ r[28] ^ r[10] ^ r[25] ^ r[26] ^ r[4] ^ r[20] ^
r[19] ^ r[0] ^ r[31] ^ r[24];
  xor32[25]=d[22] ^ d[17] ^ d[3] ^ d[21] ^ d[31] ^ d[11] ^ d[15] ^ d[19]
^ d[18] ^ d[29] ^ d[28] ^ d[8] ^ d[2] ^ r[28] ^ r[8] ^ r[15] ^ r[31] ^
r[17] ^ r[29] ^ r[11] ^ r[22] ^ r[3] ^ r[21] ^ r[2] ^ r[19] ^ r[18];
  xor32[24]=d[21] ^ d[16] ^ d[10] ^ d[20] ^ d[2] ^ d[27] ^ d[17] ^ d[30]
^ d[18] ^ d[14] ^ d[28] ^ d[7] ^ d[1] ^ r[1] ^ r[30] ^ r[28] ^ r[18] ^
r[16] ^ r[27] ^ r[14] ^ r[10] ^ r[21] ^ r[20] ^ r[2] ^ r[7] ^ r[17];
  xor32[23]= d[20] ^ d[15] ^ d[17] ^ d[29] ^ d[19] ^ d[26] ^ d[1] ^ d[16]
^ d[31] ^ d[27] ^ d[13] ^ d[9] ^ d[6] ^ d[0] ^ r[20] ^ r[13] ^ r[26] ^
r[15] ^ r[29] ^ r[0] ^ r[27] ^ r[6] ^ r[9] ^ r[31] ^ r[19] ^ r[1] ^ r[17]
^ r[16];
  xor32[22]=d[14] ^ d[11] ^ d[23] ^ d[12] ^ d[18] ^ d[31] ^ d[29] ^ d[26]
^ d[19] ^ d[27] ^ d[24] ^ d[9] ^ d[16] ^ d[0] ^ r[11] ^ r[19] ^ r[29] ^
```

```
r[27] ^ r[24] ^ r[12] ^ r[31] ^ r[14] ^ r[26] ^ r[23] ^ r[18] ^ r[0] ^
r[9] ^ r[16];
  xor32[21]=d[18] ^ d[13] ^ d[10] ^ d[22] ^ d[24] ^ d[9] ^ d[31] ^ d[27]
^ d[17] ^ d[26] ^ d[29] ^ d[5] ^ r[29] ^ r[5] ^ r[31] ^ r[27] ^ r[13] ^
r[26] ^ r[24] ^ r[18] ^ r[22] ^ r[10] ^ r[9] ^ r[17];
  xor32[20]=d[23] ^ d[17] ^ d[9] ^ d[21] ^ d[8] ^ d[12] ^ d[28] ^ d[26] ^
d[16] ^ d[25] ^ d[30] ^ d[4] ^ r[26] ^ r[30] ^ r[28] ^ r[4] ^ r[25] ^
r[12] ^ r[8] ^ r[23] ^ r[17] ^ r[9] ^ r[21] ^ r[16];
  xor32[19]=d[22] ^ d[27] ^ d[11] ^ d[16] ^ d[20] ^ d[29] ^ d[7] ^ d[25]
^ d[15] ^ d[24] ^ d[8] ^ d[3] ^ r[25] ^ r[22] ^ r[29] ^ r[24] ^ r[27] ^
r[11] ^ r[15] ^ r[3] ^ r[16] ^ r[8] ^ r[20] ^ r[7];
  xor32[18]=d[21] ^ d[23] ^ d[31] ^ d[26] ^ d[19] ^ d[10] ^ d[24] ^ d[6]
^ d[14] ^ d[7] ^ d[15] ^ d[28] ^ d[2] ^ r[24] ^ r[26] ^ r[2] ^ r[23] ^
r[10] ^ r[15] ^ r[28] ^ r[21] ^ r[31] ^ r[7] ^ r[14] ^ r[19] ^ r[6];
  xor32[17]=d[18] ^ d[23] ^ d[9] ^ d[27] ^ d[20] ^ d[22] ^ d[25] ^ d[5] ^
d[13] ^ d[6] ^ d[14] ^ d[30] ^ d[31] ^ d[1] ^ r[23] ^ r[13] ^ r[1] ^
r[14] ^ r[9] ^ r[18] ^ r[25] ^ r[20] ^ r[27] ^ r[6] ^ r[5] ^ r[22] ^
r[30] ^ r[31];
  xor32[16]=d[4] ^ d[22] ^ d[26] ^ d[29] ^ d[8] ^ d[17] ^ d[30] ^ d[19] ^
d[12] ^ d[21] ^ d[5] ^ d[13] ^ d[24] ^ d[0] ^ r[24] ^ r[30] ^ r[19] ^
r[26] ^ r[8] ^ r[22] ^ r[17] ^ r[4] ^ r[13] ^ r[5] ^ r[0] ^ r[21] ^ r[29]
^ r[12];
  xor32[15]=d[30] ^ d[8] ^ d[27] ^ d[24] ^ d[12] ^ d[21] ^ d[16] ^ d[18]
^ d[7] ^ d[5] ^ d[4] ^ d[20] ^ d[15] ^ d[9] ^ d[3] ^ r[5] ^ r[8] ^ r[18]
^ r[7] ^ r[21] ^ r[27] ^ r[15] ^ r[9] ^ r[24] ^ r[3] ^ r[4] ^ r[20] ^
r[30] ^ r[12] ^ r[16];
  xor32[14]=d[20] ^ d[15] ^ d[26] ^ d[17] ^ d[29] ^ d[23] ^ d[6] ^ d[7] ^
d[19] ^ d[4] ^ d[11] ^ d[3] ^ d[14] ^ d[8] ^ d[2] ^ r[20] ^ r[19] ^ r[17]
^ r[7] ^ r[15] ^ r[3] ^ r[6] ^ r[23] ^ r[4] ^ r[11] ^ r[26] ^ r[29] ^
r[8] ^ r[2] ^ r[14];
  xor32[13]=d[5] ^ d[28] ^ d[22] ^ d[18] ^ d[6] ^ d[16] ^ d[2] ^ d[10] ^
d[14] ^ d[19] ^ d[3] ^ d[25] ^ d[31] ^ d[13] ^ d[7] ^ d[1] ^ r[3] ^ r[22]
^ r[6] ^ r[1] ^ r[5] ^ r[13] ^ r[16] ^ r[2] ^ r[31] ^ r[25] ^ r[19] ^
r[14] ^ r[28] ^ r[18] ^ r[7] ^ r[10];
  xor32[12]=d[4] ^ d[17] ^ d[18] ^ d[9] ^ d[15] ^ d[13] ^ d[27] ^ d[31] ^
d[21] ^ d[5] ^ d[24] ^ d[2] ^ d[30] ^ d[1] ^ d[12] ^ d[6] ^ d[0] ^ r[17]
^ r[21] ^ r[0] ^ r[5] ^ r[15] ^ r[4] ^ r[31] ^ r[6] ^ r[2] ^ r[1] ^ r[30]
^ r[18] ^ r[27] ^ r[13] ^ r[24] ^ r[9] ^ r[12];
  xor32[11]=d[26] ^ d[14] ^ d[4] ^ d[15] ^ d[24] ^ d[27] ^ d[3] ^ d[12] ^
d[20] ^ d[28] ^ d[17] ^ d[1] ^ d[25] ^ d[31] ^ d[9] ^ d[16] ^ d[0] ^
r[12] ^ r[24] ^ r[26] ^ r[14] ^ r[4] ^ r[20] ^ r[28] ^ r[1] ^ r[3] ^
r[27] ^ r[15] ^ r[17] ^ r[0] ^ r[31] ^ r[25] ^ r[9] ^ r[16];
  xor32[10]=d[29] ^ d[28] ^ d[3] ^ d[19] ^ d[9] ^ d[26] ^ d[2] ^ d[13] ^
d[14] ^ d[16] ^ d[5] ^ d[31] ^ d[0] ^ r[28] ^ r[19] ^ r[3] ^ r[29] ^ r[5]
^ r[13] ^ r[26] ^ r[31] ^ r[0] ^ r[2] ^ r[9] ^ r[14] ^ r[16];
  xor32[9]=d[2] ^ d[23] ^ d[4] ^ d[5] ^ d[18] ^ d[9] ^ d[24] ^ d[29] ^
d[13] ^ d[11] ^ d[12] ^ d[1] ^ r[9] ^ r[24] ^ r[1] ^ r[18] ^ r[11] ^ r[2]
^ r[12] ^ r[23] ^ r[5] ^ r[4] ^ r[13] ^ r[29];
  xor32[8]=d[28] ^ d[23] ^ d[3] ^ d[1] ^ d[12] ^ d[8] ^ d[17] ^ d[22] ^
d[4] ^ d[11] ^ d[31] ^ d[10] ^ d[0] ^ r[8] ^ r[23] ^ r[31] ^ r[22] ^
r[28] ^ r[4] ^ r[17] ^ r[12] ^ r[10] ^ r[0] ^ r[11] ^ r[1] ^ r[3];
  xor32[7]=d[21] ^ d[22] ^ d[23] ^ d[29] ^ d[3] ^ d[16] ^ d[7] ^ d[24] ^
d[25] ^ d[5] ^ d[28] ^ d[15] ^ d[10] ^ d[8] ^ d[2] ^ d[0] ^ r[7] ^ r[21]
^ r[0] ^ r[23] ^ r[28] ^ r[3] ^ r[2] ^ r[24] ^ r[8] ^ r[22] ^ r[10] ^
r[5] ^ r[15] ^ r[25] ^ r[29] ^ r[16];
  xor32[6]=d[5] ^ d[25] ^ d[22] ^ d[8] ^ d[21] ^ d[2] ^ d[11] ^ d[6] ^
d[30] ^ d[4] ^ d[20] ^ d[29] ^ d[14] ^ d[7] ^ d[1] ^ r[6] ^ r[7] ^ r[14]
^ r[30] ^ r[2] ^ r[5] ^ r[22] ^ r[11] ^ r[21] ^ r[29] ^ r[1] ^ r[4] ^
r[20] ^ r[25] ^ r[8];
```

```
  xor32[5]=d[20] ^ d[5] ^ d[21] ^ d[1] ^ d[19] ^ d[13] ^ d[29] ^ d[10] ^
d[4] ^ d[28] ^ d[3] ^ d[24] ^ d[7] ^ d[6] ^ d[0] ^ r[5] ^ r[10] ^ r[7] ^
r[13] ^ r[21] ^ r[28] ^ r[20] ^ r[24] ^ r[0] ^ r[4] ^ r[3] ^ r[6] ^ r[29]
^ r[19] ^ r[1];
  xor32[4]=d[11] ^ d[20] ^ d[24] ^ d[30] ^ d[18] ^ d[3] ^ d[25] ^ d[29] ^
d[4] ^ d[19] ^ d[15] ^ d[31] ^ d[8] ^ d[2] ^ d[12] ^ d[6] ^ d[0] ^ r[4] ^
r[20] ^ r[11] ^ r[25] ^ r[19] ^ r[3] ^ r[24] ^ r[6] ^ r[30] ^ r[8] ^ r[2]
^ r[15] ^ r[29] ^ r[18] ^ r[0] ^ r[31] ^ r[12];
  xor32[3]=d[27] ^ d[2] ^ d[25] ^ d[3] ^ d[31] ^ d[10] ^ d[19] ^ d[15] ^
d[9] ^ d[18] ^ d[14] ^ d[8] ^ d[17] ^ d[7] ^ d[1] ^ r[3] ^ r[27] ^ r[25]
^ r[18] ^ r[8] ^ r[2] ^ r[15] ^ r[9] ^ r[14] ^ r[19] ^ r[1] ^ r[7] ^
r[10] ^ r[31] ^ r[17];
  xor32[2]=d[1] ^ d[24] ^ d[26] ^ d[17] ^ d[2] ^ d[16] ^ d[31] ^ d[18] ^
d[14] ^ d[8] ^ d[13] ^ d[7] ^ d[30] ^ d[9] ^ d[6] ^ d[0] ^ r[2] ^ r[24] ^
r[14] ^ r[6] ^ r[26] ^ r[1] ^ r[31] ^ r[9] ^ r[8] ^ r[18] ^ r[13] ^ r[7]
^ r[17] ^ r[0] ^ r[30] ^ r[16];
  xor32[1]=d[7] ^ d[1] ^ d[13] ^ d[24] ^ d[28] ^ d[17] ^ d[27] ^ d[11] ^
d[9] ^ d[16] ^ d[12] ^ d[6] ^ d[0] ^ r[1] ^ r[17] ^ r[12] ^ r[11] ^ r[27]
^ r[13] ^ r[24] ^ r[6] ^ r[7] ^ r[28] ^ r[0] ^ r[9] ^ r[16];
  xor32[0]=d[25] ^ d[31] ^ d[28] ^ d[24] ^ d[29] ^ d[30] ^ d[26] ^ d[10]
^ d[9] ^ d[16] ^ d[12] ^ d[6] ^ d[0] ^ r[0] ^ r[31] ^ r[24] ^ r[6] ^
r[26] ^ r[29] ^ r[28] ^ r[10] ^ r[25] ^ r[9] ^ r[12] ^ r[30] ^ r[16];
 end
endfunction

endmodule


// Verilog HDL for "vparts", "u2topia" "_functional"

module u2topia(rst,clock,
               rdNOTwr,sel_,add,ready_,dataIN,dataOUT,
               addREG,dinREG,doutREG,wrREG);

input       rst,clock;
input       ready_;
input [7:0] dataIN;
output      [7:0] dataOUT;
reg   [7:0] dataOUT;
output            rdNOTwr,sel_;
reg         rdNOTwr,sel_;
output      [7:0] add;
reg   [7:0] add;
input       wrREG;
input [1:0] addREG;
input [7:0] dinREG;
output      [7:0] doutREG;
reg   [7:0] doutREG;

reg   [7:0] reg1,reg2,reg3;

always @(rst)
 if (rst) begin
             assign sel_=0;
             assign rdNOTwr=0;
             assign add=0;
             disable machine1;
            end
 else begin
```

```
      deassign sel_;
      deassign rdNOTwr;
      deassign add;
     end

always @(addREG)
 case (addREG)
  0:assign doutREG=reg1;
  1:assign doutREG=reg2;
  2:assign doutREG=reg3;
  default:assign doutREG=8'bx;
 endcase

always @(posedge clock)
 begin : machine1
  if (rst) disable machine1;
  else
  if (wrREG)
  begin
   if (addREG==0) begin  //read bus
                   reg1<=dinREG;add<=dinREG;
                   rdNOTwr<=1;
                   @(posedge clock)
                   sel_<=0;
                   wait(ready_==0);
                   @(posedge clock)
                   reg3<=dataIN;
                   sel_<=1;
                  end
   else
   if (addREG==1) begin
                   reg2<=dinREG;add<=dinREG;
                   rdNOTwr<=0;
                   @(posedge clock)
                   sel_<=0;
                   dataOUT<=reg3;
                   @(posedge clock)
                   sel_<=1;
                  end
   else
   if (addREG==3) reg3<=dinREG;
  end
 end
endmodule

// Verilog HDL for "vparts", "tx" "_functional"

module tx(clock,rst,
          fullFIFO,wrFIFO,dataFIFO,
          ckin,din,full,
          ckout,dout,empty,
          enable,idle,
          TLB_BASE,TC_BASE);

input           clock,rst,enable;
output            idle;
reg         idle;

input       fullFIFO;
```

```
output  [31:0]  dataFIFO;
reg   [31:0]      dataFIFO;
reg   [23:0]      dataFIFO2;
output            wrFIFO;
reg         wrFIFO;

input           ckin;
input   [31:0]  din;
output          full;
wire    [31:0]  dataRET;
wire        emptyRET;
reg             rdRET;
retfifo
fifo1(.ckin(ckin),.din(din),.full(full),.ckout(rdRET),.dout(dataRET),.emp
ty(emptyRET),.rst(rst));

input           ckout;
output  [34:0]  dout;
output          empty;
reg             wrREQ;
reg     [34:0]  dataREQ;
wire        fullREQ;
reqfifo
fifo2(.ckin(wrREQ),.din(dataREQ),.full(fullREQ),.ckout(ckout),.dout(dout)
,.empty(empty),.rst(rst));

reg             wrRAM,rdRAM;
reg     [7:0]   addRAM;
wire    [37:0]  doutRAM;
reg     [37:0]  dinRAM;
ramtx r1(.wr(wrRAM),.rd(rdRAM),.add(addRAM),.din(dinRAM),.dout(doutRAM));

input     [31:12] TLB_BASE;
input     [31:18] TC_BASE;

reg     [31:0]  cacheTLB;
reg     [11:0]  cacheTLBoff;
wire queue=mux1(cacheTLBoff[1:0],cacheTLB);
reg   [5:0] OFFSET;

reg     [31:0]  HEADER,DMA,TRAILER,CRC;
reg             VALID,AAL5AAL0;
reg     [15:2]  RSVD1;
reg     [15:0]  COMP;

wire    [31:0]  newCRC;
xor32
x32(.a({dataFIFO[7:0],dataFIFO[15:8],dataFIFO[23:16],dataFIFO[31:24]}),.b
(CRC),.c(newCRC));

reg   [5:0] state;  // ajustar dimensoes
reg   [7:0] counter;

reg   [1:0] rot;
reg   [5:0] move,trf;
reg   [2:0] hold;
reg         endPDU;
```

```
always @(posedge clock)
 if (rst) begin
           wrFIFO<=0;
           wrREQ<=0;
           rdRET<=0;
           idle<=0;
           rdRAM<=0;
           wrRAM<=1;
           cacheTLB<=0;
           counter<=255;
           dinRAM<=38'b00000011111111111111111111111111111111;
           addRAM<=0;
           state<=0;
          end
 else
 case (state)
   0:if (counter==0) state<=1;
     else begin
           addRAM<=counter;
           counter<=counter-1;
           state<=0;
          end
   1:if (!enable) begin
                   idle<=1;
                   state<=1;
                  end
     else begin
           if (cacheTLBoff[1:0]==0) state<=2;
           else state<=4;
          end
// Get TLBs
   2:if (fullREQ) state<=2;
     else begin
           dataREQ<={5'b00000,TLB_BASE,cacheTLBoff[11:2],2'b00};
           wrREQ<=1;
           state<=3;
          end
   3:begin
      wrREQ<=0;
      cacheTLB<=dataRET;
      cacheTLBoff[11:2]<=cacheTLBoff[11:2]+1;
      state<=4;
     end
   4:if (queue==255) begin
                      cacheTLBoff[1:0]<=cacheTLBoff[1:0]+1;
                      state<=5;
                     end
     else begin
           addRAM<=queue;
           rdRAM<=1;
           state<=9;
          end
// SendIdle
   5:if (fullFIFO) state<=5;
     else begin
           dataFIFO<=32'b00000001000000000000000000000000;
           wrFIFO<=1;
           state<=6;
          end
```

BIBLIOTECA F.E. PORTO

```
 6:begin
    wrFIFO<=0;dataFIFO<=0;counter<=1;
    state<=7;
   end
 7:if (fullFIFO) state<=7;
   else begin
         wrFIFO<=1;counter<=counter+1;
         state<=8;
        end
 8:begin
    wrFIFO<=0;
    if (counter==12) state<=1;
    else state<=8;
   end
 9:begin
    rdRAM<=0;
    OFFSET<=doutRAM[37:32];
    state<=10;
   end
10:if (fullREQ) state<=10;
   else begin
         dataREQ={5'b00011,TC_BASE,queue,OFFSET,2'b00};
         wrREQ<=1;
         state<=11;
        end
11:begin
    wrREQ<=0;
    if (emptyRET) state<=11;
    else begin
          rdRET<=1;
          state<=12;
         end
   end
12:begin
    rdRET<=0;
    {COMP,RSVD1,AAL5AAL0,VALID}<=dataRET;
    state<=13;
   end
13:if (emptyRET) state<=13;
    else begin
          rdRET<=1;
          state<=14;
         end
14:begin
    rdRET<=0;
    DMA<=dataRET;
    state<=15;
   end
15:if (emptyRET) state<=15;
    else begin
          rdRET<=1;
          state<=16;
         end
16:begin
    rdRET<=0;
    HEADER<=dataRET;
    state<=17;
   end
17:if (emptyRET) state<=17;
```

```
      else begin
            rdRET<=1;
            state<=18;
           end
  18:begin
      rdRET<=0;
      TRAILER<=dataRET;
      if (VALID) state<=19;
      else begin
            cacheTLBoff[1:0]<=cacheTLBoff[1:0]+1;
            state<=5;
           end
     end
   19:if (fullFIFO) state<=19;
      else begin
            if (AAL5AAL0) endPDU=(COMP<=40);
            else endPDU=(COMP<=48);
            if (AAL5AAL0)
dataFIFO={HEADER[7:2],endPDU,HEADER[0],HEADER[15:8],HEADER[23:16],HEADER[
31:24]};
            else
dataFIFO={HEADER[7:2],endPDU,HEADER[0],HEADER[15:8],HEADER[23:16],HEADER[
31:24]};
            wrFIFO<=1;
            state<=20;
           end
   20:begin
       wrFIFO<=0;
       rot=DMA[1:0];
       move=(COMP<48 : COMP[5:0] : 48);
       trf=move+rot-4;
       trf=trf[5:2] + (trf[1] || trf[0]);
       state<=21;
      end
   21:if (move==0) state<=27;
      else begin
            counter<=0;hold<=0;
            state<=22;
           end
   22:if (fullREQ) state<=22;
      else begin
            wrREQ<=1;
            dataREQ[34]<=0;
            dataREQ[33:30]<=trf[3:0]-1;
            dataREQ[29:0]<=DMA[31:2];
            state<=23;
           end
   23:begin
       wrREQ<=0;
       state<=24;
      end
   24:begin
       if (wrFIFO==1) begin
                       wrFIFO<=0;dataFIFO={8'b0,dataFIFO2};
                       CRC=newCRC;
                      end
       if (counter==trf) state<=27;
       else if (emptyRET) state<=24;
            else begin
```

```
                rdRET<=1;counter<=counter+1;
                state<=23;
               end
   end
25:begin
    rdRET<=0;
    if (hold==0) {dataFIFO2,dataFIFO}<={24'b0,dataREQ};
    else
    if (hold==1) {dataFIFO2,dataFIFO}<={16'b0,dataREQ,dataFIFO[7:0]};
    else
    if (hold==2) {dataFIFO2,dataFIFO}<={8'b0,dataREQ,dataFIFO[15:0]};
    else
    {dataFIFO2,dataFIFO}<={dataREQ,dataFIFO[23:0]};
    if ((4-rot)<=move) hold=hold+4-rot;
    else hold=hold+move[1:0];
    rot=0;
    if (hold>=4) begin
                  if (fullFIFO) state<=26;
                  else begin
                        wrFIFO<=1;state<=24;hold=hold-4;move=move-4;
                       end
                 end
    else state<=24;
   end
26:if (fullFIFO) state<=26;
   else begin
         wrFIFO<=1;state<=24;hold<=hold-4;
        end
27:begin
    move=(COMP<48 : COMP[5:0] : 48);
    DMA=DMA+move;
    COMP=COMP-move;
    move=48-move;
    if (AAL5AAL0) state<=30;  //AAL5
    else state<=28; //AAL0
   end
28:if (move==0) state<=39;
   else if (fullFIFO) state<=28;
        else begin
              wrFIFO<=1;
              move<=move-4;
              state<=29;
             end
29:begin
    wrFIFO<=0;state<=28;
   end
30:if (move>=8) state<=31;
   else state<=37;
31:if (move==8) state<=33;
   else if (fullFIFO) state<=31;
        else begin
              wrFIFO<=1;
              move<=move-4;
              if (hold==0) dataFIFO<=0;
              else
              if (hold==1) dataFIFO[31:8]<=0;
              else
              if (hold==2) dataFIFO[31:16]<=0;
              else dataFIFO[31:24]<=0;
```

```
                hold=0;
                state<=32;
               end
   32:begin
       wrFIFO<=0;state<=31;
      end
   33:if (fullFIFO) state<=33;
      else begin
            wrFIFO<=1;

dataFIFO<={TRAILER[7:0],TRAILER[15:8],TRAILER[23:16],TRAILER[31:24]};
            state<=34;
           end
   34:begin
       wrFIFO<=0;
       state<=35;
      end
   35:if (fullFIFO) state<=35;
      else begin
            wrFIFO<=1;
            dataFIFO<={~CRC[7:0],~CRC[15:8],~CRC[23:16],~CRC[31:24]};
            state<=36;
           end
   36:begin
       wrFIFO<=0;state<=39;
      end
   37:if (move==0) state<=39;
      else if (fullFIFO) state<=37;
           else begin
                wrFIFO<=1;
                move<=move-4;
                if (hold==0) dataFIFO<=0;
                else
                if (hold==1) dataFIFO[31:8]<=0;
                else
                if (hold==2) dataFIFO[31:16]<=0;
                else dataFIFO[31:24]<=0;
                hold=0;
                state<=38;
               end
   38:begin
       wrFIFO<=0;state<=37;
      end
   39:begin
       if (endPDU) begin
                    VALID<=0;dinRAM[37:32]<=OFFSET+1;
                    dinRAM[31:0]<=32'b11111111111111111111111111111111;
                   end
       else dinRAM<={OFFSET,CRC};
       wrRAM<=1;
       state<=40;
      end
   40:begin
       wrRAM<=0;
       if (fullREQ) state<=40;
       else begin
             wrREQ<=1;
             dataREQ<={5'b10001,TC_BASE,queue,OFFSET,2'b00};
             state<=41;
```

```
            end
        end
    41:begin
        wrREQ<=0;
        state<=42;
       end
    42:if (fullREQ) state<=42;
        else begin
              wrREQ<=1;
              dataREQ<={3'b0,COMP,RSVD1,AAL5AAL0,VALID};
              state<=43;
             end
    43:begin
        wrREQ<=0;
        state<=44;
       end
    44:if (fullREQ) state<=44;
        else begin
              wrREQ<=1;
              dataREQ<={3'b0,DMA};
              state<=45;
             end
    45:begin
        wrREQ<=0;
        cacheTLBoff[1:0]<=cacheTLBoff[1:0]+1;
        state<=1;
       end
  default:state<=0;
 endcase

function [7:0] mux1;
input [1:0] s;
input [31:0]      a;
 case (s)
  0:mux1=a[31:24];
  1:mux1=a[23:16];
  2:mux1=a[15:8];
  3:mux1=a[7:0];
 endcase
endfunction

endmodule

// Verilog HDL for "vparts", "rx" "_functional"

module rx(clock,rst,
          enable, idle,
          ckin,din,full,
          ckout,dout,empty,
          dataFIFO,emptyFIFO,rdFIFO,
          reg_din,reg_add,reg_wr,
          config,incICC,incVEC,intFBCRX,intFBLRX,intPDU,intRAW,
          FBLRXg_BASE,FBLRXp_BASE,FBLRXg_TAIL,FBLRXp_TAIL,
          TCRX,TCRX_MASK,FBCRX_BASE,FBCRX_HEAD,

FBLRXg_HEAD,FBLRXp_HEAD,FBCRX_TAIL,FRCRX_BH,FRCRX_CLEFT,FRCRX_DMA);

input           clock,rst,enable;
output          idle;
```

```
reg             idle;

input           emptyFIFO;
input   [31:0]  dataFIFO;
output          rdFIFO;
reg             rdFIFO;

input           ckin;
input   [31:0]  din;
output          full;
wire    [31:0]  dataRET;
wire        emptyRET;
reg             rdRET;
retfifo
fifo1(.ckin(ckin),.din(din),.full(full),.ckout(rdRET),.dout(dataRET),.emp
ty(emptyRET),.rst(rst));

input           ckout;
output  [34:0]  dout;
output          empty;
reg             wrREQ;
reg     [34:0]  dataREQ;
wire        fullREQ;
reqfifo
fifo2(.ckin(wrREQ),.din(dataREQ),.full(fullREQ),.ckout(ckout),.dout(dout)
,.empty(empty),.rst(rst));

input   [8:0]   config;
output          incICC,incVEC,intFBCRX,intFBLRX,intPDU,intRAW;
reg             incICC,incVEC,intFBCRX,intFBLRX,intPDU,intRAW;

input [31:13]     FBLRXg_BASE,FBLRXp_BASE;
input [12:3]      FBLRXg_TAIL,FBLRXp_TAIL;
input [31:12]     TCRX;
input [31:0]      TCRX_MASK;
input [31:13]     FBCRX_BASE;
input [12:4]      FBCRX_HEAD;

output      [12:3]  FBLRXg_HEAD,FBLRXp_HEAD;
reg   [12:3]      FBLRXg_HEAD,FBLRXp_HEAD;
output  [12:4]  FBCRX_TAIL;
reg   [12:4]      FBCRX_TAIL;
output  [31:0]  FRCRX_BH;
reg   [31:0]      FRCRX_BH;
output      [31:2]      FRCRX_DMA;
reg   [31:2]      FRCRX_DMA;
output  [9:0]   FRCRX_CLEFT;
reg   [9:0] FRCRX_CLEFT;

reg     [31:0]  CRC,BH;
reg     [31:2]  DMA;
reg             PG,CLP,CE,GFC,OPEN,AAL5AAL0,PDUend,CRCerror;
reg     [12:0]  DWLeft;
reg     [13:0]  RSVD1;

reg     [3:0]   CELL_GFC;
reg     [7:0]   CELL_VPI;
reg     [15:0]  CELL_VCI;
reg     [2:0]   CELL_PT;
```

```
reg             CELL_CLP;
wire    [31:4]
TCRX_ADD=vadd(CELL_VPI,CELL_VCI,TCRX_MASK[15:0],TCRX[31:12]);
wire            on_table=(((CELL_VPI &
~TCRX_MASK[15:8])==TCRX_MASK[31:24]) && ((CELL_VCI[15:8] &
~TCRX_MASK[7:0])==TCRX_MASK[23:16]));
wire            gfc_ok=((CELL_GFC==0) || config[6]);

input           reg_wr;
input   [31:0]  reg_din;
input [2:0] reg_add;

reg             wrRAM,rdRAM;
reg     [4:0]   addRAM;
wire    [31:0]  doutRAM;
reg     [31:0]  dinRAM;
ramrx r1(.wr(wrRAM),.rd(rdRAM),.add(addRAM),.din(dinRAM),.dout(doutRAM));

wire    [31:0]  newCRC;
xor32
x32(.a({dataFIFO[7:0],dataFIFO[15:8],dataFIFO[23:16],dataFIFO[31:24]}),.b
(CRC),.c(newCRC));

reg     [2:0]   PbufferADD,GbufferADD;
reg     [2:0]   BCcacheADD;

reg   [6:0] state;
reg   [4:0] counter;
reg         return,new;
reg   [3:0] transfer;

always @(posedge clock)
 if (rst) begin
           PbufferADD=4;
           GbufferADD=4;
           BCcacheADD=0;
           wrRAM=0;
           rdRAM=0;
           idle=1;
           rdRET=0;
           wrREQ=0;
           rdFIFO=0;
           incICC=0;
           incVEC=0;
           intFBCRX=0;
           intFBLRX=0;
           intPDU=0;
           intRAW=0;
           FBLRXg_HEAD=8'b11111111;
           FBLRXp_HEAD=8'b11111111;
           //assign FBLRXg_TAIL=8'b11111111;
           //assign FBLRXp_TAIL=8'b11111111;
           //assign FBCRX_HEAD=9'b111111111;
           FBCRX_TAIL=9'b111111111;
           state=0;
         end
 else
 case (state)
  0:if (enable) begin
```

```
                idle<=0;state<=1;
               end
    else begin
          idle<=1;
          if (reg_wr) case (reg_add)
                       0:FBLRXg_HEAD<=reg_din[12:3];
                       1:FBLRXp_HEAD<=reg_din[12:3];
                       2:FBCRX_TAIL<=reg_din[12:4];
                       3:FRCRX_BH<=reg_din;
                       4:FRCRX_DMA<=reg_din[31:2];
                       5:FRCRX_CLEFT<=din[9:0];
                       default:;
                      endcase
         end
  1:if (PbufferADD!=4) state<=6;
//getPbuffers
    else begin
          if ((FBLRXp_TAIL-FBLRXp_HEAD)<=4) begin
                                             intFBLRX<=1;
                                             state<=2;
                                            end
          else state<=3;
         end
  2:begin
     intFBLRX<=0;
     if ((FBLRXp_TAIL-FBLRXp_HEAD)<=4) state<=2;
     else state<=3;
    end
  3:if (fullREQ) state<=3;
    else begin
          dataREQ[34:11]<={5'b00111,FBLRXp_BASE};
          dataREQ[10:1]<=FBLRXp_HEAD+1;
          dataREQ[0]<=0;
          FBLRXp_HEAD<=FBLRXp_HEAD+4;
          wrREQ<=1;counter<=0;PbufferADD<=0;
          state<=4;
         end
  4:begin
     wrREQ<=0;wrRAM<=0;
     if (emptyRET) state<=4;
     else begin
           rdRET<=1;
           state<=5;
          end
    end
  5:begin
     rdRET<=0;wrRAM<=1;addRAM<=counter;dinRAM<=dataRET;
     if (counter==7) state<=6;
     else begin
           counter<=counter+1;
           state<=4;
          end
    end
  6:begin
     wrRAM=0;
     if (GbufferADD!=4) state<=11;
//getGbuffers
     else begin
           if ((FBLRXg_TAIL-FBLRXg_HEAD)<=4) begin
```

```
                                                    intFBLRX<=1;
                                                    state<=7;
                                                   end
          else state<=8;
         end
   end
  7:begin
    intFBLRX<=0;
    if ((FBLRXg_TAIL-FBLRXg_HEAD)<=4) state<=7;
    else state<=8;
   end
  8:if (fullREQ) state<=8;
   else begin
         dataREQ[34:11]<={5'b00111,FBLRXg_BASE};
         dataREQ[10:1]<=FBLRXg_HEAD+1;
         dataREQ[0]<=0;
         FBLRXp_HEAD<=FBLRXg_HEAD+4;
         wrREQ<=1;counter<=8;GbufferADD<=0;
         state<=9;
        end
  9:begin
    wrREQ<=0;wrRAM<=0;
    if (emptyRET) state<=9;
    else begin
          rdRET<=1;
          state<=9;
         end
   end
 10:begin
    rdRET<=0;wrRAM<=1;addRAM<=counter;dinRAM<=dataRET;
    if (counter==15) begin
                      state<=11;
                      return<=0; //indica o POPstate para dumpcache
                     end
    else begin
          counter<=counter+1;
          state<=9;
         end
   end
 11:begin
    wrRAM<=0;
    if (BCcacheADD!=4) state<=16;
//dumpcache
    else begin
          if ((FBCRX_HEAD-FBCRX_TAIL)<=4) begin
                                           intFBCRX<=1;
                                           state<=12;
                                          end
          else state<=13;
         end
   end
 12:begin
    intFBCRX<=0;
    if ((FBCRX_HEAD-FBCRX_TAIL)<=4) state<=12;
    else state<=13;
   end
 13:if (fullREQ) state<=13;
   else begin
         BCcacheADD<=0;counter<=16;
```

```
          dataREQ[34:11]<={5'b01111,FBCRX_BASE};
          dataREQ[10:2]<=FBCRX_TAIL+1;
          dataREQ[1:0]<=0;
          FBCRX_TAIL<=FBCRX_TAIL+4;
          wrREQ<=1;
          state=14;
         end
 14:begin
     wrREQ<=0;rdRAM<=1;addRAM<=counter;
     state=15;
    end
 15:begin
     rdRAM<=0;
     if (fullREQ) state<=15;
     else begin
           dataREQ<=doutRAM;
           wrREQ<=1;
           if (counter==31)
            begin
             if (return) state<=65; // return1 vem de ProcNormal
             else state<=16; // return0 foi chamado do inicio
            end
           else begin
                 counter<=counter+1;
                 state<=14;
                end
          end
    end
 16:begin
     wrREQ<=0;
     if (emptyFIFO) state<=0;
     else begin
           rdFIFO<=1;
           state<=17;
          end
    end
// ProcCell
 17:begin
     rdFIFO<=0;
     CELL_GFC<=dataFIFO[7:4];
     CELL_VPI<={dataFIFO[3:0],dataFIFO[15:12]};
     CELL_VCI<={dataFIFO[11:8],dataFIFO[23:16],dataFIFO[31:28]};
     CELL_PT<=dataFIFO[27:25];
     CELL_CLP<=dataFIFO[24];
     state<=18;
    end
 18:if ({CELL_GFC,CELL_VPI,CELL_VCI,CELL_PT,CELL_CLP}==1)
     begin
      counter<=12;
      if (!on_table && !config[4]) begin
                                    incVEC<=1;state<=19;
                                   end
      state<=20;
     end
    else
    if ((!gfc_ok && config[5])||(!on_table &&
config[4])||(CELL_PT[2:1]==2'b10)||((CELL_PT[2:1]==2'b11) && config[7]))
     begin
      if (!gfc_ok && config[5]) begin
```

```
                                    incICC<=1;state<=21;
                                   end
      else state<=22;
     end
    else state<=34;
 19:begin
     incVEC<=0;state<=20;
    end
// Flush12
 20:if (rdFIFO==1) begin
                    rdFIFO<=0;
                    if (counter!=0) state<=19;
                    else state<=0;
                   end
    else begin
          if (emptyFIFO) rdFIFO<=0;
          else begin
                rdFIFO<=1;
                counter<=counter-1;
               end
          state<=19;
         end
  21:begin
      incICC<=0;state<=22;
     end
//ProcRaw
  22:if (fullREQ) state<=22;
     else begin
           dataREQ<={5'b11100,FRCRX_DMA};
           wrREQ<=1;
           state<=23;
          end
  23:begin
      wrREQ<=0;dataREQ<=dataFIFO;
      state<=24;
     end
  24:begin
      wrREQ<=1;counter<=1;
      state<=25;
     end
  25:begin
      wrREQ<=0;
      if (emptyFIFO) state<=25;
      else begin
            rdFIFO<=1;state<=26;
           end
     end
  26:begin
      rdFIFO<=0;
      if (fullREQ) state<=26;
      else begin
            wrREQ<=1;dataREQ<=dataFIFO;
            if (counter==12) state<=27;
            else begin
                  counter<=counter+1;
                  state<=25;
                 end
           end
     end
```

```
27:begin
    wrREQ<=0;FRCRX_DMA=FRCRX_DMA+16;
    FRCRX_CLEFT=FRCRX_CLEFT-1;
    if (FRCRX_CLEFT!=0) state<=0;
    else begin
          dataREQ={5'b10001,FRCRX_DMA};
          state<=28;intRAW<=1;
         end
   end
28:begin
    intRAW<=0;
    if (fullREQ) state<=28;
    else begin
          wrREQ<=1;state<=29;
         end
   end
29:begin
    wrREQ<=0;
    addRAM[4:1]<=GbufferADD+4;
    addRAM[0]<=0;
    rdRAM<=1;
    state<=30;
   end
30:begin
    FRCRX_BH<=doutRAM;
    addRAM<=addRAM+1;
    state<=31;
   end
31:begin
    FRCRX_DMA<=doutRAM[31:2];
    rdRAM<=0;
    GbufferADD<=GbufferADD+1;
    case (config[3:2])
     0:FRCRX_CLEFT<=63;
     1:FRCRX_CLEFT<=127;
     2:FRCRX_CLEFT<=511;
     3:FRCRX_CLEFT<=1023;
    endcase
    dataREQ<={3'b0,FRCRX_BH};
    state<=32;
   end
32:if (fullREQ) state<=32;
   else begin
         wrREQ<=1;state<=33;
        end
33:begin
    wrREQ<=0;state<=0;
   end
34:if (fullREQ) state<=34;
   else begin
         wrREQ<=1;dataREQ<={5'b00011,TCRX_ADD,2'b00};
         state<=35;
        end
35:begin
    wrREQ<=0;
    if (emptyRET) state<=35;
    else begin
          rdRET<=1;state<=36;
         end
```

```
    end
  36:begin
     rdRET<=0;
     DWLeft[12]<=0;
     {PG,DWLeft[11:0],CLP,CE,GFC,OPEN,AAL5AAL0,RSVD1}<=dataRET;
     state<=37;
    end
  37:if (emptyRET) state<=37;
    else begin
          rdRET<=1;state<=38;
         end
  38:begin
     rdRET<=0;CRC<=dataRET;state<=39;
    end
  39:if (emptyRET) state<=39;
    else begin
          rdRET<=1;state<=40;
         end
  40:begin
     rdRET<=0;DMA<=dataRET[31:2];state<=41;
    end
  41:if (emptyRET) state<=41;
    else begin
          rdRET<=1;state<=42;
         end
  42:begin
     rdRET<=0;BH<=dataRET;
     if (!OPEN && config[4]) state<=22;
     else
     if (!OPEN && !config[4]) begin
                               incVEC<=1;state<=43;
                              end
     else state<=44;
    end
  43:begin
     incVEC<=0;state<=20;
    end
// ProcNormal
  44:if (DWLeft!=0) begin
                     new<=0;
                     CLP<=CLP || CELL_CLP;
                     CE<=CE || CELL_PT[1];
                     GFC<=GFC || (CELL_GFC!=0);
                     state<=47;
                    end
    else begin
          new<=1;
          CRC<=32'b11111111111111111111111111111111;
          CLP<=CELL_CLP;
          CE<=CELL_PT[1];
          GFC<=CELL_GFC!=0;
          RSVD1<=0;
          PG<=0;
          case (config[1:0])
           0:DWLeft<=16;
           1:DWLeft<=32;
           2:DWLeft<=64;
           3:DWLeft<=128;
          endcase
```

```
        addRAM<={2'b0,PbufferADD[1:0],1'b0};
        rdRAM<=1;
        state<=45;
       end
45:begin
    BH<=doutRAM;
    addRAM<=addRAM+1;
    state<=46;
   end
46:begin
    DMA<=doutRAM[31:2];
    rdRAM<=0;
    PbufferADD<=PbufferADD+1;
    state<=47;
   end
47:begin
    transfer=((DWLeft>=12) ? 4'b1100 : DWLeft[3:0]);
    state<=48;
   end
48:if (fullREQ) state<=48;
   else begin
         wrREQ<=1;
         dataREQ[34]<=1;
         dataREQ[33:30]<=transfer-1;
         dataREQ[29:0]<=DMA;
         state<=49;
        end
49:begin
    wrREQ<=0;DMA<=DMA+transfer;DWLeft<=DWLeft-transfer;
    state<=50;counter<=1;
   end
50:begin
    if (wrREQ==1) begin
                   wrREQ<=0;CRC=newCRC;
                  end
    if (emptyFIFO) state<=50;
    else begin
          rdFIFO<=1;
          state<=51;
         end
   end
 51:begin
     rdFIFO<=0;
     if (fullREQ) state<=51;
     else begin
           wrREQ<=1;dataREQ<=dataFIFO;
           if (counter==transfer) state<=52;
           else begin
                 counter<=counter+1;
                 state<=50;
                end
          end
    end
 52:begin
     wrREQ<=0;CRC=newCRC;
     if (transfer==12) state<=64;
     else begin
           PDUend<=0;CRCerror<=0;
           addRAM<={BCcacheADD,2'b0}+16;
```

```
dinRAM<={PG,CLP,CE,GFC,AAL5AAL0,DWLeft[11:0],PDUend,CRCerror,13'b00000000
00001};
                wrRAM<=1;
                state<=53;
               end
        end
     53:begin
         dinRAM<={8'b0,CELL_VPI,CELL_VCI};addRAM<=addRAM+1;
         state<=54;
        end
     54:begin
         dinRAM<=BH;addRAM<=addRAM+1;
         state<=55;
        end
     55:begin
         dinRAM<=CRC;addRAM<=addRAM+1;
         state<=56;
        end
     56:begin
         wrRAM<=0;BCcacheADD<=BCcacheADD+1;
         addRAM[4:1]<=GbufferADD+4;addRAM[0]<=0;
         rdRAM<=1;
         state<=57;
        end
     57:begin
         BH<=doutRAM;addRAM<=addRAM+1;
         state<=58;
        end
     58:begin
         rdRAM<=0;DMA<=doutRAM[31:2];
         GbufferADD<=GbufferADD+1;
         case (config[3:2])
          0:DWLeft<=256;
          1:DWLeft<=512;
          2:DWLeft<=2048;
          3:DWLeft<=4096;
         endcase
         transfer<=12-transfer;
         state<=59;
        end
     59:if (fullREQ) state<=59;
        else begin
              wrREQ<=1;
              dataREQ[29:0]<=DMA;
              dataREQ[33:30]<=transfer-1;
              dataREQ[34]<=1;
              state<=60;
             end
     60:begin
         wrREQ<=0;counter<=1;state<=61;
        end
     61:begin
         if (wrREQ==1) begin
                        wrREQ<=0;CRC=newCRC;
                       end
         if (emptyFIFO) state<=61;
         else begin
               rdFIFO<=1;
```

```
            state<=62;
          end
      end
    62:begin
        rdFIFO<=0;
        if (fullREQ) state<=62;
        else begin
              wrREQ<=1;dataREQ<=dataFIFO;
              if (counter==transfer) state<=63;
              else begin
                    counter<=counter+1;
                    state<=61;
                   end
          end
      end
    63:begin
        wrREQ<=0;CRC<=newCRC;
      end
    64:if (CELL_PT[0] && AAL5AAL0)
        begin
         PDUend<=1;
         CRCerror<=(CRC!=32'b11000111000001001101110101111011);
         if (BCcacheADD==4) begin
                             return<=1;
                             state<=11;
                            end
         else state<=65;
        end
       else state<=73;
    65:begin
        wrREQ<=0;
        addRAM<={BCcacheADD,2'b0}+16;

dinRAM<={PG,CLP,CE,GFC,AAL5AAL0,DWLeft[11:0],PDUend,CRCerror,13'b00000000
00001};
        wrRAM<=1;
        state<=66;
       end
    66:begin
        dinRAM<={8'b0,CELL_VPI,CELL_VCI};addRAM<=addRAM+1;
        state<=67;
       end
    67:begin
        dinRAM<=BH;addRAM<=addRAM+1;
        state<=68;
       end
    68:begin
        dinRAM<=CRC;addRAM<=addRAM+1;
        state<=69;
       end
    69:begin
        wrRAM<=0;BCcacheADD<=BCcacheADD+1;
        state<=70;intPDU<=1;
       end
    70:begin
        intPDU<=0;
        if (fullREQ) state<=70;
        else begin
              wrREQ<=1;
```

```
                dataREQ<={5'b10000,TCRX_ADD,2'b0};
                state<=71;
               end
      end
    71:if (fullREQ) state<=71;
      else begin
            wrREQ<=1;
            dataREQ<={19'b0,OPEN,AAL5AAL0,RSVD1};
            state<=72;
           end
    72:begin
       wrREQ<=0;state<=0;
      end
    73:if (fullREQ) state<=73;
      else begin
            wrREQ<=1;
            dataREQ<={5'b10011,TCRX_ADD,2'b0};
            state<=74;
           end
    74:begin
       wrREQ<=0;state<=75;
      end
    75:if (fullREQ) state<=75;
      else begin
            wrREQ<=1;

dataREQ<={3'b0,PG,DWLeft[11:0],CLP,CE,GFC,OPEN,AAL5AAL0,RSVD1};
            state<=76;
           end
    76:begin
       wrREQ<=0;state<=77;
      end
    77:if (fullREQ) state<=77;
      else begin
            wrREQ<=1;
            dataREQ<={3'b0,CRC};
            state<=78;
           end
    78:begin
       wrREQ<=0;state<=79;
      end
    79:if (fullREQ) state<=79;
      else begin
            wrREQ<=1;
            dataREQ<={3'b0,DMA,2'b0};
            state<=80;
           end
    80:begin
       wrREQ<=0;state<=81;
      end
    81:if (fullREQ) state<=81;
      else begin
            wrREQ<=1;
            dataREQ<={3'b0,BH};
            state<=82;
           end
    82:begin
       wrREQ<=0;state<=0;
      end
```

```
  default:state<=0;
 endcase

function [31:4] vadd;
 input  [7:0]   vp;
 input      [15:0]      vc;
 input      [15:0]      mask;
 input      [31:12] base;
 begin
  casex (mask[7:0])
   8'b00000000:vadd={base[31:20],((base[19:12] & ~mask[15:8])|(vp &
mask[15:8])),vc[7:0]};
   8'b00000001:vadd={base[31:21],((base[20:13] & ~mask[15:8])|(vp &
mask[15:8])),vc[8:0]};
   8'b0000001x:vadd={base[31:22],((base[21:14] & ~mask[15:8])|(vp &
mask[15:8])),vc[9:0]};
   8'b000001xx:vadd={base[31:23],((base[22:15] & ~mask[15:8])|(vp &
mask[15:8])),vc[10:0]};
   8'b00001xxx:vadd={base[31:24],((base[23:16] & ~mask[15:8])|(vp &
mask[15:8])),vc[11:0]};
   8'b0001xxxx:vadd={base[31:25],((base[24:17] & ~mask[15:8])|(vp &
mask[15:8])),vc[12:0]};
   8'b001xxxxx:vadd={base[31:26],((base[25:18] & ~mask[15:8])|(vp &
mask[15:8])),vc[13:0]};
   8'b01xxxxxx:vadd={base[31:27],((base[26:19] & ~mask[15:8])|(vp &
mask[15:8])),vc[14:0]};
   8'b1xxxxxxx:vadd={base[31:28],((base[27:20] & ~mask[15:8])|(vp &
mask[15:8])),vc[15:0]};
  endcase
 end
endfunction

endmodule
```

Glossário

AAL	ATM Adaptation Layer
ABR	Abailable bit rate
ATM	Asynchronous Transfer Mode
AUU	ATM layer user to ATM layer user
CAD	Computer Aided Design
CBR	Constant bit rate.
CI	Congestion Indicator
CI	Circuito Integrado
CIAE	Circuito Integrado de Aplicação Específica
CLP	Cell Loss Priority
CMOS	Complementary Metal-Oxide Semiconductor
CPU	Central Processing Unit
CRC	Cyclic Redundant Code
FDDI	Fibre-Distributed Data Interface
FIFO	First-in First-out
LAN	Local Area Network
MRP	Modelo de Referência Protocolar
NNI	Network-Node Interface
OAM	Operation and Maintenance
OSI	Open Systems Interconnection
PC	Personal Computer
PCI	Peripheral Component Interconnect
PCI	Protocol Control Information
PCI	Peripheral Component Interconnect
PDU	Protocol Data Unit
PT	Payload Type
PTI	Payload Type Indicator
RDIS	Rede Digital com Integração de Serviços
SAP	Service Access Point
SAR	Segmentation and Reassembly
SDH	Synchronous digital hierarchy
SDU	Service Data Unit
TC	Transmission Convergence sub-layer
UNI	User-Network Interface
UTOPIA	Universal Test and Operations PHY Interface for ATM
UTP	Unshielded Twisted Pair
UU	CPCS user-to-user indication
VBR	Variable bit rate
VC	Virtual Channel
VCC	Virtual Channel Connection
VCI	Virtual Channel Identifier
VP	Virtual Path
VPC	Virtual Path Connection
VPI	Virtual Path Identifier
WAN	Wide Area Network

FACULDADE DE ENGENHARIA

Rua dos Bragas 4099 PORTO CODEX PORTUGAL
Telef.: 351 2 323201 • Fax: 351 2 319280 • Telex: 27323 FEUP P

FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO
BIBL