Projeto de Protocolos visando a Implementação em Hardware: aplicação ao Protocolo IPv6 no Âmbito da Abordagem "IP over ATM"

UFRJ

Tese submetida para a obtenção do título de

Mestre em Ciências em Engenharia Elétrica

aos Programas de Pós-Graduação de Engenharia Elétrica

da COPPE/UFRJ

por

Emerson Carli

Projeto de Protocolos visando a Implementação em Hardware: aplicação ao Protocolo IPv6 no Âmbito da Abordagem "IP over ATM"

Emerson Carli

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA ELÉTRICA.

Aprovada por:

Prof. Aloysio de Castro Pinto Pedroza, Dr.
(Presidente)

Prof. Antônio Carneiro de Mesquita Filho, Dr. d'État

Prof ª. Luci Pirmez, D.Sc.

Prof. Jorge Lopes de Souza Leão, Dr.Ing.

Prof. Julio Salek Aude, Ph.D.

RIO DE JANEIRO, RJ - BRASIL
ABRIL DE 1998

CARLI, EMERSON

Projeto de Protocolos visando a Implementação em Hardware: aplicação ao Protocolo IPv6 no Âmbito da Abordagem "IP over ATM" [Rio de Janeiro] 1998

XIV, 136 p. 29,7 cm (COPPE/UFRJ, M.Sc., Engenharia Elétrica, 1998)

Tese - Universidade Federal do Rio de Janeiro, COPPE

1. Especificação Formal de Protocolos de Comunicação

2. Implementação de Protocolos em Hardware

I. COPPE/UFRJ II. Título (Série)

Agradecimentos

Em primeiro lugar, gostaria de agradecer minha mãe, que me incentivou a fazer este curso de mestrado, com o qual adqüiri grandes conhecimentos e que sempre me apoiou e torceu pelo meu sucesso.

Aos professores Aloysio de Castro Pinto Pedroza e Otto Carlos Muniz Bandeira Duarte, pelos conhecimentos passados, pela amizade conseguida ao longo do curso e pela confiança prestada a mim.

À todos meus amigos de mestrado, pelo ótimo e descontraído ambiente de trabalho, pela amizade e pelo espírito de equipe: Marcelo Baiano, Ricardo Belem, LuisH, Antônio (Antguria), Andrea, Adriana, Rubi, Ronaldo, Fred, Smara, Daniel, Evandro, Antônio (Tony), Flávio, Renata, Jonny, Marcos, João Marcelo (JM), Salomão, Coutinho, Castanon, Rosas, Spindola, Marcial, Marivan.

Aos meus três orientadores, pelos conselhos e pela atenção dada na revisão deste trabalho.

Aos professores Jorge Lopes de Souza Leão e Julio Salek Aude, por participarem de minha banca.

Um agradecimento especial à minha orientadora Luci Pirmez e ao meu amigo Belem, pela grande ajuda na realização deste trabalho.

À UFRJ, FUJB, CNPq e CAPES, pelo financiamento deste trabalho.

A Deus que me guia e ilumina.

Resumo da Tese de Mestrado apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

Projeto de Protocolos visando a Implementação em Hardware: aplicação ao Protocolo IPv6 no Âmbito da Abordagem "IP over ATM"

Emerson Carli

Abril de 1998

Orientadores: Aloysio de Castro Pinto Pedroza
Antônio Carneiro de Mesquita Filho
Luci Pirmez

Departamento: Engenharia Elétrica

Com o surgimento de redes de alta velocidade e de aplicações multimídias, novos requisitos de desempenho são impostos nas estações que compõem estas redes, refletindo-se em modificações no projeto de protocolos, que devem ser mais simples e mais eficientes. Entretanto, o alto desempenho de subsistemas de comunicação é obtido não apenas com a otimização em *software* dos protocolos mas também com sua implementação em *hardware*.

Este trabalho tem como objetivo aplicar uma metodologia que traduz especificações Estelle em descrições VHDL com o intuito de implementar em *hardware* protocolos destinados a sistemas de comunicação de alta velocidade. Este trabalho aplica esta metodologia na nova versão do protocolo IP, oficialmente chamado de IPv6. Levando-se em consideração as futuras redes, este trabalho adota a abordagem "IP over ATM" para a implementação em *hardware* do protocolo IPv6.

Este projeto recebeu recursos dos seguintes orgãos financiadores de pesquisa: UFRJ, FUJB, CNPq, CAPES e COFECUB.

Abstract of the thesis presented to COPPE/UFRJ as partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

Design for Hardware Implementation of Protocols: Study of the IPv6 Protocol in the IP over ATM Approach

Emerson Carli

April 1998

Supervisors: Aloysio de Castro Pinto Pedroza
Antônio Carneiro de Mesquita Filho
Luci Pirmez
Department: Electrical Engineering

The emergence of high-speed networks and new applications has imposed new performance requirements to workstations within such networks. Therefore, protocols must be simpler and more efficient to support these new requirements. However, high performance in communication subsystems is obtained not only with software optimization of high-speed protocols but with their implementation in hardware.

This work follows a methodology that efficiently translates Estelle formal specifications into VHDL descriptions in order to implement high-speed protocols in hardware. This work applies this methodology at the new IP protocol version, officialy called IPv6.

Taking into account the future networks, this work adopts the IP over ATM approach to implement the IPv6 protocol in hardware.

Palavras-chave

Estelle

VHDL

IPv6

Abordagem “IP over ATM”

Implementação de Protocolos em Hardware

Glossário

ATM :	Modo de Transferência Assíncrono;
GTA :	Grupo de Teleinformática e Automação;
ISO :	*International Organization for Standardization*;
OSI :	*Open Systems Interconnection*;
IP :	*Internet Protocol*;
IPv6 :	*Internet Protocol version 6*;
MAC :	*Medium Access Control*;
AAL5 :	*ATM Adaptation Layer 5*;
CCITT :	*Comité Consultatif International Telégraphique et Téléphonique*;
UNI :	*User-Network Interface*;

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo I

Introdução

OS avanços nas tecnologias de semicondutores e o surgimento das fibras ópticas como meio físico de transmissão vêm permitindo o desenvolvimento de redes de comunicação cada vez mais velozes e confiáveis. Como conseqüência, o ponto crucial na obtenção de um bom desempenho passou a ser a quantidade de processamento necessária para executar as funções dos protocolos de comunicação e não mais a limitação de velocidade do meio físico. Além disso, a viabilização de novas aplicações, as chamadas aplicações multimídias, que exigem bandas passantes largas, impõem melhorias de desempenho nas estações.

O surgimento de aplicações multimídias introduziu novas exigências que demandam modificações no projeto de protocolos, principalmente no que se refere à obtenção de subsistemas de comunicação que permitam uma alta vazão. Os protocolos devem ser mais simples para permitir operações de alta velocidade e mais eficientes, para suportar os tráfegos multimídias.

Neste sentido, vários trabalhos vêm sendo realizados ao longo dos anos com o objetivo de se obter subsistemas de alto desempenho. No trabalho de Zitterbart [1], procurou-se explorar o paralelismo eventualmente existente nos protocolos, para reduzir o *overhead* no processamento de suas funções e de tornar o processamento dos mesmos mais eficiente. No trabalho de Duarte *et al.* [2] foi proposta uma arquitetura visando uma implementação de alto desempenho e analisou-se alguns fatores que afetam o desempenho de um sistema de comunicação estruturado em camadas baseado no modelo OSI. A alocação e o gerenciamento de *buffers* existentes na implementação de modelos organizados em camadas foi o assunto abordado em [3], que propôs estratégias de se evitar cópia de dados em memória. Estes estudos mostram que nem sempre é possível obter subsistemas de comunicação de alto desempenho com a otimização do *software* de controle dos protocolos nas redes de alta velocidade. Nestes casos, a implementação de controladores de protocolo em *hardware* surge como a alternativa mais evidente. Krishnakumar e Sabnani [4] mostraram que as funções de uma ou mais camadas de protocolos podem ser implementadas em circuitos VLSI. Um controlador de protocolo é dito programável se puder implementar diferentes protocolos ou variações de um mesmo protocolo. A implementação de protocolos de comunicação em *hardware*, porém, não é uma tarefa trivial, demandando, como já foi visto, mudanças nos métodos de projeto.

Circuitos VLSI são atualmente projetados de maneira semi-automática usando-se uma série de ferramentas integradas a um determinado ambiente de projeto. O procedimento de projeto normalmente tem como ponto de partida uma especificação do sistema em alto nível, denominada de descrição comportamental. No caso dos protocolos de comunicação, entretanto, os procedimentos de projeto tradicionais envolvem especificações do sistema em um nível de abstração maior que o normalmente empregado na descrição comportamental de circuitos. Este é o caso das linguagens de especificação derivadas das Técnicas de Descrição Formal (FDTs - *Formal Description Techniques*) [5, 6, 7] tais como, por exemplo, a linguagem Estelle [8]. Como conseqüência, o projetista de protocolos quando confrontando com o problema de implementar uma dada especificação em *hardware*, deverá lidar com dois ambientes de projeto distintos: um envolvendo a especificação formal, verificação e teste do protocolo e o outro envolvendo a simulação, síntese e implementação física do circuito VLSI. Em geral, os dois ambientes são baseados em diferentes linguagens de descrição, o que requer uma interface de tradução de uma linguagem para outra.

As Técnicas de Descrição Formal permitem a especificação precisa de protocolos de comunicação evitando qualquer tipo de ambigüidade em sua interpretação. As ferramentas de projeto baseadas nas Técnicas de Descrição Formal são as ferramentas mais familiares ao projetista de protocolos. Entre as linguagens derivadas das FDTs uma das mais utilizadas é a linguagem Estelle [8]. A linguagem Estelle é uma das linguagens formais de especificação da ISO sendo utilizada para a descrição de sistemas distribuídos. A linguagem Estelle tem como características estruturais o uso de módulos, pontos de interação, primitivas e um sistema de transições para a descrição da arquitetura e do comportamento de um dado sistema, o que facilita a modelagem da hierarquia e da comunicação entre processos.

Uma linguagem de descrição de *hardware* (HDL) tal como o VHDL (*Very high speed integrated circuits Hardware Description Language*) [9, 10] é necessária para a utilização de ambientes específicos de projeto de *hardware*, tais como as ferramentas de síntese de alto nível (HLS - *High Level Synthesis*).

I.1 Objetivo do Trabalho

O objetivo deste trabalho é aplicar a metodologia elaborada por Pirmez [11] na implementação de protocolos de redes de alta velocidade em *hardware*. Esta metodologia envolve a tradução de especificações Estelle em descrições VHDL. A vantagem desta metodologia é poder implementar protocolos em *hardware* a partir de uma descrição formal mais próxima do projetista de protocolos. Durante a escolha do protocolo a ser implementado, optou-se pela nova versão proposta do protocolo IP, oficialmente chamado de IPv6 [12, 13, 14]. O protocolo IPv6 é um protocolo novo, em grande discussão e pertence à nova geração de protocolos, elaborado para suportar os atuais requisitos de desempenho das novas aplicações. Levando em consideração as futuras redes, adotou-se neste trabalho uma abordagem de implementação para este protocolo. Esta abordagem consiste na implementação do protocolo IPv6 diretamente acima da tecnologia ATM (*Asynchronous Transfer Mode*) [15], baseada no caminho que vem sendo adotado pelo grupo IETF (*Internet Engineering Task Force*) responsável pela RFC 1577 [16], sendo comumente chamada de "IP over ATM".

Dentre as linguagens formais existentes (Estelle, LOTOS, SDL e Promela), Estelle foi escolhida pela existência de um ambiente de *software* adequado e disponível para o desenvolvimento de protocolos nesta linguagem. Já a linguagem de descrição de *hardware* adotada neste trabalho foi a linguagem VHDL por ser de maior aceitação tanto pela indústria quanto pelos grupos de pesquisa, sendo por isto a linguagem de entrada utilizada pela maioria das ferramentas de síntese atuais.

Após o mapeamento de uma especificação Estelle de um protocolo numa descrição VHDL, um protocolo pode ser especificado por diferentes estilos de descrição comportamental em VHDL. Tais estilos fornecem descrições de igual semântica que diferem somente na qualidade da estrutura gerada na síntese de alto nível (HLS). A qualidade da descrição estrutural de saída fornecida pela HLS é medida pelo número de estados e caminhos gerados pelo processo específico de *scheduling* executado durante uma HLS. Dentre os vários estilos de descrição propostos e analisados em [11, 17], este trabalho utilizará o estilo que apresentou os melhores resultados, no que diz respeito à obtenção de estruturas de *hardware* de menor área e, conseqüentemente menor custo, durante o processo de síntese de alto nível.

Durante o processo realizado na direção da implementação em *hardware* do protocolo escolhido, o protocolo IPv6, este trabalho contribui no sentido de avaliar a complexidade do *hardware* resultante, em termos da área do circuito sintetizado [18].

I.2 Organização do Trabalho

Este trabalho está organizado da seguinte forma. O capítulo II apresenta a metodologia utilizada neste trabalho para a implementação de protocolos em *hardware*. Para seu correto entendimento, os principais conceitos e características das linguagens Estelle e VHDL serão destacados. No capítulo III é apresentado o protocolo utilizado neste trabalho, o protocolo IPv6, mostrando suas principais características. Além disso, a abordagem de implementação adotada para este protocolo é descrita. O capítulo IV destacará os principais resultados obtidos ao se aplicar a metodologia no protocolo IPv6 para sua implementação em *hardware*. As considerações finais sobre a implementação em *hardware* de protocolos são comentadas no capítulo V.

Capítulo II

Uma Metodologia para a Implementação de Protocolos em Hardware

II.1 Introdução

NA década passada, o interesse pelos protocolos padronizados pela ISO motivaram várias pesquisas na área de engenharia de protocolos. Desde o início, notou-se a importância dos métodos formais na especificação de protocolos de comunicação. O termo Técnicas de Descrição Formal (FDTs) surgiu de modo a referenciar as técnicas usadas para descrever precisamente os protocolos sem ambigüidades na sua interpretação. A ISO e o CCITT contribuíram com o desenvolvimento de FDTs padronizadas no fim da década de 80. As linguagens Estelle e LOTOS foram definidas pela ISO, enquanto que a linguagem SDL foi definida pelo CCITT.

Paralelamente, pesquisas recentes vêm sendo desenvolvidas na área de microeletrônica com o objetivo de se obter subsistemas de alto desempenho. Conforme as redes se tornam mais velozes e confiáveis, o processamento da informação passou a ser o fator limitante. Como tendência, aumenta-se o interesse na implementação de funções críticas dos protocolos ou de todo o protocolo em *hardware*.

Para alcançar este objetivo, técnicas vem sendo desenvolvidas para permitir o projeto semi-automático de circuitos VLSI a partir das especificações dos protocolos em níveis os mais abstratos possíveis. Como exemplo, podem ser citados trabalhos de pesquisa que tentam fazer uma ligação entre uma especificação no nível de sistema e as ferramentas de *hardware* existentes, tais como a tradução de LOTOS para VHDL [19] e a tradução de Estelle para VHDL [20]. Contudo, estes trabalhos preocuparam-se apenas com problemas de simulação e não com os de síntese.

Este trabalho aplica a metodologia que envolve a tradução de especificações Estelle em descrições VHDL elaborada por Pirmez [11, 21]. Esta metodologia permite a implementação em *hardware*, de maneira semi-automática, de protocolos destinados a sistemas de comunicação de alta velocidade. A metodologia utilizada neste trabalho para a obtenção de circuitos VLSI é composta pelas seguintes etapas:

- a especificação do protocolo de comunicação em Estelle;
- a verificação do protocolo de comunicação através do uso de ferramentas específicas para a simulação e a geração automática de seqüências de testes, pertencentes a um ambiente próprio ao desenvolvimento de protocolos de comunicação. As ferramentas utilizadas neste trabalho se baseiam na linguagem Estelle e fazem parte de um ambiente de desenvolvimento do Grupo de Teleinformática e Automação (GTA) [22, 23, 24, 25], denominado C.A.D. Estelle;
- tradução das especificações em Estelle para descrições em VHDL (chamada de Mapeamento Estelle-VHDL), de forma a permitir o uso de um ambiente C.A.D. de projeto de circuitos, cuja entrada com o projetista de protocolos é uma descrição comportamental em VHDL do sistema. O C.A.D. de projeto de circuitos é composto de ferramentas adequadas ao projeto de circuitos VLSI tais como o Synopsys [26] para a síntese lógica e o Opus [27] para a geração de *layout*.

Esta metodologia integra, portanto, dois ambientes de projeto: um C.A.D. Estelle e um C.A.D. de projeto de circuitos, permitindo a implementação em *hardware* de protocolos a partir de uma descrição formal mais próxima do projetista de protocolos. O objetivo deste trabalho ao se utilizar esta metodologia é gerar um circuito integrado, que implemente o protocolo projetado no C.A.D. Estelle, a partir das descrições VHDL obtidas no processo de tradução, e utilizando-se as ferramentas de síntese de *hardware* adequadas e eventualmente disponíveis. Como parte integrante desta metodologia, o mapeamento de Estelle para VHDL é facilitado pois a linguagem VHDL possui estruturas similares às da linguagem Estelle.

Segundo o trabalho de Pirmez [11], para a aplicação desta metodologia algumas restrições são necessárias em ambas as linguagens para uma eficiente integração destes dois ambientes de projeto. As restrições são as seguintes:

- definição de um subconjunto de instruções da linguagem Estelle, permitindo descrições eficientes e facilitando o mapeamento para a linguagem VHDL. Com este objetivo, esta metodologia possui como ponto de partida uma versão mais simplificada e eficiente da linguagem Estelle, elaborada por Courtiat em [28]. Nesta sublinguagem, denominada Estelle*, o sistema de filas, as variáveis partilhadas e a cláusula de prioridade, entre outras, são revistas quanto a sua utilização. Estas instruções, além de gerar implementações pesadas, dificultam a obtenção de uma especificação com alto nível de abstração. Visando um mapeamento mais eficiente de uma especificação Estelle em uma descrição VHDL, este trabalho utiliza a linguagem Estelle* com algumas restrições adicionais, resultando na sublinguagem Estelle** definida no trabalho de Pirmez [11];

- definição de um subconjunto de instruções da linguagem VHDL permitindo descrições eficientes de protocolos em VHDL. A linguagem VHDL foi projetada para atender à descrição de *hardware* em diversos níveis dos domínios estrutural e comportamental, possuindo assim um conjunto de instruções muito amplo. Com este objetivo, este trabalho utilizará o mesmo subconjunto da linguagem VHDL empregado em [11] visando a síntese de alto nível. Este subconjunto da linguagem VHDL foi definido para ser aceito como entrada à ferramenta de síntese de alto nível AMICAL [29]. Este trabalho, entretanto, não utiliza este ambiente específico, fazendo uso somente do algoritmo de *scheduling* **DLS** (*Dynamic Loop Scheduling*) [30, 31] necessário para a síntese de alto nível e que é integrado ao AMICAL. Desta maneira, as restrições empregadas para a linguagem VHDL e definidas no trabalho de Pirmez [11] são também empregadas neste trabalho.

Na seção seguinte são apresentados os principais conceitos e características da linguagem Estelle. Ainda nesta seção, é apresentada a sublinguagem Estelle**, de modo a se obter um mapeamento eficiente na linguagem VHDL. Analogamente, na seção II.3 são mostrados os principais conceitos da linguagem VHDL, bem como a sublinguagem utilizada neste trabalho para a obtenção de circuitos VLSI eficientes. Em seguida, na seção II.4, é descrita a metodologia empregada, apresentando as diversas etapas tomadas durante a implementação em *hardware* de protocolos, enfocando o mapeamento Estelle-VHDL, a síntese de alto nível e a síntese lógica.

II.2 A Linguagem Estelle

A linguagem Estelle é uma Técnica de Descrição Formal desenvolvida pela ISO para a especificação de sistemas distribuídos. Estelle é utilizada para remover as ambigüidades encontradas nas descrições informais dos protocolos de comunicação.

II.2.1 Principais Conceitos

Um sistema distribuído especificado em Estelle é composto por vários componentes comunicantes organizados de forma hierárquica. Cada componente é especificado em Estelle pela declaração de um *módulo*, definindo um tipo dentro de uma especificação Estelle. Um sistema pode possuir mais de um módulo de um mesmo tipo. Variáveis de um determinado tipo de módulo especificam o que se chama de instâncias desse módulo.

Cada módulo Estelle é composto por duas partes: o cabeçalho (*module*) e um ou mais corpos (*body*). A visão externa é definida no cabeçalho, ou seja, é no cabeçalho que são definidos os pontos de interação, variáveis exportadas e o atributo de classe do módulo. O comportamento interno de um módulo é descrito dentro do seu corpo. A figura II.1 é um exemplo de um módulo, com seu cabeçalho e seu corpo.

```
module exemplo activity;    (* cabeçalho *)
        ip A: medium_interface (provider);
           export Z : integer;
end;
body exemplo_body for exemplo;    (* corpo *)
      corpo do modulo;
end;
```

Figura II.1: Exemplo de especificação de um módulo Estelle.

A especificação do comportamento de um módulo é baseado no modelo de máquina de estados finita (*FSM* - Finite State Machine). Esta máquina de estados é composta por um conjunto de transições cujo comportamento é descrito através de instruções em Pascal, com algumas restrições e extensões. A evolução de um módulo corresponde ao disparo das transições desta máquina de estados.

Um módulo especificado em Estelle pode incluir declarações de outros módulos. Isto aplicado repetidamente resulta numa estrutura hierárquica de módulos. A linguagem Estelle fornece meios de se criar instâncias de módulos-filhos definidos dentro do mesmo módulo-pai. Assim, uma especificação Estelle é dita ser composta por um conjunto de módulos organizados de forma hierárquica.

A comunicação entre módulos Estelle

A especificação da interface de comunicação de um módulo em Estelle é definida usando-se três conceitos: *pontos de interação*, *canais* e *interações*.

Cada módulo pode ser visto como uma caixa preta que possui pontos de acesso de entrada/saída chamados *pontos de interação*. Existem dois tipos de pontos de interação: interno e externo. Para cada ponto de interação existente em um módulo

```
channel medium_interface (user,provider);
    by user:
        msg1; msg3; msg5; ... msgn;
    by provider:
        msg2; msg4; msg6; ... msgm;
```

Figura II.2: Exemplo de especificação de um canal Estelle.

(interno ou externo) é associado um canal, que define dois conjuntos de interações. Um conjunto consiste de interações de saída e o outro conjunto consiste de interações de entrada, que fluem pelo ponto de interação. Interações são mensagens que são trocadas entre módulos através dos pontos de interação. Considere, por exemplo, a definição de um canal ilustrado na figura II.2, onde $msg_1,...,msg_n,msg_2,...,msg_m$ são interações.

Dois pontos de interação, p_1 e p_2, são declarados a seguir:

p_1 : medium_interface (provider);

p_2 : medium_interface (user);

O conjunto de interações que pode ser enviado via p_1 contém todas as interações especificadas por *provider* na definição do canal e o conjunto de interações que pode ser recebido em p_1 contém todas as interações especificadas por *user*. No ponto de interação p_2 acontece exatamente o oposto. As interações que podem ser enviadas por p_1 são recebidas por p_2 e vice-versa. Neste exemplo, p_1 e p_2 possuem papéis diferentes. Dois pontos de interação só podem ser conectados se tiverem papéis diferentes, uma vez que a troca de interações (mensagens) acontece entre eles.

Resumindo, as instâncias de módulos Estelle podem trocar mensagens, chamadas interações. Uma instância de um módulo pode enviar uma interação para outra instância de um outro módulo através de um canal previamente estabelecido entre seus respectivos pontos de interação. Uma interação quando recebida no ponto de interação de uma instância é armazenada na fila FIFO ilimitada associada à este ponto de interação. Esta fila FIFO pode pertencer tanto a um único ponto de interação (sendo chamada de fila individual - *individual queue*) como pode pertencer a vários pontos de interação (sendo chamada de fila comum - *common queue*).

Estruturação entre módulos Estelle

Estelle possui apenas dois tipos de ligação entre pontos de interação. No primeiro tipo, um ponto de interação externo de uma instância é conectado (*attached*) à um ponto de interação externo de sua instância-pai. No segundo tipo, dois pontos de interação são conectados (*connected*) se: estes pontos de interação pertencem às instâncias de módulos de mesma hierarquia; ou se um dos pontos de interação é interno à instância de um módulo e o outro é um ponto de interação externo de uma de suas instâncias-filhas; ou se os dois pontos de interação são internos à mesma instância.

Os atributos de um módulo Estelle

Um módulo Estelle pertence a uma das seguintes classes de atributo: "systemprocess", "systemactivity", "process" e "activity". Um módulo com atributo "systemprocess" pode ser estruturado em submódulos hierárquicos com atributos do tipo "activity" ou "process". Já um módulo com o atributo "systemactivity" pode ser estruturado em submódulos hierárquicos com atributos apenas do tipo "activity". O papel dos atributos é melhor ilustrado por dois casos particulares, vistos nas figuras II.3 e II.4, respectivamente.

No primeiro caso (figura II.3), todas as transições prontas para disparo (no máximo uma por instância) que não possuem o conflito de prioridade entre pai e filhos são selecionadas, enquanto que no segundo caso (figura II.4) somente uma delas é selecionada.

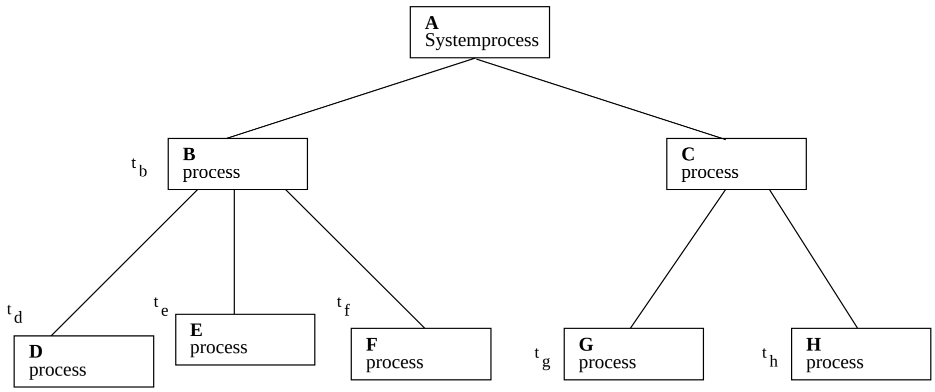


Figura II.3: Transições selecionadas: t_b, t_g e t_h.

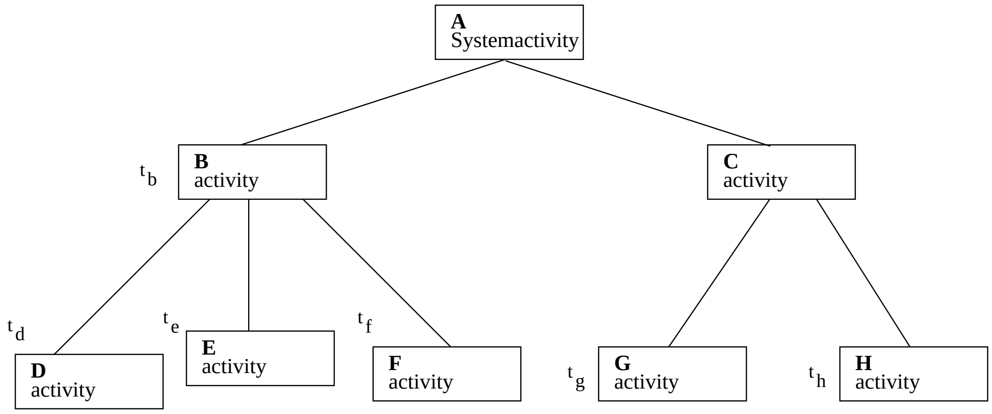


Figura II.4: Transições selecionadas: t_b ou t_g ou t_h.

Os Tipos de Paralelismo em Estelle

Estelle possui dois tipos de paralelismo: o paralelismo *síncrono* e o paralelismo *assíncrono*. O paralelismo assíncrono é permitido somente entre sistemas diferentes, ou seja, entre ações de diferentes instâncias pertencentes a diferentes sistemas. Já o paralelismo síncrono é permitido somente dentro de um sistema, isto é, entre ações de diferentes instâncias pertencentes ao mesmo sistema. Cada sistema executa seu próprio passo de execução. Um passo de execução é iniciado pela seleção de um conjunto de transições entre aquelas prontas para disparo. Estas transições são executadas em paralelo e somente após o término de todas é que o próximo passo de execução é iniciado. O paralelismo obtido é dito ser síncrono e é o utilizado pelos sistemas do tipo "process". Quando o sistema é do tipo "activity", a seleção do

```
initialize
   begin
     init A with User_body;
     init B with User_body;
     init C with Network_body;
     init D with Network_body;
     connect A.p1 to C.p1;
     connect B.p2 to D.p2;
   end;
```

Figura II.5: Exemplo da parte de iniciação de um corpo Estelle.

conjunto de transições prontas para disparo se reduz a uma única transição.

As partes do corpo de um módulo Estelle

A definição de um corpo (*body*) de um módulo Estelle é composta por 3 partes: a parte declarativa, a parte de iniciação e a parte de transições.

A parte declarativa é composta de declarações encontradas em Pascal (funções, procedimentos, variáveis e constantes) e de declarações de objetos em Estelle (canais, módulos, variáveis de tipo módulo, estados utilizados pela máquina de estados finita e pontos de interação internos). As variáveis de tipo módulo são instâncias de um certo módulo. Por exemplo, a declaração **modvar A,B: Z** define que A e B são variáveis de tipo módulo especificadas pelo cabeçalho do módulo Z.

A parte de iniciação (*initialize*) especifica valores iniciais de algumas variáveis pertencentes ao módulo como, por exemplo, variáveis locais e o estado inicial da máquina de estados do módulo. Além disso, são criadas as instâncias de módulos-filhos (*init*) e são estabelecidas as ligações entre as instâncias através de pontos de interação por meio das instruções *connect* e *attach*, como ilustrado na figura II.5.

A parte de transições descreve em detalhes o comportamento interno de um módulo. Esta parte é composta por um conjunto de declarações de transições onde cada transição começa com a palavra *trans*. Na figura II.6 é ilustrado um exemplo

```
trans
  from Repouso to Repouso
    when UICEP.IP_Data_Req
    name Datgint:
    begin
      Conv IPhexIPbin (SDU1,ipsrcbin,ipdestbin);
      .
      .
      output MIPCEP.AAL_Unitdata_Req (prox_no_aux1);
    end;
```

Figura II.6: Exemplo de uma transição em Estelle.

de uma transição.

II.2.2 Restrições à linguagem Estelle (Estelle**) para um mapeamento eficiente em VHDL

Courtiat em [28] definiu uma versão mais simplificada e eficiente da linguagem Estelle, denominada Estelle*. Esta versão permite obter especificações de protocolo com alto nível de abstração, utilizando uma semântica de paralelismo entre instâncias de módulos consistentes com a semântica definida nos modelos utilizados para a modelagem e validação de protocolos de comunicação (tais como máquina de estados e redes de Petri). A metodologia empregada neste trabalho para o obtenção de circuitos VLSI tem como ponto de partida a linguagem Estelle* com algumas restrições adicionais, visando a obtenção de uma tradução eficiente de uma especificação Estelle em uma descrição VHDL. À esta nova versão denomina-se Estelle** [11], cujas restrições são as seguintes:

- não existe a cláusula de prioridade associada a qualquer transição definida com a cláusula *when*;
- a única instância de módulo ativa são as folhas da árvore de instâncias de módulos;

- considerar a fila FIFO como sendo o mecanismo de comunicação utilizado. Tal mecanismo difere do utilizado por Estelle* mas é o mesmo utilizado por Estelle;

- restringir o uso da cláusula **delay** em Estelle para cláusulas da forma *delay(T1)* ou *delay(T1,T1)* de modo a ser mapeada em VHDL.

As duas primeiras restrições torna possível representar de uma forma assíncrona o paralelismo existente entre instâncias de módulo Estelle do tipo activity dentro do tipo de especificação systemactivity. Esta restrição se faz necessária uma vez que não existem conceitos em VHDL equivalentes aos tipos de paralelismo existentes entre instâncias de módulos Estelle.

II.3 A Linguagem VHDL

VHDL é uma linguagem padrão desenvolvida pelo Departamento de Defesa Americano (DoD) para a descrição e a simulação de estruturas de *hardware* em diferentes níveis de abstração e domínios de descrição, indo do ***comportamental*** ao ***estrutural***.

A descrição comportamental permite a especificação das funções executadas por um projeto, sem se referir à sua estrutura interna, utilizando estruturas familiares da linguagem de programação. No domínio estrutural, as estruturas que implementam um dado comportamento são descritos, bem como sua decomposição em subprojetos e a interligação entre eles.

A linguagem VHDL é uma linguagem multi-domínios e multi-níveis. Além disso, a linguagem VHDL permite que um projeto seja simulado antes de ser fabricado, de forma que os projetistas possam comparar alternativas rapidamente e testar sua correção sem o atraso e o custo de um protótipo de *hardware*.

II.3.1 Principais Conceitos

O componente básico de uma especificação em VHDL é uma entidade. Uma entidade é composta por duas partes: uma interface (*entity*) e um conjunto de uma ou mais arquiteturas (*architecture*). A interface define as características observáveis externamente, através das portas de entrada ou de saída, enquanto uma arquitetura define a implementação desta entidade. A figura II.7 ilustra um exemplo de uma entidade com sua respectiva arquitetura.

A declaração *port* define todas as portas que uma entidade irá possuir. As portas são sinais que podem ser de entrada, de saída ou de entrada/saída, possibilitando que uma entidade troque informações com outras entidades VHDL.

Uma vez que uma entidade teve sua interface especificada pela declaração *entity*, sua implementação pode ser descrita por diferentes arquiteturas. Como exemplo, uma arquitetura pode simplesmente descrever o comportamento de entrada/saída da entidade enquanto outras podem incluir informações estruturais específicas através de um conjunto de componentes dispostos hierarquicamente. Em outras palavras, uma arquitetura pode ser descrita tanto de maneira comportamental como de maneira estrutural.

Quando descrito estruturalmente, os sub-módulos de uma arquitetura podem ser descritos como blocos. Um bloco é uma unidade de módulo dentro da descrição estrutural, com sua própria interface, conectada a outros blocos ou portas por meio de sinais. Por outro lado, quando descrito de maneira comportamental, uma arquitetura é implementada através da instrução *process*. A instrução *process* é executada de forma concorrente, formada por um conjunto de instruções seqüenciais.

II.3.2 Restrições à linguagem VHDL para a Obtenção de Circuitos VLSI Eficientes

A linguagem VHDL foi projetada para atender à descrição de *hardware* em diversos níveis dos domínios estrutural e comportamental, possuindo assim um conjunto

```
entity exemplo is
   port(msg2: out bit; msg4: out bit; msg6: out bit; msgm: in bit;
      msg1: in bit; msg3: in bit; msg5: in bit; msgn in bit);
end exemplo;
architecture exemplo_body of exemplo is
   begin
     process
       begin
         IF (msg1 = '1') THEN
         .
         .
         msg3 <= buffer_exemplo;
         END IF;
       end process;
   end exemplo_body;
```

Figura II.7: Exemplo de uma Entidade VHDL.

de instruções muito amplo. Além disso, a linguagem VHDL possui estruturas que não tem equivalências ou possíveis realizações em *hardware*.

Com o objetivo de obter circuitos VLSI eficientes, Pirmez [11] definiu uma versão simplificada da linguagem VHDL capaz de fornecer descrições mais eficientes a fim de se obter arquiteturas eficientemente sintetizáveis. Esta sublinguagem foi definida para ser aceita como entrada à ferramenta de síntese de alto nível AMICAL [29]. Este trabalho, entretanto, não utiliza este ambiente específico, não tendo, a princípio, nenhuma restrição aplicada à linguagem VHDL. Contudo, uma vez seguindo a metodologia elaborada por Pirmez [11], optou-se em empregar neste trabalho as mesmas restrições empregadas para a linguagem VHDL no trabalho de Pirmez [11]. As restrições aplicadas à linguagem VHDL, definindo uma sublinguagem VHDL que servirá como entrada à síntese de alto nível, são as seguintes:

- só serão aceitas as declarações de sinais em uma entidade VHDL. Não será utilizado instruções *generic*;
- não serão utilizados os seguintes tipos em VHDL: *physical*, *floating point*, *enu-*

meration, *arrays* e *records*;

- não utilizar a instrução **wait on...**. Será utilizado somente intruções **wait until...** nas descrições VHDL;
- não utilização da instrução *block*;
- cada arquitetura deve conter somente uma única instrução concorrente e esta deve ser uma instrução *process*.

II.4 A Metodologia Adotada

A metodologia adotada neste trabalho para a síntese de circuitos VLSI que implementa o protocolo IPv6 é ilustrada pelo diagrama da figura II.8. O procedimento adotado tem como ponto de partida uma especificação Estelle de um protocolo corretamente compilada e simulada (1) obtida com o uso de um ambiente adequado para o desenvolvimento de protocolos em Estelle [22, 24, 25]. Em seguida, esta especificação é traduzida numa descrição VHDL através de uma estratégia de mapeamento Estelle-VHDL (2), passo este necessário para a utilização de um conjunto de ferramentas de projeto de circuitos VLSI, denominado na figura de C.A.D. de projeto de circuitos. Após este mapeamento, o comportamento de uma entidade VHDL pode ser descrito tanto de maneira comportamental como de maneira estrutural. Neste trabalho apenas a descrição comportamental foi considerada, uma vez que esta descrição é mais familiar ao projetista de protocolos além de representar, com uma maior semelhança, a especificação funcional do protocolo.

A entrada do C.A.D. de projeto de circuitos é, portanto, uma descrição comportamental em VHDL do sistema (do protocolo) (3). O C.A.D. de projeto de circuitos é composto de ferramentas padrão de projeto VLSI tais como o Synopsys [26] para a síntese lógica e o Opus [27] para a geração de *layout*. O Synopsys pode ser utilizado na simulação de descrições comportamentais e estruturais em VHDL e para a síntese lógica de descrições estruturais. Já o Opus é utilizado na geração de *layout* das estruturas VLSI, a partir do esquema lógico.

A etapa seguinte do processo, consiste na obtenção de uma descrição comportamental VHDL do sistema corretamente simulada, com o auxílio da ferramenta de simulação existente no Synopsys, o *vhdldbx* (3,4). A síntese de alto nível propriamente dita consiste na aplicação de um algoritmo de *scheduling* (5,6) à esta descrição comportamental obtida em (4). A síntese de alto nível tem como definição a tradução de uma descrição comportamental em uma descrição estrutural. Entretanto, para se obter uma descrição estrutural, a descrição comportamental do sistema, que é a entrada do processo de *scheduling*, deve ser formada por um único processo principal constituído por um conjunto de componentes (*component*). Um componente é um subsistema que é representado por uma entidade VHDL. Desta maneira, se a descrição comportamental do sistema possuir subsistemas, o algoritmo de *scheduling* deve ser empregado em todos seus subsistemas, gerando descrições comportamentais sem dependências de dados e de controle. A descrição comportamental do sistema, após se aplicar a síntese de alto nível, resultará numa descrição estrutural sem dependências.

No caso do projeto de um protocolo, especificado inicialmente em Estelle, os subsistemas existentes em uma descrição comportamental são facilmente identificados, pois os submódulos (módulos-filhos), contidos numa especificação Estelle, correspondem diretamente a subsistemas descritos por entidades VHDL. Entretanto, a especificação em Estelle do protocolo utilizado neste trabalho, o protocolo IPv6, não possui submódulos. Isto corresponde a dizer que sua descrição comportamental em VHDL equivalente também não terá subsistemas (componentes). Desta maneira, ao se aplicar a síntese de alto nível nesta descrição comportamental ter-se-á como resultado uma descrição ainda comportamental só que sem dependências de dados e de controle, e não uma descrição estrutural. Esta descrição resultante corretamente simulada é passível de ser sintetizada logicamente, através da ferramenta de síntese lógica existente no Synopsys, *design_analyzer*.

A partir desta descrição sem dependências, é possível a síntese lógica no Synopsys. O algoritmo de *scheduling* utilizado foi o algoritmo DLS (*Dynamic Loop Scheduling*) proposto por Rahmouni [30, 31]. Novamente utiliza-se a ferramenta *vhdldbx* com o intuito de se verificar, agora, a validade dessa descrição VHDL sem

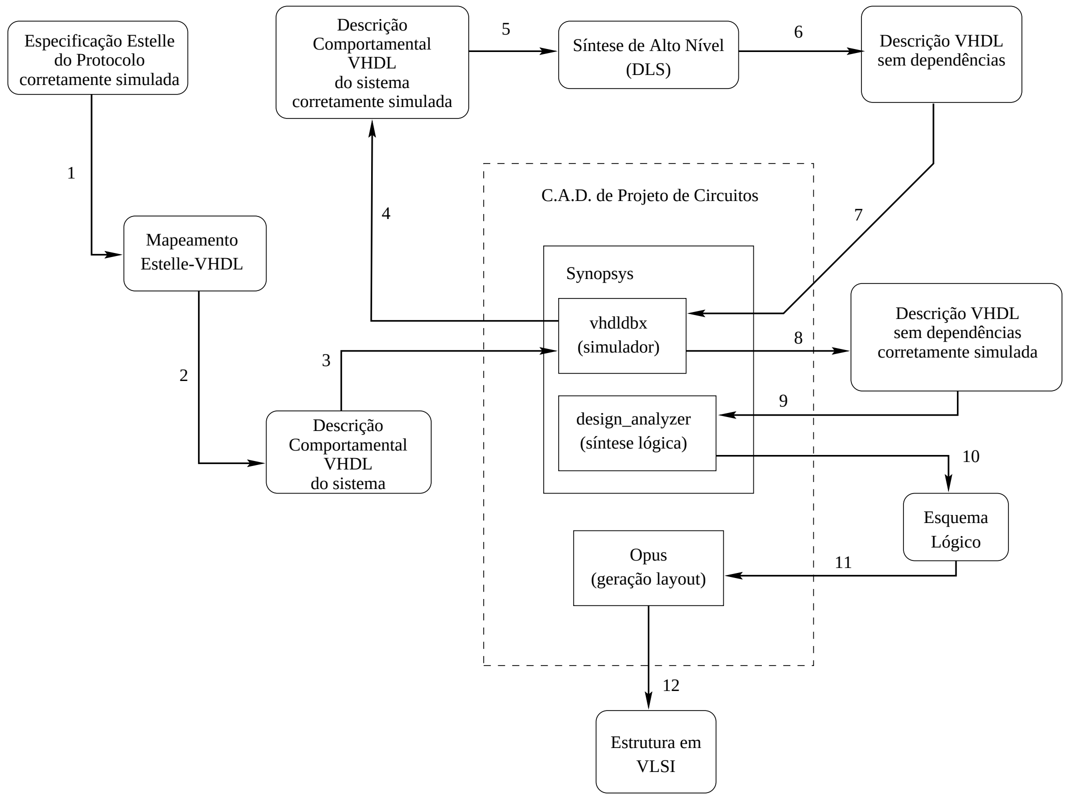


Figura II.8: A Metodologia adotada na obtenção de circuitos VLSI.

dependências (`7,8`).

Esta descrição corretamente simulada é então submetida à ferramenta de síntese lógica *design_analyzer* (`9,10`). O esquema lógico resultante é utilizado como descrição de entrada da ferramenta Opus para a geração do *layout* desta estrutura física (`11,12`).

A próxima subseção apresenta em maiores detalhes a estratégia de mapeamento de especificações Estelle em descrições VHDL.

II.4.1 A Estratégia de Mapeamento Estelle-VHDL

Uma especificação Estelle é composta por um conjunto de *módulos* organizados de forma hierárquica enquanto que uma descrição VHDL é composta por um conjunto de *entidades* independentes. Cada módulo Estelle pode ser mapeado em uma entidade VHDL conforme mostrado na figura II.9 (mapeamento `1`).

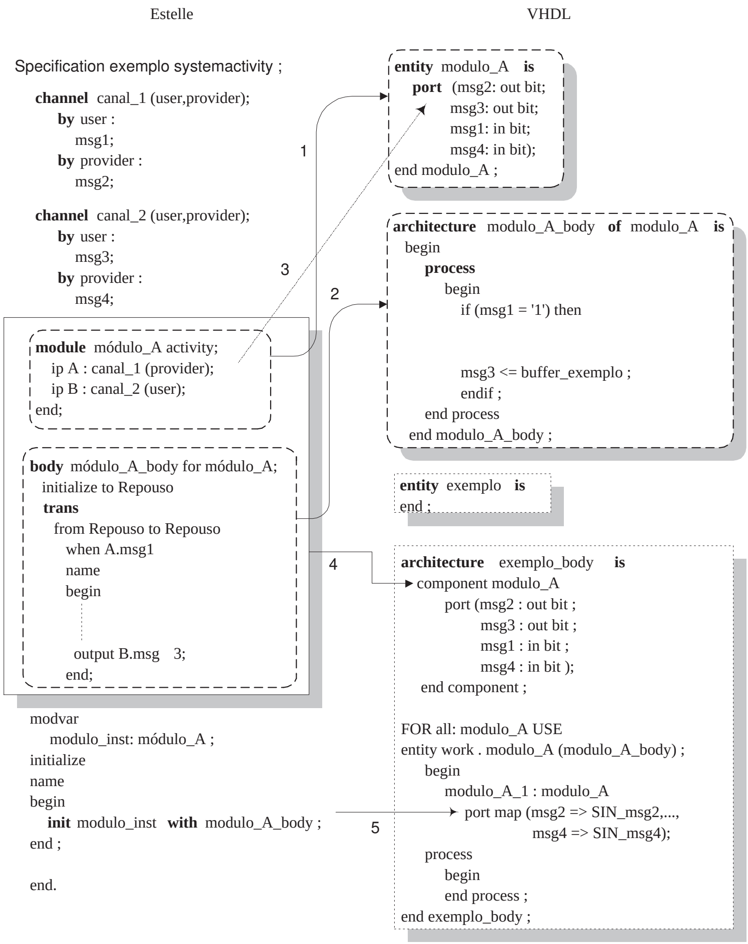


Figura II.9: O Mapeamento Estelle-VHDL.

A hierarquia existente entre módulos Estelle é mantida em VHDL através da declaração *component* como mostrado na figura II.9 (mapeamento 4). Cada uma destas declarações corresponde à uma entidade e sua respectiva descrição é armazenada em bibliotecas. A instrução *component instantiation* do VHDL é usada para criar instâncias de entidades VHDL (análoga à função *init* do Estelle) cujas portas são conectadas aos sinais correntes ou às portas da entidade (análogas às funções *connect* e *attach* do Estelle) como mostrado na figura II.9 (mapeamento 5). A ligação de uma entidade VHDL à sua instância é efetuada através da declaração *FOR ALL*. Nesta declaração são feitas chamadas às entidades VHDL que irão compor a hierarquia entre entidades. Estas entidades são armazenadas em arquivos independentes.

Cada módulo Estelle é composto de duas partes: o cabeçalho (*module*) e um ou mais corpos (*body*). A visão externa é definida no cabeçalho e o comportamento interno é descrito no corpo. Cada descrição VHDL também é composta de dois segmentos: uma interface (*entity*) e um conjunto de uma ou mais arquiteturas (*architecture*). Desta forma, o par *module-body* em Estelle corresponde ao par *entity-architecture* em VHDL, respectivamente, como é mostrado na figura II.9 (mapeamentos 1 e 2). O módulo raiz de Estelle (*Specification*) não possui uma visão externa e, em conseqüência, sua correspondente interface (*entity*) é vazia.

No cabeçalho de um módulo Estelle, cada interação (que no caso de protocolos de comunicação é uma primitiva de serviço) que faz parte de um conjunto de interações pertencente a um dado ponto de interação irá corresponder a um sinal de uma entidade VHDL. O tipo de sinal será *input* ou *output* dependendo do papel tomado pela interação, podendo ser de recepção ou de transmissão, respectivamente, como mostrado na figura II.9 (mapeamento 3). A comunicação entre dois módulos Estelle se faz através de um *canal* entre dois pontos de interação. Este *canal* permite que dois módulos troquem mensagens em ambas as direções. Em VHDL, entretanto, a única forma de comunicação permitida entre entidades é feita através de sinais, que são unidirecionais. Com isso, é necessária a criação de dois sinais em cada entidade VHDL para representar um *canal*.

O comportamento de um módulo Estelle é descrito pelas partes de iniciação e de transição do corpo. A execução de ambas as partes é seqüencial. O mapeamento de cláusulas Estelle em estruturas VHDL é simples e o conjunto em VHDL é incluído na declaração *process*. As cláusulas de Estelle "**from** *estado*", "**when** *ponto-de-interação.interação*", "**provided** *expressão*", "**to** *estado*" e "**output** *ponto-de-interação.interação(msg)*" são convertidas diretamente em VHDL para "**when** *state => da estrutura case*", "**wait until** *condição*", "**if** *expressão* **then...**", "**state** := *atribuição ao estado*" e "*interação <= msg*". As instruções em Estelle "**procedure** *nome (lista de parâmetros)*" e "**function** *nome (lista de parâmetros) : tipo do parâmetro do retorno*" são diretamente convertidas em declarações VHDL equivalentes.

A estratégia de mapeamento para a tradução de especificações Estelle em descrições VHDL pode ser dividida em três partes. Inicialmente são identificados todos os tipos de módulos em Estelle e seu relacionamento hierárquico. A partir desta identificação, cada tipo de módulo é implementado como uma entidade VHDL independente e a sua hierarquia é obtida através da declaração *component*. Por fim, criam-se entidades gerenciadoras de filas, uma para cada entidade criada e uma para cada grupo de entidades filhas. O presente trabalho não aborda esta última etapa.

Para estruturar cada entidade VHDL, deve-se primeiramente determinar os estados e transições pertencentes ao corpo de cada tipo de módulo Estelle. Estes estados irão corresponder aos estados da máquina de estados de cada entidade VHDL. Entretanto, a linguagem VHDL não possui o conceito de máquina de estados implícito na linguagem, sendo necessário, portanto, desenvolver procedimentos que implementem tal conceito. Este trabalho irá adotar o procedimento proposto em [11] em que a entidade VHDL será criada com sua respectiva arquitetura contendo uma descrição comportamental da máquina de estados obtida diretamente da especificação Estelle. As etapas necessárias à estruturação das entidades VHDL são as seguintes:

- o conjunto dos diferentes estados de um módulo Estelle dará origem a uma estrutura **case** em VHDL. Uma variável (*Estado*) é definida como o estado atual da máquina de estados em VHDL;

- para cada estado, as transições correspondentes ao estado são agrupadas por interação recebida. A instrução "**wait until** *conjunto_de_interações*" suspenderá a execução até que a cláusula *conjunto_de_interações* seja satisfeita. Se existir mais de uma condição *conjunto_de_interações* para um mesmo estado, uma construção "**if**" do VHDL será colocada logo após a instrução "**wait until** *conjunto_de_interações*".

Uma vez terminada a construção da máquina de estados, esta deve ser colocada no interior de uma estrutura *process* do VHDL de modo a permitir que a máquina de estados seja continuamente executada. Esta máquina de estados permanecerá suspensa enquanto não ocorrer uma interação (mudança de sinal). Se mais de um processo estiver ativo ao mesmo tempo, eles serão executados de forma concorrente.

II.4.2 O Estilo de Descrição Adotado

A linguagem VHDL foi projetada para atender descrições de projeto no domínio comportamental e no estrutural. Conseqüentemente, essa linguagem possui uma variedade de estruturas que permite a descrição do mesmo comportamento de maneiras diferentes.

Após o mapeamento de uma especificação Estelle em uma descrição VHDL, o comportamento de uma entidade VHDL pode ser descrito tanto de maneira comportamental como de maneira estrutural. Entretanto, no trabalho de Pirmez [11] apenas a descrição comportamental foi considerada no processo de síntese de alto nível, devido à restrição imposta pela ferramenta de síntese AMICAL, que exclui a possibilidade de descrições estruturais. Neste trabalho, apesar de não empregar esta ferramenta específica, também se considera apenas a descrição comportamental, uma vez que esta descrição é mais familiar ao projetista de protocolos além de representar, com uma maior semelhança, a especificação funcional do protocolo.

Partindo-se de uma descrição comportamental, após o mapeamento Estelle-VHDL um protocolo pode ser especificado por diferentes estilos de descrição comportamental VHDL. Tais estilos produzem o mesmo resultado a nível de simulação. Entretanto, essas descrições simuladas corretamente diferem na qualidade da estrutura gerada na síntese de alto nível (HLS - *High-Level Synthesis*). A qualidade da descrição estrutural de saída fornecida pela HLS é medida pelo número de estados e caminhos gerados pelo processo específico de *scheduling* executado durante uma HLS. Em outras palavras, os resultados da HLS dependem da qualidade da descrição de entrada, que por sua vez, depende do tipo das instruções e da ordem em que aparecem em uma descrição. Em [11] foram propostos e analisados diversos estilos de descrição. Neste trabalho será utilizado o estilo que apresentou os melhores resultados, por possuir o menor número de camihos e de estados gerados, refletindo-se em uma menor área e, conseqüentemente, em um menor custo. O estilo de descrição adotado neste trabalho é descrito abaixo:

- a modelagem do protocolo em máquina de estados (FSM) corresponde ao processo VHDL. Este processo é composto de dois *loops*: um externo, que modela a reiniciação do processo e um interno que modela a FSM do protocolo;
- a instrução "**wait until...**" é posta no início do *loop* mais interno, onde os sinais de controle são testados;
- a máquina de estados pode ser construída utilizando a instrução "**case...**" ou a construção "**if...then...elsif...end if**". Em ambas as soluções, a escolha de testar inicialmente o estado ou as condições de entrada depende se o processamento é mais dependente do estado ou das condições de entrada;
- após o uso da instrução "**case...**" ou da construção "**if...then...elsif...end if**", os diferentes sinais de controle recebem o valor zero.

A próxima subseção apresenta os principais conceitos da síntese de alto nível bem como o algoritmo de *scheduling* DLS (*Dynamic Loop Scheduling*) proposto por Rahmouni [30, 31] que será aplicado à descrição comportamental VHDL do protocolo IPv6 de forma a produzir uma descrição de máquina de estados sem dependências de dados e de controle.

II.4.3 A Síntese de Alto Nível

Um circuito integrado VLSI pode ser descrito por um dos três domínios de especificação: o *comportamental*, o *estrutural* e o *físico*. O domínio *comportamental* especifica o que o sistema faz e não como ele é construído. Neste domínio o projeto corresponde a uma ou mais caixas pretas onde são considerados apenas a sua interface com o meio externo e seu comportamento. O domínio *estrutural* é a ponte entre o domínio comportamental e o domínio físico. O domínio *estrutural* especifica como os componentes de um sistema são conectados para efetuar o comportamento prescrito. No domínio *físico* é ignorado o que o sistema faz. Neste domínio é descrita a alocação de módulos de *hardware* na área destinada à fabricação do circuito bem como a estrutura física dos módulos.

Cada domínio pode ser hierarquicamente dividido em vários níveis de abstração, fazendo com que a especificação de um circuito integrado seja classificada quanto ao seu domínio e quanto ao seu nível. Dentre os níveis existentes pode-se citar o nível *arquitetural*, o de *transferência de registros* (RTL - *Register-transfer level*), o *lógico* e o de *circuito*. A relação entre os domínios e os níveis de abstração pode ser melhor entendida através do diagrama em Y de Gajski [32], mostrado na figura II.10.

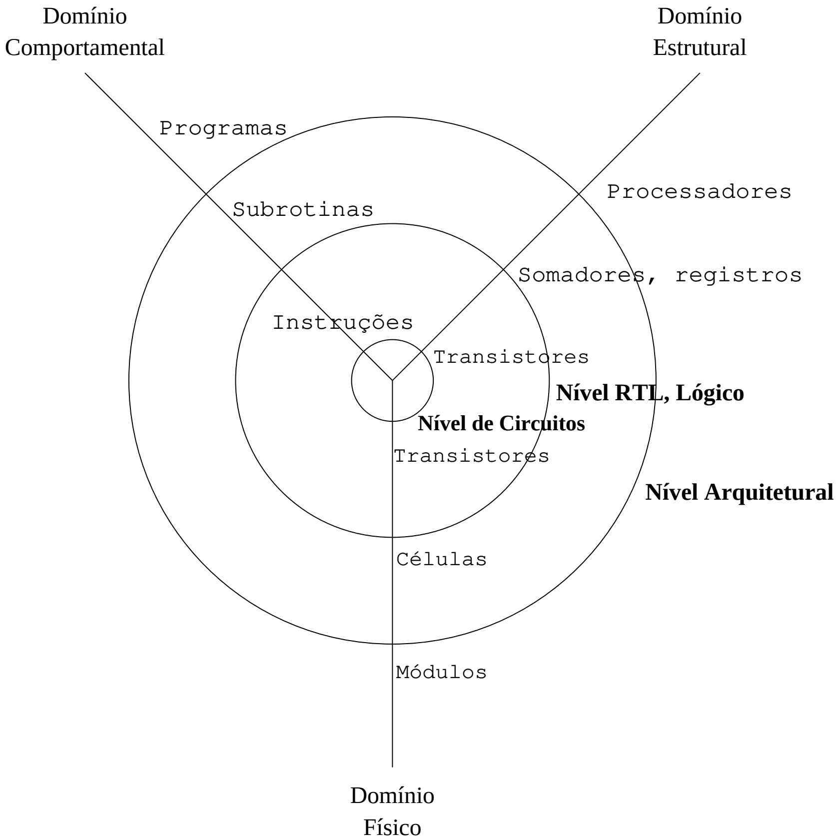


Figura II.10: Diagrama Y: domínios de descrição e níveis de abstração.

Um circuito integrado VLSI pode ser projetado de maneira semi-automática a partir de especificações do sistema em alto nível. Entende-se por especificação de alto nível o mapeamento de descrições comportamentais no domínio estrutural que esteja pelo menos no nível de transferência de registros ou acima.

A síntese de alto nível corresponde à tradução de uma especificação *comporta-*

mental em uma especificação *estrutural*. Desse modo, a síntese de uma especificação em Estelle é obtida após o mapeamento de uma especificação em Estelle em descrições comportamentais em VHDL.

A síntese de alto nível envolve os seguintes passos:

- macro_*scheduling*;
- alocação;
- micro_*scheduling*;
- geração da arquitetura.

A partir da descrição comportamental do sistema que serve de entrada à síntese de alto nível, o macro_*scheduling* produz uma máquina de estados finita apresentada sob a forma de tabela de estados onde cada transição corresponde à execução de um passo de controle. Neste estágio do processo, cada operação pode ser executada em vários ciclos de relógio. Uma transição também é chamada macro_ciclo sendo descrita por um grafo de fluxo de dados (*data-flow graph*). A maior parte do processo de síntese consiste, assim, em aplicar sucessivamente os passos de *scheduling* e *alocação* com o objetivo de refinar cada macro_ciclo em um conjunto de passos de controle básicos que podem ser executados, cada um, em um ciclo de relógio. Estes passos de controle básicos são também chamados de micro_ciclos.

A alocação é executada após o *scheduling* e as informações de entrada necessárias nesta etapa são: uma descrição *scheduled*, especificada por um conjunto de grafos de fluxo de dados, e uma biblioteca de unidades funcionais externas. O passo de alocação tem como objetivo associar uma unidade funcional a cada operação existente na tabela de estados, além de ligar as instâncias de componentes a um componente da biblioteca. Este último processo é chamado de *binding*. O segundo passo de *scheduling*, denominado de micro_*scheduling*, é então executado de acordo com o esquema de execução de cada operação. Cada operação é decomposta em um conjunto de instruções que são programadas em micro_ciclos. Cada micro_ciclo contém um conjunto de instruções (*transfers*) paralelas que podem ser executadas em um ciclo básico de relógio.

O último passo de síntese de alto nível é a geração da arquitetura. A descrição detalhada ao nível de ciclo de relógio é mapeada em uma arquitetura composta de um *datapath* e um controlador. A rede de comunicação pode ser composta de barramentos, multiplexadores e registros. O número de barramentos e multiplexadores é fixado de acordo com as instruções paralelas requisitadas pela arquitetura.

O ambiente AMICAL [29], utilizado no trabalho de Pirmez [11], e a ferramenta *behaviour_compiler*, existente no Synopsys [26], são exemplos de sistemas de síntese de alto nível. Entretanto, a não-disponibilidade destas ferramentas ao longo do desenvolvimento deste trabalho resultou na escolha de empregar, somente, o algoritmo de *scheduling* **DLS** (*Dynamic Loop Scheduling*) [30, 31] como etapa fundamental da síntese de alto nível. O algoritmo DLS é integrado ao ambiente AMICAL e adequado à síntese de compiladores comportamentais dominados por fluxo de controle [32]. Além disso, não houve uma preocupação maior com a alocação neste trabalho uma vez que a própria natureza do sistema que está sendo sintetizado (o protocolo IPv6) possui um mínimo de paralelismo, sendo sua grande parte de natureza seqüencial.

O algoritmo DLS tem como entrada uma descrição comportamental VHDL composta pelo par *entity/architecture*. Este algoritmo é baseado nos princípios de escalonamento de caminhos (*path scheduling*) [33, 34] mas reduz de maneira significativa o número de caminhos gerados e, conseqüentemente, o custo de computação e a complexidade do *hardware* gerado posteriormente. Este algoritmo consiste dos seguintes passos:

- geração do grafo de fluxo de controle;
- geração de caminhos;
- geração de estados e transições.

A principal representação usada pelo DLS é um grafo de fluxo de controle (CFG - *Control-Flow Graph*). Este grafo fornece um meio conveniente de representar as propriedades inerentes de circuitos caracterizados por fluxo de controle, que são descritos por *loops* aninhados e por construções de sincronismo tais como a construção *wait*. Este CFG contém nós, representando as operações da descrição original, e ramos, indicando os fluxos de sinal da descrição original. O CFG identifica três tipos diferentes de operações: a construção *wait*, as construções de fluxo de controle (como *if* e *while*) e expressões.

A mais importante característica deste CFG é a introdução do nó *wait*. Um nó *wait* significa sempre uma mudança de estado que será efetuada. Isto causa a interrupção incondicional da geração de caminhos, evitando desta maneira o problema da explosão de caminhos [31].

Este grafo pode ser definido como

$$G = (N, F), \tag{II.1}$$

onde N é o conjunto de nós e F é o conjunto de ramos definindo a precedência entre os nós. O conjunto de nós é definido por:

$$N = N_w \cup N_f \cup N_e, \tag{II.2}$$

onde N_w é o conjunto de construções *wait*, N_f o conjunto de construções de fluxo de controle e N_e o conjunto de expressões.

Um caminho $C = (n_1, n_2, ..., n_i)[n_s]$ é definido como um conjunto de nós, começando pelo nó n_1 e tendo como seu sucessor um outro caminho iniciado por n_s. Novos caminhos são gerados conforme se progride no CFG começando-se pelo primeiro nó. A geração de um caminho é interrompida e um novo caminho se inicia quando uma das duas condições abaixo é satisfeita:

- uma construção *WAIT* é encontrada (condição C_1);
- o novo nó encontrado possui dependência de dados com relação a um dos nós já existentes no caminho (condição C_2).

A tradução de descrições comportamentais em VHDL para o CFG é feita de forma simples. No capítulo IV, este algoritmo será aplicado ao protocolo IPv6 tornando mais claro o entendimento dos conceitos apresentados anteriormente.

Partindo-se das representações dos CFGs, todos os caminhos são então calculados. O primeiro caminho inicia-se com o primeiro nó do CFG. Nós sucessivos são seqüencialmente adicionados ao caminho até que uma das condições mostradas anteriormente (C_1 e C_2) seja satisfeita.

Na fase de geração de estados e transições, cada caminho gerado irá constituir uma transição. As transições em conjunto com os estados gerados irão produzirá uma máquina de estados (FSM). Transições adicionais serão geradas devido aos *loops* implícitos de todas as construções *wait*. Todos os caminhos iniciados com o mesmo nó serão colocados juntos no mesmo estado.

O resultado do DLS é uma tabela de estados e transições que pode ser manipulada pela ferramenta de síntese lógica existente no Synopsys, *design_analyzer*.

II.4.4 A Síntese Lógica

Síntese é um método automático de converter uma descrição de alto nível de abstração em uma descrição de baixo nível. A síntese lógica envolve o mapeamento de uma descrição no nível RTL do domínio comportamental em uma *netlist* no nível de portas lógicas. Esta *netlist* especifica as interconexões de macrocélulas descritas ao nível de portas lógicas. Na figura II.11 é mostrada a estrutura geral de um ambiente de síntese lógica.

A descrição RTL é um estilo que especifica todos os registradores e a lógica combinacional entre eles. Os registradores podem ser indicados seja explicitamente através da instanciação de componentes seja implicitamente através de inferência. A lógica combinacional pode ser descrita através de equações lógicas, instruções de controle seqüenciais, subprogramas ou instruções concorrentes. No primeiro passo da síntese lógica a descrição RTL é convertida numa descrição booleana não otimizada como é mostrado na figura II.12.

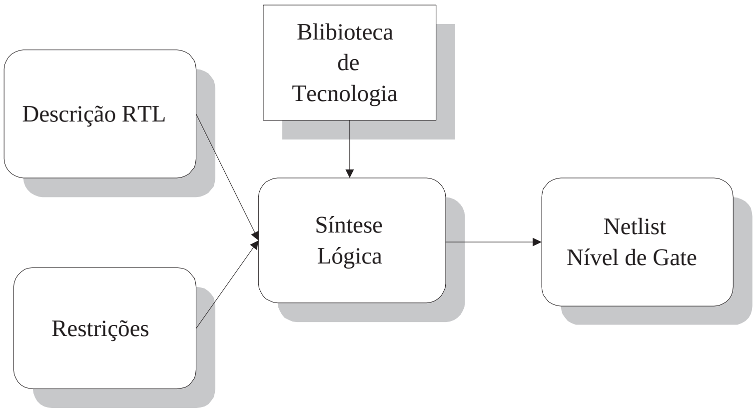


Figura II.11: Estrutura geral de um ambiente de síntese lógica.

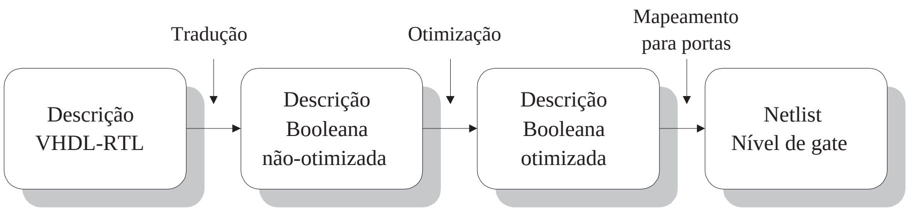


Figura II.12: Etapas da síntese lógica.

A tradução consiste em converter todas as construções *if*, *case*, *loop*, atribuições condicionais de sinal e instruções de atribuição de sinal em equivalentes booleanos. *Flip-flops* e *latches* podem ser instanciados ou inferidos.

O segundo passo da síntese lógica é justamente otimizar a descrição booleana não-otimizada resultante da tradução. A otimização tem como objetivo eliminar os estados repetidos que eventualmente são criados durante a síntese de alto nível, uma vez que para cada construção de fluxo de controle um novo estado é gerado. Como resultado tem-se uma descrição booleana otimizada.

O último passo da síntese lógica consiste em mapear as expressões booleanas otimizadas em uma *netlist* a partir de bibliotecas de portas lógicas, tais como NAND e NOR.

II.5 Conclusão

Neste capítulo, foram apresentados os principais conceitos das linguagens Estelle e VHDL, cada uma com suas restrições para a obtenção de circuitos VLSI eficientes ao final da metodologia empregada. A metodologia empregada para a obtenção de circuitos VLSI também foi descrita, dando-se um enfoque maior nos passos referentes ao mapeamento Estelle-VHDL, à síntese de alto nível e à síntese lógica.

A síntese de alto nível consiste principalmente no uso de um algoritmo de *scheduling*. Neste capítulo foi apresentado o algoritmo DLS, que será aplicado ao protocolo IPv6 no capítulo IV. Após a utilização deste algoritmo, obtém-se uma descrição sem dependências de dados e de controle, uma vez que a descrição comportamental deste protocolo não possui componentes.

A síntese lógica envolve o mapeamento de uma descrição no nível RTL do domínio comportamental em uma *netlist* no nível de portas lógicas. A síntese lógica será realizada na descrição comportamental sem dependências do protocolo IPv6.

No capítulo a seguir, é apresentado o protocolo IPv6, mostrando suas principais caractrísticas e são destacados os conceitos envolvidos na abordagem "IP over ATM".

Capítulo III

O Protocolo IPv6 e a Abordagem "IP over ATM"

III.1 Introdução

A REDE Internet é uma rede virtual construída pela interconexão de diferentes tecnologias de redes (locais, metropolitanas e de longa distância). A arquitetura Internet, constituída pela família de protocolos TCP/IP, é baseada principalmente em um serviço de transporte orientado a conexão, fornecido pelo *Transmission Control Protocol* [35] e em um serviço de rede não orientado a conexão (datagrama não confiável), fornecido pelo *Internet Protocol* (IP) [36].

A Internet vem sendo utilizada por décadas sem nenhuma mudança em sua arquitetura. Entretanto, com o surgimento de novas aplicações que exigem o uso de redes de computadores de alto desempenho e com o crescimento explosivo da Internet, uma atenção especial vem sendo dada ao protocolo IP. Pesquisas foram realizadas e chegou-se ao desenvolvimento de uma nova versão do protocolo IP, chamado de IPv6 [12, 13, 14]. O protocolo IPv6 pertence à nova geração de protocolos e elaborado para suportar os atuais requisitos de desempenho das novas aplicações.

Atualmente, as redes locais (LANs - *Local Area Networks*), que possuem acesso compartilhado ao meio físico, começam a se mostrar limitadas em termos de desempenho devido ao crescente número de estações interconectadas e ao surgimento das aplicações multimídias que exigem grande banda passante. A tecnologia ATM [15] é uma tecnologia emergente que oferece uma alta banda passante além de suportar tráfegos multimídias, despertando, assim, grande interesse em sua utilização em redes locais [37, 38]. A simples implantação de novas LANs baseadas na tecnologia ATM mostra-se, no entanto, inviável dada a enorme base instalada. Os custos de mudança para estas novas redes integralmente baseadas na tecnologia ATM seriam muito grandes, o que inviabiliza a introdução da nova tecnologia a curto prazo. Como conseqüência, é necessário uma migração suave para as redes ATM, com soluções intermediárias, permitindo que as aplicações existentes possam continuar sendo utilizadas.

A compatibilização das redes ATM com as LANs tradicionais apresenta alguns problemas importantes, fruto de diferenças fundamentais entre as duas tecnologias. O modo de acesso às LANs é não orientado a conexão, enquanto que a tecnologia ATM se baseia no serviço orientado a conexão. Desta forma, é necessária a emulação de um serviço não orientado a conexão utilizando-se um serviço inferior orientado a conexão.

Existem basicamente duas abordagens possíveis na implementação da emulação de LANs sobre redes ATM. A primeira abordagem consiste em se implementar o protocolo IP diretamente sobre a tecnologia ATM, sendo comumente chamada de "IP over ATM". Este é o caminho que vem sendo adotado pelo grupo do IETF (*Internet Engineering Task Force*) responsável pela RFC 1577 [16]. A segunda abordagem resulta da solução proposta pelo ATM Forum, denominada "LAN Emulation" (LANE) [39], que consiste na implementação de uma camada emulando os serviços da subcamada MAC (*Medium Access Control*) das LANs tradicionais sobre a camada de adaptação ATM (AAL - *ATM Adaptation Layer*).

Este trabalho adotará a abordagem "IP over ATM" na implementação do protocolo IPv6 dada sua popularidade [40, 38]. O protocolo IPv6 será implementado em *hardware* utilizando-se a metodologia de projeto descrita no capítulo II. O protocolo IPv6 está em conformidade com as características dos protocolos de nova geração

- por exemplo, ter um cabeçalho básico fixo. Na próxima seção será apresentado o protocolo IPv6 e suas principais características. Na seção III.3 serão apresentados os principais conceitos da abordagem “IP over ATM”.

III.2 O Protocolo IPv6

O atual protocolo IP (IPv4) está se tornando inadequado para suportar os atuais requisitos de desempenho das novas aplicações. Entretanto, o principal motivo para a atualização do atual protocolo IP é o esgotamento iminente do seu espaçamento de endereços de 32 bits. Quando o IP foi desenvolvido, o espaço de 32 bits destinado ao endereçamento era mais do que suficiente. Algumas organizações usavam uma única LAN e um número ainda menor usava rede WAN. Atualmente, quase todas as corporações de médio porte possuem múltiplas LANs e a maioria das corporações de grande porte possui WANs. Conseqüentemente, o atual espaço de 32 bits para o endereçamento não é mais suficiente para acomodar este crescimento explosivo de redes.

Além da necessidade de um maior espaço de endereçamento, existem outros fatores que forçam mudanças no protocolo IP. Como exemplo, a nova versão do IP deve prover um mecanismo que torne possível associar um determinado datagrama à uma reserva de recurso pré-estabelecida. Tal mecanismo permitirá tráfegos de áudio e vídeo em tempo real, que necessitam de limitações máximas de atraso em suas apresentações.

Em resposta a todas estas novas necessidades, o grupo IETF publicou uma chamada a propostas para um protocolo IP de nova geração chamado IPng (*Next Generation Internet Protocol*), em julho de 1992. Um grande número de propostas foram recebidas e, em 1994, surgiu o modelo final para o IPng, com a publicação da RFC 1752 [12], publicado em janeiro de 1995. Outros documentos definem detalhes deste protocolo [41, 42, 43, 44, 45, 46, 47], agora oficialmente chamado de IPv6.

III.2.1 Principais Características do IPv6

As mudanças introduzidas para o IPv6 podem ser vistas, resumidamente, sob 3 aspectos:

- *endereços maiores.* A mudança mais perceptível é o novo tamanho destinado aos campos de endereçamento do datagrama. O protocolo IPv6 quadruplicou o tamanho do endereçamento do IPv4 de 32 bits para 128 bits;
- *cabeçalho de formato flexível.* O protocolo IPv6 adota um formato completamente novo para seu datagrama. Ao contrário do IPv4 que usa um cabeçalho único de tamanho variável, o protocolo IPv6 utiliza um cabeçalho básico de tamanho fixo seguido ou não de cabeçalhos opcionais;
- *suporte aos tráfegos multimídias.* O protocolo IPv4 oferece pouco suporte aos tráfegos multimídias. Já o IPv6 permite que os datagramas pertencentes a um mesmo fluxo sejam "rotulados", traduzindo desta maneira o tratamento especial que devem ter por parte dos roteadores intermediários;
- *provisão a extensões futuras.* O protocolo IPv6 é um protocolo que aceita características adicionais. Sua capacidade de extensão permite ao grupo IETF adaptá-lo a mudanças feitas nas redes físicas ou a novas aplicações.

O protocolo IPv6 mantém muitas das características do protocolo IP atual (IPv4). O protocolo IPv6 pode ser caracterizado como sendo basicamente o mesmo protocolo IPv4, só que com algumas modificações. Como exemplo, o protocolo IPv6 continua sendo um protocolo não orientado a conexão (tratando cada datagrama independentemente).

Como mencionado anteriormente neste capítulo, o formato do datagrama do protocolo IPv6 é mudado completamente. De acordo com a figura III.1, o datagrama IPv6 possui um *cabeçalho básico* de tamanho fixo seguido de zero, um ou mais *cabeçalhos de extensão*, seguido, por sua vez, pela área destinada aos dados.

O modelo de um cabeçalho básico seguido por uma série de cabeçalhos opcionais foi escolhido de modo a conciliar generalidade e eficiência. Para ser genérico,

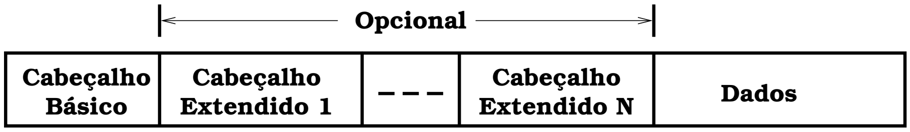


Figura III.1: Forma geral de um datagrama IPv6.

o protocolo IPv6 precisa incluir mecanismos que suportem funções tais como fragmentação, autenticação e roteamento. Entretanto, a escolha de alocar campos no cabeçalho básico do datagrama para todos os mecanismos é ineficiente pois a maioria dos datagramas não utiliza todos estes mecanismos.

Tal modelo comporta-se de maneira similar às opções existentes no IPv4. Uma estação de origem pode escolher quais cabeçalhos estendidos devem ser incluídos num dado datagrama e quais devem ser omitidos, fornecendo, assim, uma ampla flexibilidade. O padrão IPv6 definiu os seguintes cabeçalhos estendidos:

- `hop-by-hop options header`. Contém opções especiais que requerem processamento em cada nó existente no caminho tomado pelo datagrama. De um modo geral, cada opção existente neste cabeçalho consiste de um tipo, de um tamanho e de um valor;

- `routing header`. Normalmente, a estação origem de um datagrama IP deixa que a rede se encarregue de entregar o datagrama ao seu destino. Às vezes, entretanto, esta estação de origem deseja um controle maior na rota tomada pelo datagrama. A estação de origem pode desejar controlar o caminho a ser tomado para garantir que o datagrama, por exemplo, não irá passar por roteadores inapropriados. Este cabeçalho fornece à estação de origem este controle, similar à opção *source routing* do IPv4;

- `fragment header`. Contém informação de fragmentação e concatenação;

- `authentication header`. Este campo garante que um datagrama recebido é autenticado, ou seja, não foi alterado durante seu percurso e que realmente veio da estação de origem declarada. A autenticação faz parte das características de segurança do IPv6;

- `encapsulating security Payload header`. Este campo define o padrão adotado para enviar informação encriptografada dentro de datagramas IPv6;
- `destination options header`. Contém informação opcional a ser processada somente no nó destino.

O cabeçalho básico e os cabeçalhos estendidos possuem um campo em comum denominado *next header*. Este campo identifica o tipo do próximo cabeçalho a ser analisado. Se o próximo cabeçalho for um cabeçalho estendido, este campo conterá o identificador relativo àquele cabeçalho. Caso contrário, este campo conterá o identificador do protocolo utilizado acima do IPv6 (tipicamente um protocolo de transporte).

O cabeçalho básico do IPv6 possui um tamanho fixo de 40 octetos, consistindo dos seguintes campos:

- `version` (4 bits). Os primeiros quatro bits de cada datagrama IPv6 contém o número da versão do IP. Por se tratar do IPv6, todos os datagramas possuem o valor 6 neste campo;
- `priority` (4 bits). Os próximos quatro bits de cada datagrama definem sua prioridade. O campo de prioridade inicialmente diferencia dois tipos de tráfego: *tráfego com controle de fluxo* e *tráfego sem controle de fluxo*. Posteriormente, este campo refina as prioridades relativas a cada tipo de tráfego. Para tráfegos com controle de fluxo, o IPv6 define oito prioridades específicas, assumindo os valores de 0 (menor prioridade) a 7 (maior prioridade). Para o segundo tipo de tráfego, sem controle de fluxo, o protocolo IPv6 reserva os valores de prioridade de 8 (menor prioridade) a 15 (maior prioridade). Quanto mais baixa for a prioridade maior será a possibilidade do datagrama ser descartado. Deve ficar claro que não existe prioridade relativa a estes dois tipos de tráfego. Um datagrama de prioridade 8 (caracterizado assim como tráfego sem controle de fluxo) não possui prioridade nem maior nem menor do que um datagrama com prioridade 7 (caracterizado como tráfego com controle de fluxo);

- `flow label` (24 bits). Este campo pode ser usado por uma estação para "rotular" datagramas que terão tratamento especial por parte dos roteadores intermediários. Um *fluxo* é uma seqüência de datagramas identificado pela combinação do endereço de origem com o conteúdo do campo *flow label*. Todos os datagramas que fazem parte de um mesmo fluxo devem possuir o mesmo endereço de destino e o mesmo *flow label*, requerendo, todos, o mesmo tratamento da rede. Uma utilização efetiva de fluxos é usá-los como chaves dentro do cache do roteador. Normalmente, quando um datagrama chega em um roteador, este deve executar um determinado processamento de modo a determinar o próximo nó a ser seguido pelo datagrama. Se um datagrama contiver um *flow label*, entretanto, o roteador pode optar por consultar uma simples tabela em sua memória, associando o *flow label* a um destino e desta forma passar o datagrama adiante imediatamente, evitando-se o processamento tradicional. A utilização mais ambiciosa de *flow labels* suporta aplicações avançadas. Uma transmissão de vídeo é um bom exemplo de aplicação. O vídeo digital pode requerer uma quantidade substancial de recursos da rede, como banda passante, buffers e processamento. Para ter certeza de que os requisitos de transmissão do seu vídeo serão satisfeitos, a aplicação pode reservar estes recursos *a priori*. Assim, durante toda a transmissão do vídeo, os datagramas relativos a esta aplicação podem usar um *flow label* particular permitindo o uso dos recursos reservados. Se uma dada aplicação não suporta fluxos, seus datagramas devem ter o campo *flow label* "setados" em zero;

- `payload length` (16 bits). Tamanho em octetos do datagrama IPv6 após seu cabeçalho básico. Em outras palavras, define o tamanho total de todos os cabeçalhos estendidos mais a PDU de transporte;

- `next header` (8 bits). Este campo identifica o próximo cabeçalho em um datagrama IPv6. Ele pode indicar um cabeçalho estendido ou um protocolo acima do IPv6. A tabela III.1 lista os principais valores deste campo;

- `hop limit` (8 bits). Determina a vida útil de um datagrama, ou seja, define o quanto um datagrama irá trafegar. Quando uma estação cria um datagrama,

Tabela III.1: Valores estabelecidos para o campo *next header*.

0	Hop-by-Hop Options Header
4	Protocolo Internet
6	TCP
17	UDP
43	Routing Header
44	Fragment Header
45	Interdomain Routing Protocol
46	Resource Reservation Protocol
50	Encapsulating Security Payload
51	Authentication Header
58	Internet Control Message Protocol
59	No Next Header
60	Destination Options Header

é atribuído um valor arbitrário a este campo por esta estação. Este valor é decrementado de uma unidade em cada roteador existente no caminho tomado pelo datagrama. O datagrama é descartado se o valor deste campo assume o valor zero;

- `source address` (128 bits). Endereço de origem do datagrama;
- `destination address` (128 bits). Endereço de destino do datagrama. Pode não ser o endereço da estação de destino caso o cabeçalho de roteamento esteja presente.

A figura III.2 mostra o formato do cabeçalho básico. Vários campos do IPv6 correspondem diretamente a campos do cabeçalho do IPv4. Conforme o IPv4, os primeiros 4 bits do cabeçalho básico, relativos ao campo *version*, especificam a versão do protocolo. Os campos de endereçamento *source address* e *destination address* especificam os endereços IPv6 de origem e de destino do datagrama, exceto

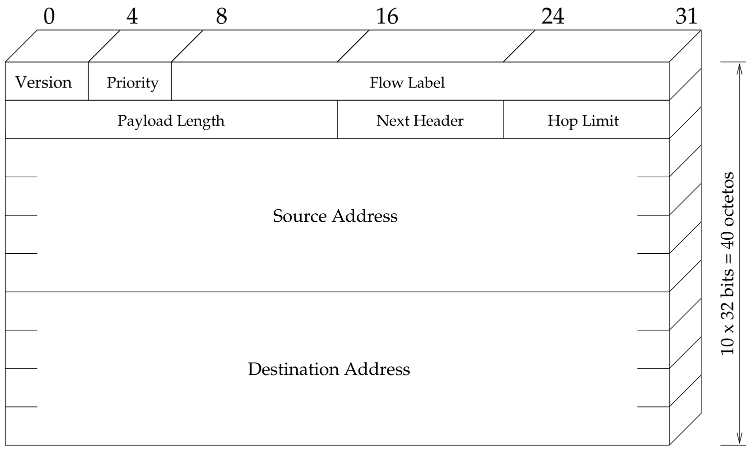


Figura III.2: O cabeçalho básico do IPv6.

que cada endereço requer 16 octetos. O campo *hop limit* corresponde ao campo do IPv4 *time to live*.

Quanto às especificações de tamanho do datagrama, o IPv6 o faz de uma maneira diferente do IPv4. O cabeçalho básico possui sempre 40 octetos, não necessitando, assim, que o cabeçalho básico possua um campo informando seu tamanho. Assim sendo, o IPv6 substitui o campo *total length* do IPv4 pelo campo de 16 bits *payload length* que especifica o número de octetos existentes no datagrama excluindo-se seu próprio cabeçalho básico. Um datagrama IPv6 pode conter até 64k octetos de dados.

Neste ponto, uma observação se faz importante. Apesar do cabeçalho básico IPv6 ser maior do que o cabeçalho do IPv4 (40 octetos contra 20 octetos, respectivamente), este cabeçalho contém menos campos (8 campos contra 12 do IPv4). Deste modo, roteadores perdem menos tempo processando cada cabeçalho básico, o que é refletido num aumento de velocidade durante o roteamento.

Em ambas as versões do protocolo IP, o nó de destino é responsável por concatenar todos os fragmentos de um datagrama. Entretanto, uma tática diferente foi tomada durante sua fragmentação. No IPv4, a responsabilidade de fragmentar qualquer datagrama que fosse maior do que o MTU (*Maximum Transmission Unit*) de uma determinada rede a qual o datagrama deveria trafegar é do roteador. No IPv6, a fragmentação é restrita à estação origem do datagrama. Antes de enviá-lo, a estação origem deve executar um algoritmo chamado de *Path Discovery MTU*, que determina o MTU mínimo existente ao longo do caminho a ser percorrido pelo datagrama. Assim, a fragmentação é realizada fim-a-fim e não mais em roteadores. O algoritmo de fragmentação aplicado ao protocolo IPv6 é o mesmo aplicado ao protocolo IPv4.

O cabeçalho básico do IPv6 não contém nenhum campo referente à fragmentação como os existentes no IPv4. Ao invés disso, quando a fragmentação se faz necessária, a estação origem do datagrama insere um cabeçalho estendido referente à fragmentação após o cabeçalho básico de cada fragmento. A figura III.3 mostra o conteúdo deste cabeçalho estendido. O campo *fragment offset* especifica a posição a qual pertence o primeiro octeto do fragmento atual em relação ao datagrama original. Seu valor é dado em unidades de 8 octetos. Assim, o IP deverá fragmentar seus datagramas em pedaços cujos tamanhos sejam múltiplos de 8 octetos.

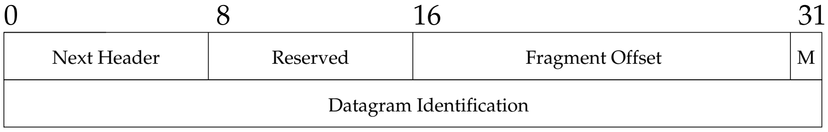


Figura III.3: O formato do cabeçalho estendido referente à fragmentação.

III.3 A Abordagem "IP over ATM"

Redes ATM (*Asynchronous Transfer Mode*) são, por natureza, orientadas a conexão. A transmissão de dados ocorre em canais virtuais estabelecidos previamente. A tecnologia ATM surge como uma possibilidade de aumento de desempenho e eficiência em redes locais, apesar da tecnologia ATM ter sido projetada originalmente para as redes WANs (*Wide Area Networks*). Desta maneira, aumenta-se o interesse em utilizar redes ATM para oferecer serviços sem conexão e possibilitar a construção e a interconexão de redes LANs. Grandes discussões vêm sendo realizadas sobre o local ideal onde o serviço sem conexão em redes ATM deve ser oferecido [48]: *internamente à rede*, o que corresponde a oferecer estes serviços diretamente na UNI (*User Network Interface*) pública, ou *externamente à rede*, o que corresponde a oferecer serviços sem conexão indiretamente, através de seu mapeamento em conexões

ATM existentes ou estabelecidas na rede. Essas duas formas básicas de fornecer serviços sem conexão em redes ATM correspondem às abordagens *direta* e *indireta*, respectivamente.

Com a abordagem indireta, é possível que protocolos de acesso ao meio (MAC) e protocolos do tipo IP sejam mapeados em conexões ATM, permitindo a interconexão transparente dessas redes. Este é o caminho que vem sendo adotado pelo ATM Forum e pelo grupo IETF responsável pelo RFC 1577 (“IP over ATM”) [16].

A maneira mais simples de realizar o serviço sem conexão é através de uma malha de conexões *semipermanentes* estabelecida entre cada par de estações da rede ATM. Estabelecer e liberar uma conexão para transmissão de cada pacote é ineficiente, pois impõe uma carga muito grande ao serviço de sinalização e um retardo de estabelecimento de conexão antes de cada transferência de pacote. Como a maioria das conversações consiste de múltiplos pacotes, se uma conexão é estabelecida por toda a duração da conversação, tanto a carga imposta ao serviço de sinalização quanto o retardo são minimizados. Nesta alternativa, supõem-se que futuros dados para aquele destino serão prováveis. Esta solução só é possível pois uma conexão virtual ATM não consome recursos quando inativa e um número grande de conexões virtuais inativas pode ser suportado pela rede.

Este trabalho empregará a abordagem “IP over ATM” por duas razões. Primeiramente, esta alternativa oferece maior eficiência ao evitar as complicações desnecessárias do nível de enlace, que é utilizada pela segunda abordagem, a emulação de LAN (LANE) [39]. Em segundo lugar porque o protocolo IPv6 será implementado diretamente acima da tecnologia ATM.

Na abordagem “IP over ATM” o mecanismo de resolução de endereços deve traduzir diretamente o endereço IP para o endereço ATM e vice-versa, fornecendo a ilusão da existência de sub-redes lógicas IP (LIS - *Logical IP Subnetworks*) [49, 50]. LIS são grupos de estações que possuem o mesmo endereço de sub-rede IP mas que estão fisicamente dispersos ao longo da rede ATM. Assim, cada LIS age como uma rede LAN.

As estações existentes numa LIS estabelecem canais virtuais entre si para enviar datagramas IP. Pelo fato de cada LIS ser, de maneira conceitual, uma rede LAN, o protocolo IP aplica as regras tradicionais a cada uma delas. Como exemplo, todas as estações de uma mesma LIS compartilham um único prefixo do endereço IP (*netid*) e este prefixo difere dos prefixos usados pelas outras sub-redes LIS. Além disso, uma estação de uma LIS não pode se comunicar diretamente com uma estação de uma outra LIS. Ao invés disso, toda a comunicação entre sub-redes LIS deve ser realizada através de um roteador pertencente a várias sub-redes LIS.

Duas AALs foram definidas para protocolos que trabalham com datagramas como é o caso do protocolo IPv6: AAL3/4 e AAL5. A AAL5 é, em geral, utilizada por melhor atender às características das redes locais: transferências de dados assíncronos, vazões variáveis, serviço não orientado a conexão e maior eficiência. A AAL5 pode suportar Unidade de Dados de Serviço (AAL_SDU) com tamanho de até 64k octetos. Entretanto, o impacto da transmissão de pacotes deste porte pode ser significativo quando os pacotes entram em enlaces com MTUs menores. Para atender este propósito, o *Networking Workgroup* produziu um documento, intitulado "Default IP MTU for use over ATM AAL5" [51], que define o valor de 9.180 octetos como a MTU para o protocolo IP.

Um outro documento, o RFC1483 [52], possui o intuito de desenvolver um padrão para carregar múltiplos protocolos, dos quais o protocolo IP é um deles. O IETF observa dois métodos para este objetivo. Quando a criação rápida e dinâmica de conexões virtuais é possível, cada protocolo deve escolher abrir uma conexão virutal para seu próprio uso, ou seja, realizando uma multiplexação baseada em VCs (*Virtual Circuits*). Quando as conexões virtuais possuem maiores dificuldades ou menor velocidade para serem criadas, é proposto um encapsulamento de forma a multiplexar diferentes protocolos em uma única conexão AAL5. A figura III.4 mostra este encapsulamento para datagramas IPv6 de modo a formar uma AAL_SDU, conhecido como encapsulamento LLC/SNAP. Os 3 primeiros octetos identificam o cabeçalho LLC (*Logical Link Control*). Este cabeçalho possui o valor 0xAA-AA-03 indicando a presença do cabeçalho SNAP (*SubNetwork Attachment Point*). O cabeçalho SNAP possui 5 octetos que irão indicar o protocolo, sendo que os 3 primeiros octetos especificam o fabricante e os dois octetos seguintes especificam o protocolo propriamente dito.

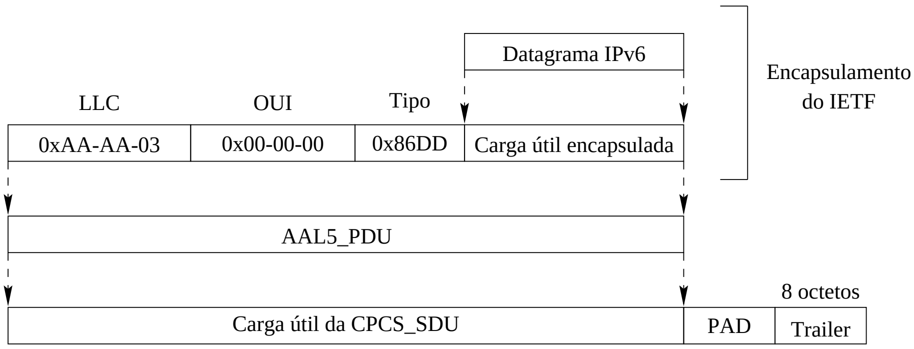


Figura III.4: Encapsulamento LLC/SNAP dos datagramas IPv6.

Tal encapsulamento se faz necessário pois o protocolo AAL5 não possui em seu *trailer* um campo que defina o tipo de protocolo que está sendo carregado (*type field*). No caso específico do protocolo IPv6, o campo que define o tipo do protocolo (*tipo* na figura III.4) possui o valor hexadecimal 0x86DD [53].

Analogamente às outras tecnologias de rede, a tecnologia ATM atribui para cada estação conectada um endereço físico que deve ser usado no estabelecimento de um canal virtual.

O gerenciamento de canais virtuais ocorre na interface de rede abaixo do protocolo IPv6. Quando uma estação (A) deseja enviar um datagrama para outra estação (B), o protocolo IPv6 em (A) executa o algoritmo de roteamento tradicional para determinar o endereço apropriado do próximo nó (N), que também é um endereço IPv6, para o qual o datagrama será enviado. De posse deste endereço, a estação (A) o passa juntamente com o datagrama IPv6 para a interface de rede, que examina sua tabela de canais virtuais abertos. Neste ponto, dois caminhos podem ser tomados pela interface de rede. Se existir um canal virtual para (N), este endereço é mapeado no par VPI/VCI do canal virtual e a estação utiliza o protocolo AAL5 para enviar o datagrama. Caso contrário, antes da estação (A) enviar o datagrama, esta estação deve exercer a sinalização ATM [54] para estabelecer o canal virtual necessário. Para tanto, a estação entra em contato com um servidor existente na sua LIS, chamado de "ATMARP Server", de modo a traduzir o endereço IPv6 de

(N) no seu respectivo endereço ATM. Após a criação do canal virtual, a estação (A) o adiciona à sua tabela e o envia para a estação destino.

Neste trabalho, será considerado que os canais virtuais utilizados para a transferência de informações entre duas estações já estão estabelecidos, ou seja, que o processo de sinalização ATM necessário para criá-los já foi realizado. Será enfocada principalmente a transferência de um datagrama IPv6 para o protocolo AAL5, utilizando-se o encapsulamento LLC/SNAP.

III.4 Conclusão

Neste capítulo foram apresentadas as principais características da nova versão do protocolo IP (IPv6) proposta pelo grupo IETF. Este protocolo estŕelacionado com as mudanças necessárias para se atingir os novos requisitos de desempenho impostos pelas novas aplicações, o que motivou sua implementação em *hardware*. Neste capítulo, também foi descrita a abordagem "IP over ATM", abordagem escolhida para a implementação do protocolo IPv6 sobre redes ATM. Esta abordagem é bastante promissora, uma vez que o protocolo IPv6 pode ser implementado diretamente sobre a tecnologia ATM.

O capítulo a seguir apresentará os passos seguidos, de acordo com a metodologia descrita no capítulo II e de acordo com a abordagem escolhida, na implementação em *hardware* do protocolo IPv6. Primeiramente serão apresentados os pontos principais da especificação em Estelle do protocolo, obedecendo certas restrições, apresentadas no capítulo II para se obter um mapeamento eficiente em VHDL. Em seguida, é mostrada parte da descrição comportamental em VHDL do protocolo. Tal descrição comportamental é obtida segundo o mapeamento Estelle-VHDL e o estilo de descrição adotado, descrito no capítulo II. Uma parte da descrição equivalente sem dependências, após a utilização do algoritmo DLS, também é apresentada neste capítulo.

Capítulo IV

A Implementação em Hardware do Protocolo IPv6

IV.1 Introdução

COM o surgimento de redes de alta velocidade e de aplicações multimídias, novos requerimentos de desempenho são impostos nas estações que compõem estas redes, refletindo-se em modificações no projeto de protocolos, que devem ser mais simples e mais eficientes. Entretanto, o alto desempenho de subsistemas de comunicação é obtido não apenas com a otimização em *software* dos protocolos mas também com sua implementação em *hardware*.

A implementação de protocolos de comunicação em *hardware*, porém, não é uma tarefa trivial. Circuitos VLSI são atualmente projetados de maneira semi-automática usando-se uma série de ferramentas integradas a um determinado ambiente de projeto. O procedimento de projeto normalmente tem como ponto de partida uma especificação do sistema em alto nível, denominada de descrição comportamental. No caso dos protocolos de comunicação, entretanto, os procedimentos de projeto tradicionais envolvem especificações do sistema em um nível de abstração maior que o normalmente empregado na descrição comportamental de circuitos. Este é o caso das linguagens de especificação derivadas das Técnicas de Descrição

Formal (FDTs - *Formal Description Techniques*) [5, 6, 7] tais como, por exemplo, a linguagem Estelle [8]. Como conseqüência, o projetista de protocolos quando confrontando com o problema de implementar uma dada especificação em *hardware*, deverá lidar com dois ambientes de projeto distintos: um envolvendo a especificação formal, verificação e teste do protocolo e o outro envolvendo a simulação, síntese e implementação física do circuito VLSI. Em geral, os dois ambientes são baseados em diferentes linguagens de descrição, o que requer uma interface de tradução de uma linguagem para outra.

Uma linguagem de descrição de *hardware* (HDL) tal como o VHDL (*Very high speed integrated circuits Hardware Description Language*) [9, 10] é necessária para a utilização de ambientes específicos de projeto de *hardware*, tais como as ferramentas de síntese de alto nível (HLS - *High Level Synthesis*).

Este trabalho tem como objetivo aplicar uma metodologia que traduz especificações Estelle em descrições VHDL com o intuito de implementar em *hardware* protocolos destinados a sistemas de comunicação de alta velocidade. Este trabalho aplica esta metodologia na nova versão do protocolo IP, oficialmente chamado de IPv6. Levando-se em consideração as futuras redes, este trabalho adota a abordagem "IP over ATM" para a implementação em *hardware* do protocolo IPv6.

Os resultados obtidos durante as diversas etapas da metodologia empregada neste trabalho para a implementação em *hardware* do protocolo IPv6 são mostrados nas seções subseqüentes. Na seção IV.3 são descritos os pontos principais da especificação Estelle do protocolo IPv6, onde são definidas suas primitivas de serviço e sua máquina de estados. Na seção IV.4 é mostrada a descrição comportamental em VHDL resultante do mapeamento Estelle-VHDL, enfocando o mecanismo de sincronização necessário para a comunicação entre entidades VHDL. A descrição sem dependências obtida através do algoritmo de *scheduling* DLS é apresentada na seção IV.5.1. Finalmente, na seção IV.6 é apresentado o resultado da síntese lógica pela ferramenta Synopsys aplicada à descrição sem dependências do protocolo.

IV.2 As Funções Básicas do Protocolo IPv6

De acordo com a RFC1122 [55], o protocolo IPv4 possui duas funções básicas: escolher o próximo nó para a emissão de datagramas (um processo chamado de *fowarding*) e concatenar datagramas IPv4 provenientes da rede. No caso do protocolo IPv6 as funções básicas são as mesmas, só diferindo no caso de fragmentação de datagramas, que deve ser realizada, se necessário, na estação emissora do datagrama. O protocolo IPv6 executa as seguintes funções para o envio de datagramas:

- seleciona o próximo nó (*next hop*) conectado à sua rede para o qual será enviado o datagrama;
- fragmenta o datagrama se necessário, inserindo o cabeçalho de fragmentação estendido;
- envia o datagrama para o *driver* apropriado da camada de enlace. No caso em questão, será considerado o protocolo AAL5 imediatamente abaixo do IPv6, de acordo com a abordagem "IP over ATM".

Na recepção dos datagramas, o protocolo IPv6 executa as seguintes funções:

- verifica se o datagrama está corretamente formatado;
- verifica se o datagrama é destinado à estação que o recebeu;
- concatena o datagrama se necessário;
- passa a mensagem encapsulada para o protocolo de transporte apropriado. Neste caso, é considerado apenas o protocolo TCP.

Neste trabalho, com o objetivo de fazer um primeiro estudo da complexidade do *hardware* gerado, foram tomadas as seguintes decisões:

- não será abordado o roteamento dos pacotes. Será considerado apenas o envio de datagramas entre estações pertencentes a uma mesma rede local. A rede local simulada será considerada uma sub-rede lógica (LIS) contendo 6 estações. O envio de datagramas para fora desta LIS não será considerado;

- será abordado apenas um sub-grupo das funções do protocolo IPv6. Não estão sendo abordadas, por exemplo, as interações deste protocolo com o protocolo ICMPv6 [45] por envolver outro protocolo, e as abordagens de segurança, características adicionadas ao protocolo IPv6;

- o protocolo IPv6 será implementado diretamente acima do protocolo AAL5, de acordo com a abordagem "IP over ATM". Desta maneira, por envolver conceitos da tecnologia ATM, será considerado que os canais virtuais utilizados para a transferência de informações entre duas estações já estão estabelecidos, ou seja, que o processo de sinalização ATM necessário para criá-los já foi realizado. Além disso, o mapeamento do endereço de um datagrama IPv6 no par VPI/VCI do canal virtual correspondente durante o envio do datagrama para o protocolo AAL5, não será abordado. Estas decisões serão tomadas durante a especificação em Estelle do protocolo IPv6 bem como na descrição comportamental em VHDL do protocolo IPv6 obtida após o mapeamento Estelle-VHDL;

- a especificação em Estelle bem como a descrição comportamental em VHDL do protocolo IPv6 envolve a transferência de datagramas completos e fragmentados entre duas estações, cada uma sendo simulada dentro de seu ambiente específico. Entretanto, ao se direcionar para a síntese lógica da descrição em VHDL do protocolo IPv6, este trabalho irá considerar apenas a transferência de datagramas completos IPv6 para o protocolo AAL5, como um primeiro estudo da complexidade do *hardware* gerado por esta descrição.

IV.3 A Especificação Estelle do Protocolo IPv6

A especificação Estelle do protocolo IPv6 envolve a simulação de uma sub-rede LIS constituída de 6 estações. Para efeito de simulação, foi designado um nome e um endereço IPv6 para cada estação desta sub-rede. A composição hierárquica da especificação Estelle do protocolo IPv6 em termos de módulos, pontos de interação e canais pode ser vista na figura IV.1.

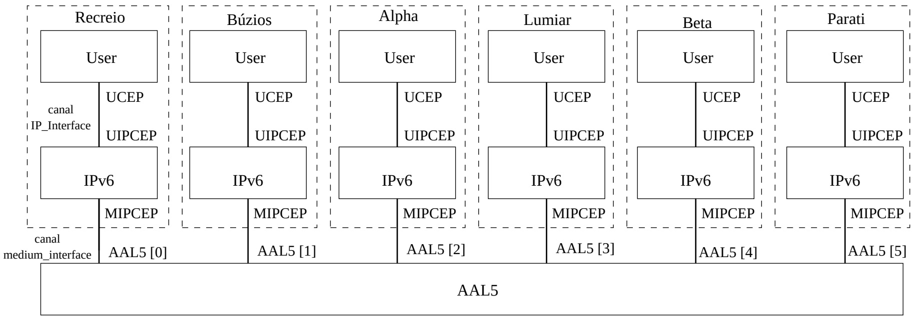


Figura IV.1: Composição hierárquica da especificação do protocolo IPv6.

Foram definidas 6 primitivas de serviço para a especificação Estelle do protocolo IPv6. Duas primitivas para a comunicação entre os módulos de usuário e os módulos do protocolo IPv6 e quatro primitivas para a comunicação entre os módulos do protocolo IPv6 e o módulo do meio de comunicação. As primitivas de serviço são apresentadas a seguir:

- **IP_Data_Req**: o módulo usuário aciona esta primitiva quando deseja transmitir dados (SDU) para o módulo IPv6;
- **IP_Data_Ind**: o módulo IPv6 aciona esta primitiva quando deseja transmitir dados, provenientes do meio de comunicação, para o módulo usuário;
- **AAL_Unitdata1_Req**: o módulo IPv6 aciona esta primitiva quando deseja transmitir um datagrama completo para o meio de comunicação;
- **AAL_Unitdata1_Ind**: o meio de comunicação aciona esta primitiva quando deseja enviar um datagrama completo para o módulo IPv6;
- **AAL_Unitdata2_Req**: o módulo IPv6 aciona esta primitiva quando deseja transmitir um fragmento de um datagrama para o meio de comunicação;
- **AAL_Unitdata2_Ind**: o meio de comunicação aciona esta primitiva quando deseja enviar um fragmento de um datagrama para o módulo IPv6.

A linguagem Estelle é baseada em máquina de estados. Entretanto, o protocolo IPv6 possui apenas um estado, relativo à fase de transferência de dados. Como o protocolo IPv6 é um protocolo não orientado a conexão (baseado em datagramas), não existe as fases de estabelecimento e fechamento de conexão. A máquina de estados do protocolo IPv6 é mostrada pela rede predicado-ação na figura IV.2.

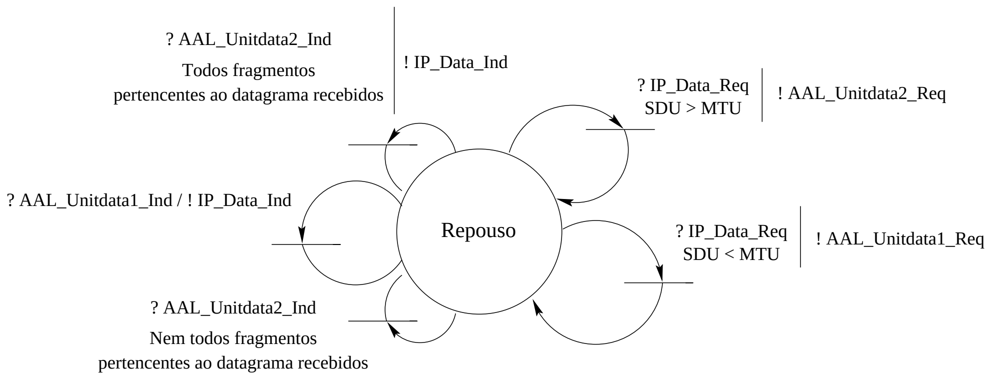


Figura IV.2: Rede predicado-ação do protocolo IPv6.

A figura IV.2 apresenta as ações tomadas por uma instância do módulo IPv6 na fase de transferência de dados. Ao acionamento da primitiva `IP_Data_Req`, a instância do módulo IPv6 verifica se o tamanho dos dados recebidos é maior ou menor do que o valor da MTU definida para o envio de datagramas em redes ATM (9.180 octetos). Se for maior, a instância do módulo IPv6 executa o algoritmo de fragmentação e aciona a primitiva `AAL_Unitdata2_Req` para o envio de cada fragmento pertencente ao datagrama para o meio de comunicação. Se for menor, a instância do módulo IPv6 aciona a primitiva `AAL_Unitdata1_Req` enviando o datagrama completo para o meio de comunicação.

A instância do módulo IPv6 recebe dados do meio de comunicação através do acionamento de duas primitivas: `AAL_Unitdata1_Ind` ou `AAL_Unitdata2_Ind`. Ao acionamento da primitiva `AAL_Unitdata1_Ind`, que representa a chegada de um datagrama completo, a instância do módulo IPv6 repassa diretamente os dados recebidos para a instância do módulo de usuário, através do acionamento da primitiva `IP_Data_Ind`. Ao acionamento da primitiva `AAL_Unitdata2_Ind`, que representa a chegada de um fragmento de um determinado datagrama, a instância do módulo

IPv6 verifica se todos os fragmentos pertencentes ao datagrama foram recebidos. Se afirmativo, a instância do módulo IPv6 concatena todos os fragmentos, de modo a restaurar o datagrama, e o repassa para a instância do módulo de usuário, através do acionamento da primitiva `IP_Data_Ind`. Caso negativo, a instância do módulo IPv6 espera recebê-los até que seja expirado o *timeout* para o recebimento de fragmentos. Se o *timeout* for expirado, a instância do módulo IPv6 descarta todos os fragmentos anteriormente recebidos e armazenados.

Cada primitiva de serviço possui um parâmetro representando a SDU a ser transmitida na comunicação entre módulos Estelle. No caso das primitivas de serviço acionadas entre os módulos IPv6 e o módulo do meio de comunicação, a SDU é na realidade um datagrama (ou um fragmento de um datagrama) que é composta por um cabeçalho e por uma parte referente aos dados.

Para tornar a simulação de envio e recepção de datagramas completos e fragmentados a mais próxima possível da realidade, este trabalho optou por representar a SDU como um *record*. Cada *record* possui campos relativos aos campos do cabeçalho do datagrama IPv6 e um campo de dados referente à parte de dados do datagrama. Na realidade, um campo de um *record* pode conter mais de um campo do cabeçalho de um datagrama IPv6, como conseqüência da concatenação dos campos deste cabeçalho. Isto foi feito de forma a tornar um campo de um *record* múltiplo de 16 bits, que é o tamanho definido para o tipo inteiro (*integer*) na linguagem modula2 utilizada durante uma especificação Estelle. Como exemplo, a figura IV.3 mostra o *record* definido na transmissão de um fragmento de um datagrama IPv6 para o meio de comunicação, através do acionamento da primitiva de serviço AAL_Unitdata2_Req.

Duas primitivas de serviço foram definidas no envio de datagramas IPv6 para o módulo do meio de comunicação: uma para o envio de datagramas completos (*AAL_Unitdata1_Req*) e outra para o envio de datagramas fragmentados (*AAL_Unitdata2_Req*). A definição dessas duas primitivas foram necessárias devido à adição de um cabeçalho estendido ao datagrama IPv6 referente à fragmentação do mesmo. Em ambas as primitivas estão sendo adicionados os campos relativos ao

```
IP_PDU_type2 = RECORD
      LLC : integer;
      LLCOUI1 : integer;
      OUI2 : integer;
      TIPO : integer;
      VerPriFlow : array [0..1] of integer;
      PayLength : integer;
      NextHop : integer;
      SourceAddr : array [0..7] of integer;
      DestAddr : array [0..7] of integer;
      Next2Res1 : integer;
      FragoffRes2M : integer;
      Id : array [0..1] of integer;
      dados : array [0..49] of char;
END;
```

Figura IV.3: Exemplo de um *record*.

encapsulamento LLC/SNAP. Entretanto, para a recepção de datagramas IPv6 provenientes do meio de comunicação, não há a necessidade da existência dos campos relativos ao encapsulamento LLC/SNAP. Mesmo assim, foram definidas duas primitivas para a recepção de datagramas IPv6 por parte dos módulos IPv6: uma para a recepção de datagramas completos (*AAL_Unitdata1_Ind*) e outra para a recepção de datagramas fragmentados (*AAL_Unitdata2_Ind*).

Um nome e um endereço IPv6 foi designado, como já mencionado, para cada estação pertencente à sub-rede LIS simulada neste trabalho. Entretanto, este trabalho levou em consideração a migração necessária de um endereço IPv4 para um endereço IPv6, pois apesar deste novo protocolo ser bem mais poderoso do que seu antecessor, nenhuma rede que se baseia na arquitetura Internet TCP/IP mudará rapidamente do IPv4 para o IPv6.

São definidos para o protocolo IPv6 vários endereços particulares que possuem um significado especial. Um desses endereços é utilizado para suportar a transição do endereço IPv4 [47] para o endereço IPv6. Este endereço é chamado de endereço *compatível ao IPv4* (*IPv4-compatible address*) que pode ser convertido de e para o antigo formato do endereço IPv4. Este endereço IPv6 é formado prefixando-se 96 bits iguais a zero ao endereço de 32 bits do IPv4. Vale lembrar que a notação para um endereço IPv6 não é mais a decimal, notação usada para representar um endereço IPv4, mas sim a hexadecimal. Os 128 bits são divididos em 8 partes de 16 bits separadas por dois pontos (:). Como exemplo, o endereço IPv4 classe C `146.164.53.71` pode ser convertido para o endereço IPv6 `::92A4:3547`.

IV.3.1 Resultados de Simulação

A especificação em Estelle do protocolo IPv6 foi completamente simulada e validada usando-se o ambiente C.A.D. Estelle para verificar a correção da especificação do protocolo. Com o auxílio deste ambiente foram realizadas várias simulações que incluíram a transferência de datagramas completos e fragmentados envolvendo duas estações. Também foram realizadas simulações em que uma mesma estação pode receber fragmentos, de maneira intercalada, de diferentes estações de origem. Neste ambiente de simulação, o meio de comunicação não foi considerado ideal, havendo portanto a possibilidade de perdas de fragmentos. Não está sendo considerada a entrega de fragmentos fora de ordem, uma vez que o meio de comunicação simulado é orientado a conexão. Com tais simulações ficaram validados:

- o envio e o recebimento de datagramas completos e fragmentados;
- o algoritmo de fragmentação;
- o procedimento de descarte realizado pelo protocolo IPv6 quando um ou mais fragmentos pertencentes a um dado datagrama são perdidos no meio de comunicação.

Dentre as simulações realizadas, algumas encontram-se no apêndice A e são denominadas de simulação `A`, `B`, `C` e `D`. Como exemplo, a figura IV.4 apresenta o resultado da simulação `B`, onde foi realizada a transferência de um datagrama fragmentado entre duas estações. Neste caso foram escolhidas as máquinas `Alpha` e `Recreio`. Não houve perda de fragmentos.

```
============================================================
     | ALPHA[ 0]                | INICIOTRANSaot       |     0
     |                          |----------------------|
     |  --> SYS[ 0]             | ENVIAR               |
============================================================
     | ALPHA[ 0]                | INICIOENVIAaov       |     0
     |                          |----------------------|
     |  --> UCEP[ 0]            | IPDATAREQ            |
============================================================
     | IPALPHA[ 0]              | FRAGanj              |     0
     |                          |----------------------|
     |  --> MIPCEP[ 0]          | AALUNITDATA2REQ      |
     |                          |----------------------|
     |  --> MIPCEP[ 0]          | AALUNITDATA2REQ      |
============================================================
     | MEIO[ 0]                 | AAL2ENVIANDOFRAGafv  |     0
     |                          |----------------------|
     |  --> AAL5[ 0]            | AALUNITDATA2IND      |
============================================================
     | IPRECREIO[ 0]            | RECFRAGanp           |     0
============================================================
     | MEIO[ 0]                 | AAL2ENVIANDOFRAGafv  |     0
     |                          |----------------------|
     |  --> AAL5[ 0]            | AALUNITDATA2IND      |
============================================================
     | IPRECREIO[ 0]            | RECFRAGanp           |     0
     |                          |----------------------|
     |  --> UIPCEP[ 0]          | IPDATAIND            |
============================================================
     | RECREIO[ 0]              | RECDATaox            |     0
```

Figura IV.4: Resultado da simulação B.

A primeira coluna da figura IV.4 descreve as instâncias dos módulos e os pontos de interação pertencentes à cada instância que são envolvidos durante a transferência de dados entre duas estações. Quando a instância de módulo `IPALPHA` recebe dados pelo seu ponto de interação `UCEP`, esta instância verifica se o tamanho dos dados excede o valor da MTU estabelecida. Como neste caso o tamanho dos dados é superior ao valor da MTU, a instância `IPALPHA` executa o algoritmo de fragmentação e envia pelo seu ponto de interação `MIPCEP` dois fragmentos IPv6 para a instância `MEIO`. Ao receber os dois fragmentos, a instância `IPRECREIO` concatena-os e passa os dados para a instância `RECREIO`. A segunda coluna da figura IV.4 descreve as primitivas de serviço associadas na transferência de dados entre as instâncias de módulos envolvidas.

IV.4 A Descrição Comportamental do Protocolo IPv6

A descrição comportamental em VHDL da especificação em Estelle do protocolo IPv6 é obtida após o mapeamento Estelle-VHDL, conforme mostrado no capítulo II. A figura IV.5 ilustra, em parte, a entidade do protocolo IPv6. Deve-se lembrar que a arquitetura do protocolo foi descrita de maneira comportamental permitindo, assim, que os projetistas de protocolos os descrevam de modo mais abstrato.

Um ponto importante a ser observado na figura IV.5 é sua interface (*entity*). Durante a explanação sobre o mapeamento Estelle-VHDL descrita no capítulo II, foi dito que cada primitiva de serviço numa especificação Estelle corresponde a um sinal de uma entidade VHDL. Entretanto, na especificação em Estelle do protocolo IPv6, cada primitiva de serviço possui como parâmetro um *record*, o que dificulta a tradução da primitiva de serviço em um único sinal. Como solução, este trabalho optou por desmembrar cada campo deste *record* em um sinal. No caso do datagrama IPv6, cada campo de seu cabeçalho foi traduzido como um sinal, tornando mais legível sua interpretação bem como mais eficiente a transferência dos dados, resultando em um processamento menos complexo e conseqüentemente mais rápido por parte da entidade receptora dos dados.

Para que a troca de informações entre entidades VHDL seja eficaz é necessário que haja uma sincronização entre as entidades, de modo que uma entidade não envie dados a uma outra entidade sem que esta última esteja preparada para recebê-los. Esta sincronização é estabelecida através de pontos de sincronismo (sinais de controle) existentes em ambas as entidades.

Para que uma entidade `A` envie uma mensagem a uma entidade `B`, é necessário que `A` envie um sinal a `B` informando a existência de transmissão de uma mensagem. Por exemplo, quando uma entidade IPv6 deseja enviar uma mensagem para uma entidade AAL5, a instrução `AAL5_Unitdata_Req_cod <= '1'` é usada. Entretanto, a execução da instrução de envio deste sinal não habilita a entidade emissora de enviar o sinal. É necessário que a entidade receptora esteja preparada para recebê-la.

Após o envio do sinal informando a existência de uma mensagem a ser transmitida, a entidade emissora suspende seu processo até receber um sinal de confirmação. A espera pelo sinal de recepção é exemplificada pela linha de código `WAIT UNTIL aal5preparado = '1'`. Vale lembrar que este sincronismo acontece em ambos os sentidos de comunicação das duas entidades envolvidas.

```
entity IPv6_entity_type is
        port (start: in bit;
              clk: in bit;
              AAL5_Unitdata_Req_cod: out bit;
              AAL5_Unitdata_Req_version: out bit_4;
              AAL5_Unitdata_Req_priority: out bit_4;
              ipv6habilitado: out bit;
              aal5preparado: in bit;
              IPv6_Data_Req_cod: in bit;
end IPv6_entity_type;
architecture IPv6_entity_body of IPv6_entity_type is
    begin
        process
            begin
                AAL5_Unitdata_Req_cod <= '0';
                WAIT UNTIL (start = '1' and clk = '1');
                MEF: LOOP
                IF (IPv6_Data_Req_cod = '1') THEN
                 LermsgUserIPv6;
                 AAL5_Unitdata_Req_cod <= '1';
                 WAIT UNTIL aal5preparado = '1';
                 EnviarmsgIPv6aal5;
                ELSIF (AAL5_Unitadata_Ind_cod = '1') THEN
                .
                .
                ENDIF;
                WAIT UNTIL (clk = '1');
                IPv6_Data_Req_cod <= '0';
                END LOOP MEF;
            end process;
end IPv6_entity_body;
```

Figura IV.5: Uma parte da descrição comportamental protocolo IPv6.

O mecanismo de sincronização descrito acima realiza-se a nível de mensagem.

No caso de protocolos, o envio de uma mensagem pode ser representado por uma primitiva de serviço. Uma primitiva de serviço pode ter vários parâmetros. Dentre eles, o parâmetro de dados (SDU) merece uma atenção especial, visto que este parâmetro é de tamanho variável. O sincronismo necessário para o envio e recepção de dados entre entidades de uma mesma estação, análogo ao sincronismo de mensagens, deve ser definido. Tal sincronismo pode ser exemplificado na figura IV.6 pelo procedimento `Lermsgaal5IPv6`, relativo ao envio de dados do protocolo AAL5 para o protocolo IPv6.

```
procedure Lermsgaal5IPv6 is
    begin
      i := 0;
      lendomsgaal5: WHILE (i < aal5_pdu) LOOP
      WAIT UNTIL (ready_validin2 = '1');
      buffer_aal5ipv6_pdu(i) := AAL5_Unitdata_Ind_dados;
      ready_ackin2 <= '1';
      WAIT UNTIL (ready_validin2 = '0');
      ready_ackin2 <= '0';
      i := i + 1;
      END LOOP lendomsgaal5;
    end Lermsgaal5IPv6;
```

Figura IV.6: O procedimento Lermsgaal5IPv6.

Este mecanismo de sincronização utilizado na troca de dados pode ser melhor visualizado na figura IV.7. Antes da entidade IPv6 receber o primeiro dado, esta habilita a entidade AAL5 através do sinal `ipv6habilitado` e suspende seu processo através do código `WAIT UNTIL ready_validin2 = '1'`. A entidade AAL5 então envia o primeiro dado, o sinal `varind_validout3` e suspende seu processo. Este sinal `varind_validout3` reabilita a entidade IPv6 a continuar a execução do processo, efetuando o tratamento do dado, enviando o sinal `ready_ackin2` e suspendendo novamente o seu processo. Este procedimento se repete até que o último dado seja enviado.

A visão global do diagrama em blocos dos protocolos IPv6 e AAL5, com os si-

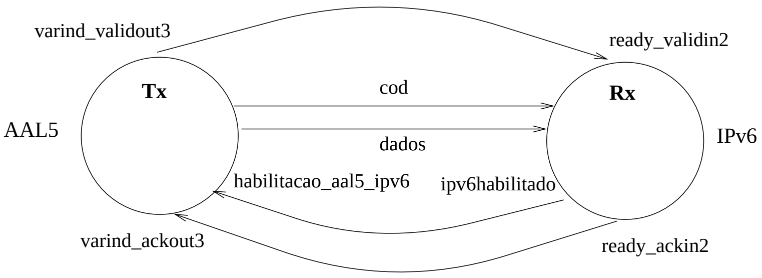


Figura IV.7: Diagrama simbólico do mecanismo de sincronização.

nais de controle necessários para efetuar a comunicação entre eles é ilustrada na figura IV.8. Nesta figura, a comunicação entre duas entidades do protocolo IPv6 é realizada através da entidade do protocolo AAL5, que possui dois conjuntos de sinais para cada sentido da comunicação. Quando a entidade IPv6_A deseja enviar dados à sua entidade par IPv6_B, a entidade AAL5 recebe os dados pelo conjunto de sinais da família `AAL5_Unitdata_Req_A` e os envia para a entidade IPv6_B pelos sinais da família `AAL5_Unitdata_Ind_B`. No sentido oposto da comunicação, ou seja, quando a entidade IPv6_B deseja enviar dados à sua entidade par IPv6_A, os dados são recebidos pela entidade AAL5 através do conjunto de sinais da família `AAL5_Unitdata_Req_B` e enviados à entidade IPv6_A através do conjunto de sinais da família `AAL5_Unitdata_Ind_A`.

A forma de comunicação entre duas entidades VHDL de uma mesma estação (neste caso entre as entidades IPv6_A e AAL5 e entre as entidades IPv6_B e AAL5) difere da comunicação entre dois módulos Estelle. Em Estelle, a comunicação entre dois módulos se faz através de um *canal* entre dois pontos de interação. Este canal permite que os dois módulos troquem mensagens em ambas as direções. Entretanto, em VHDL, a única forma de comunicação permitida entre entidades é feita através de sinais, que são unidirecionais. Por isso, foi necessária a criação dos quatro conjuntos de sinais na entidade AAL5, dois conjuntos para a comunicação com cada entidade IPv6 (`AAL5_Unitdata_Req_A` e `AAL5_Unitdata_Ind_A` na comunicação com a entidade IPv6_A e `AAL5_Unitdata_Req_B` e `AAL5_Unitdata_Ind_B` na comunicação com a entidade IPv6_B). Para possibilitar a simulação da comunicação entre as entidades IPv6 é necessária a intermediação da entidade AAL5, devido ao encapsulamento LLC/SNAP que deve ser obedecido pelo protocolo IPv6 no envio de dados ao protocolo AAL5.

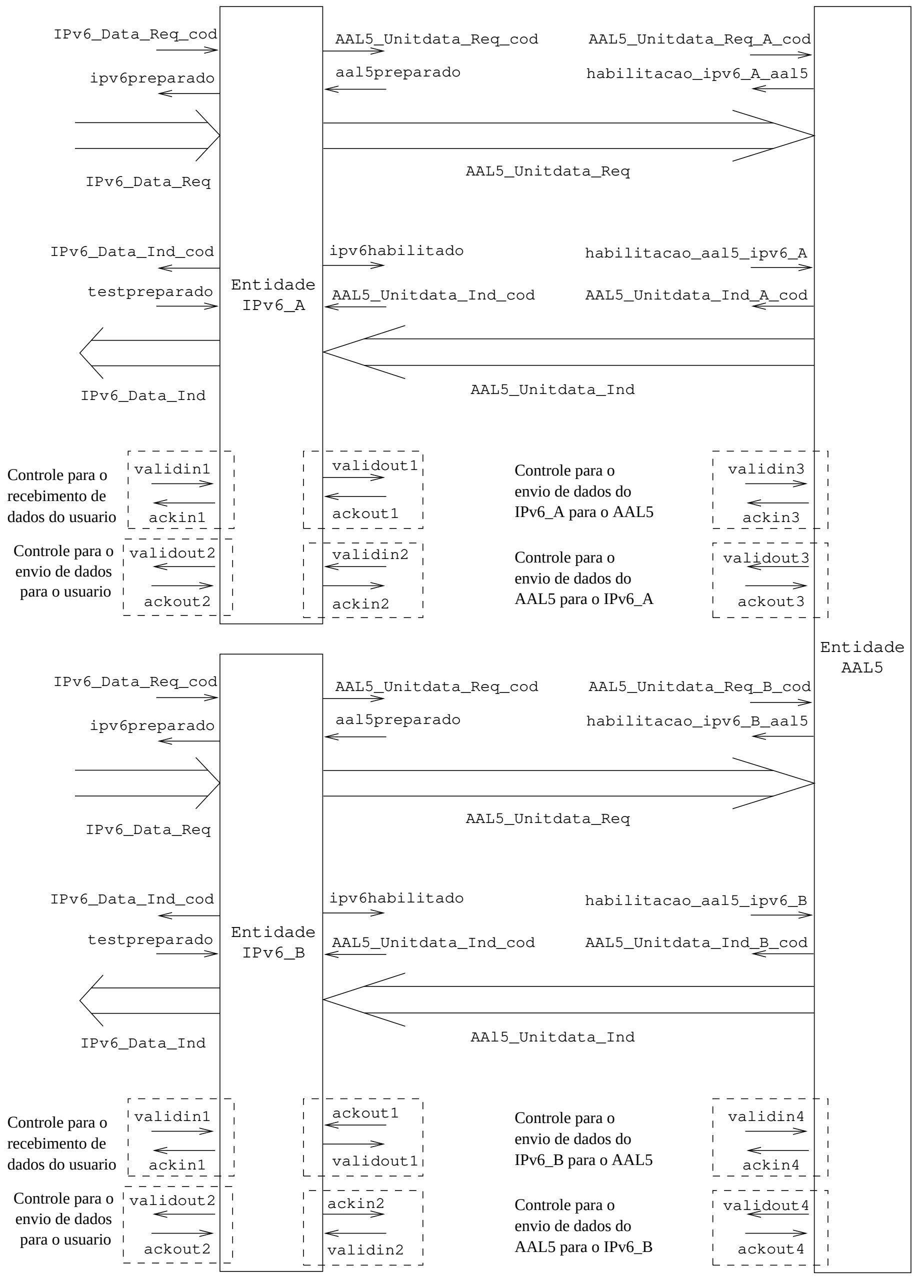


Figura IV.8: Sinais de controle trocados entre os protocolos IPv6 e AAL5.

Este encapsulamento se faz necessário somente quando uma entidade IPv6 deseja enviar dados à entidade AAL5. Tal encapsulamento não é repassado para a entidade receptora IPv6. Isto resulta em sinais adicionais relativos ao envio de datagramas que não possuem correspondência aos sinais de entrada da entidade receptora IPv6.

Por este motivo, foi necessária a inclusão da entidade AAL5 na comunicação entre duas entidades IPv6.

IV.4.1 Resultados de Simulação

Após o mapeamento da especificação em Estelle do protocolo IPv6 para uma descrição comportamental em VHDL foram realizadas simulações que incluíram troca de mensagens entre entidades IPv6. Tais simulações enfocaram principalmente a transferência de datagramas IPv6 através de uma entidade AAL5, obedecendo-se o encapsulamento LLC/SNAP, realizadas com o auxílio da ferramenta de simulação existente no Synopsys, o *vhdldbx*. Uma vez que o mapeamento Estelle/VHDL ainda é manual, as simulações excessivas envolvendo a ferramenta *vhdldbx* foi a alternativa encontrada neste trabalho para tentar garantir a consistência do mapeamento. Isto é facilitado pelo número pequeno de transições existentes na especificação do protocolo IPv6. De acordo com tais simulações e levando em consideração o número pequeno de transições existentes, pode-se dizer que a descrição comportamental em VHDL do protocolo está de acordo com a especificação funcional do protocolo e que o mapeamento Estelle-VHDL foi corretamente efetuado. A figura IV.9 apresenta a evolução dos sinais de controle pertencentes à descrição comportamental do protocolo IPv6 durante uma das simulações. A descrição comportamental completa em VHDL do protocolo IPv6 encontra-se no apêndice B.

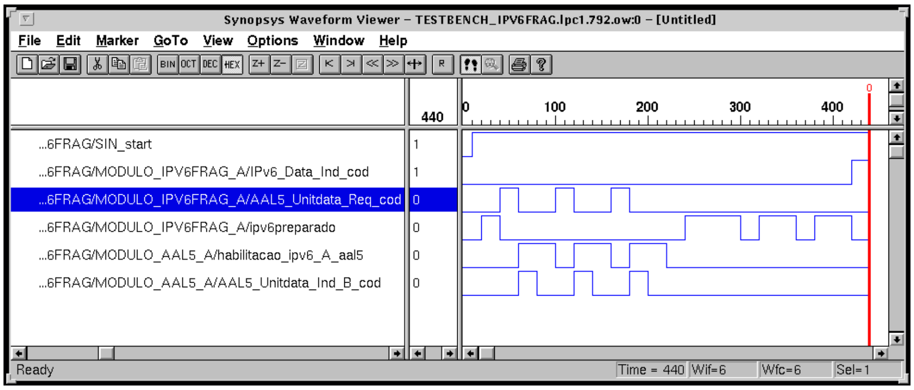


Figura IV.9: Evolução dos sinais de controle na simulação do protocolo IPv6.

IV.5 Obtenção de Descrições Sintetizáveis

A descrição comportamental do protocolo IPv6 corretamente simulada envolve a transferência de datagramas completos e fragmentados entre duas entidades pares (`IPv6_A` e `IPv6_B`). Entretanto, visando um primeiro estudo da complexidade de *hardware* gerado por esta descrição, este trabalho irá considerar apenas a transferência de datagramas completos entre as entidades pares durante os passos restantes da metodologia empregada, indo desde a obtenção de uma descrição sem dependências do protocolo IPv6 até a realização da síntese lógica desta descrição pela ferramenta Synopsys. Com este propósito, uma nova descrição comportamental em VHDL do protocolo IPv6 foi descrita, sem os procedimentos relativos ao algoritmo de fragmentação, sendo corretamente simulada com o auxílio da ferramenta de simulação *vhdldbx*.

Apesar do protocolo AAL5 suportar Unidades de Dados de Serviço (AAL_SDU) com tamanho de até 64k octetos, foi definido o valor de 9180 octetos como a MTU *default* para a emissão de datagramas IP em redes ATM. A fim de possibilitar a síntese lógica de uma descrição sem dependências do protocolo IPv6 que contivesse *buffers* com tamanhos de 9180 octetos, adotou-se a partição funcional do sistema mostrada na figura IV.10. A separação dos *buffers* do protocolo IPv6 em uma outra entidade (*entity*) requer a inclusão de novos sinais para a comunicação entre as entidades envolvidas, bem como novos mecanismos de sincronização, de forma a tornar a descrição comportamental do protocolo IPv6 funcionalmente equivalente à descrição usada na simulação. Vale lembrar que esta descrição usada na simulação envolve apenas a transferência de datagramas IPv6 completos. Esta descrição comportamental encontra-se no apêndice B.

O objetivo da partição funcional é evitar que a ferramenta de síntese lógica considere o *buffer* como uma máquina de estados. O Synopsys converte a descrição VHDL original em uma descrição lógica interna a ser otimizada. No caso de um *buffer* o número de variáveis torna-se impraticável.

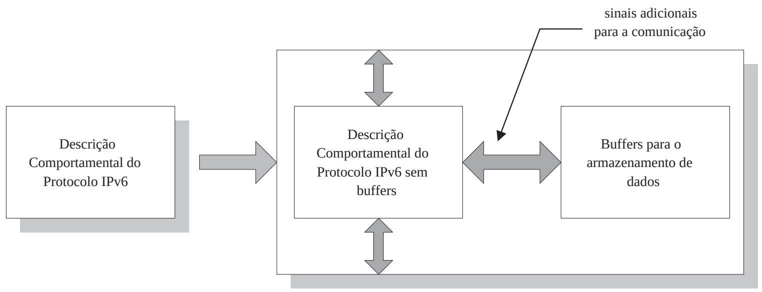


Figura IV.10: Divisão em duas entidades da descrição do protocolo IPv6.

IV.5.1 O DLS Aplicado no Protocolo IPv6

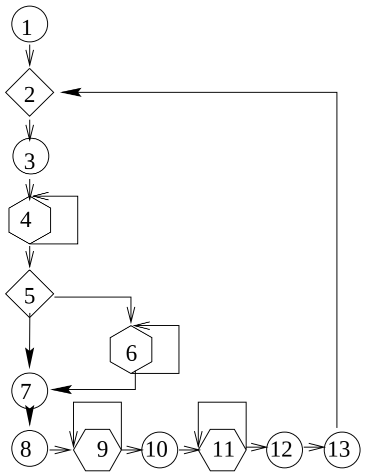
O algoritmo de *scheduling* DLS foi utilizado após a obtenção da descrição comportamental do protocolo IPv6 corretamente simulada envolvendo a transferência de datagramas IPv6 completos. O *scheduling* é o passo principal no processo de síntese de alto nível (HLS). A utilização deste algoritmo resultou em uma descrição sem dependências do protocolo passível de ser sintetizável pela ferramenta Synopsys [26], ferramenta adequada ao projeto de circuitos VLSI.

Conforme visto no capítulo II, a principal representação usada pelo DLS é um grafo de fluxo de controle (CFG - *Control-Flow Graph*). Este grafo fornece um meio conveniente de representar as propriedades inerentes de circuitos caracterizados por fluxo de controle, que são descritos por *loops* aninhados e por construções de sincronismo tais como a construção *wait*. Este CFG contém nós, representando as operações da descrição original, e ramos, indicando os fluxos de sinal da descrição original. Vale lembrar que a mais importante característica deste CFG é a introdução do nó *wait*, significando sempre uma mudança de estado que será efetuada. Isto causa a interrupção incondicional da geração de caminhos.

A tradução da descrição comportamental do protocolo IPv6 para seu CFG foi feita de acordo com o método descrito em [30]. Como exemplo para ilustrar este método, as figuras IV.11(a) e IV.11(b) mostram a representação em VHDL do procedimento `EnviarmsgIPv6aal5` e seu respectivo CFG. Este procedimento é executado no envio dos dados de uma entidade IPv6 para a entidade AAL5.

```
Procedure EnviarmsgIPv6aal5 is
begin
  i := 0; (1)
  enviandomsgaal5 : WHILE (i < sdu_usu) LOOP (2)
  ipv6_envia_aal5_mem_cod <= '1'; (3)
  WAIT UNTIL mem_autorizando = '1'; (4)
  IF (varind_ackout1 /= '0') THEN (5)
    WAIT UNTIL varind_ackout1 = '0'; (6)
  END IF;
  AAL5_Unitdata_Req_dados <= ipv6_envia_aal5_mem; (7)
  ipv6_envia_aal5_mem_cod <= '0'; (8)
  WAIT UNTIL mem_autorizando = '0'; (9)
  varind_validout1 <= '1'; (10)
  WAIT UNTIL (varind_ackout1 = '1'); (11)
  varind_validout1 <= '0'; (12)
  i := i + 1; (13)
  END LOOP enviandomsgaal5;
end EnviarmsgIPv6aal5;
```

(a)



(b)

Figura IV.11: Procedimento EnviarmsgIPv6aal5 (a) e seu respectivo CFG (b).

O procedimento `EnviarmsgIPv6aal5` contém treze operações, indicadas por números entre parênteses, quatro construções *wait*, duas construções de fluxo de controle (*if* e *while*) e sete expressões. Cada construção *wait* (os nós 4, 6, 9 e 11) contém um *loop* implícito representado por um fluxo que sai deste nó e entra novamente neste nó. Este fluxo é seguido quando a expressão booleana da construção *wait* for falsa.

Partindo-se da representação do CFG do protocolo IPv6, foram realizados os dois passos seguintes do algoritmo DLS para se chegar à descrição sem dependências do protocolo: a geração de caminhos e a geração de estados e transições. Durante a geração de caminhos, o primeiro caminho inicia-se com o primeiro nó do CFG. Os nós sucessivos são seqüencialmente adicionados ao caminho até que uma das condições mostradas anteriormente no capítulo II (C_1 e C_2) seja satisfeita. Na fase de geração de estados e transições, cada caminho gerado irá constituir uma transição. As transições em conjunto com os estados gerados irão produzir uma máquina finita de estados (FSM). Transições adicionais são geradas devido aos *loops* implícitos de todas as construções *wait*. Todos os caminhos iniciados com o mesmo nó serão colocados juntos no mesmo estado. Apesar da máquina de estados da descrição comportamental do protocolo IPv6 possuir apenas um único estado, com a utilização do algoritmo DLS este estado é "quebrado" em estados intermediários, adicionados devido às condições encontradas (C_1 e C_2) ao longo de seu CFG. A descrição comportamental do protocolo IPv6, envolvendo apenas a transferência de datagramas completos, resultou numa descrição sem dependências de dados e de controle, com 21 estados. A figura IV.12 ilustra parte da descrição sem dependências do protocolo IPv6.

```
entity IPv6_entity_sembuffer_type_sint is
        port (start: in bit; clk: in bit;
              AAL5_Unitdata_Req_cod: out bit;
              AAL5_Unitdata_Req_version: out bit_4;
              AAL5_Unitdata_Req_priority: out bit_4;
              IPv6_Data_Req_cod: in bit;
end IPv6_entity_sembuffer_type_sint;
architecture IPv6_entity_sembuffer_body_sint of IPv6_entity_sembuffer_type_sint is
    begin
        process
            begin
                MEF: LOOP
                WAIT UNTIL clk'event AND clk = '1';
                CASE state IS
                WHEN 12 =>
                    IF (ready_validin2 /= '0') THEN
                        state <= S12;
                    ELSE
                        ipv6_recebe_aal5_mem_cod <= '0';
                        ready_ackin2 <= '0';
                        i := i + 1;
                        state <= S6;
                    END IF;
                END CASE;
                END LOOP MEF;
        END process;
end IPv6_entity_sembuffer_body_sint;
```

Figura IV.12: Uma parte da descrição sem dependências do protocolo IPv6.

IV.5.2 Resultados de Simulação

Após a obtenção da descrição sem dependências do protocolo IPv6, foram realizadas as mesmas simulações, envolvendo troca de mensagens entre as entidades IPv6 e AAL5, que foram realizadas para a descrição comportamental em VHDL do protocolo IPv6. Novamente utilizou-se a ferramenta *vhdldbx* com o intuito de se verificar, agora, a validade dessa descrição VHDL sem dependências. A descrição completa, sem dependências, do protocolo IPv6 é encontrada no apêndice C.

IV.6 A Síntese Lógica do Protocolo IPv6

A descrição sem dependências do protocolo IPv6 foi simulada com a mesma seqüência de troca de mensagens das descrições comportamentais, ou seja, troca de mensagens entre as entidades IPv6 e AAL5. Conseqüentemente, a descrição sem dependências foi validada. Após o término da validação da descrição sem dependências do protocolo IPv6 é realizada a síntese lógica.

A figura IV.13(a) apresenta o esquema lógico gerado pelo Synopsys da descrição sem dependências do protocolo IPv6 sem a existência dos *buffers*. A figura IV.13(b) detalha uma pequena parte do esquema lógico gerado. Para a descrição sem dependências do protocolo IPv6 envolvendo apenas a transferência de datagramas completos, obteve-se uma estimativa de área de 12,76 mm^2. Este valor foi obtido utilizando uma biblioteca de células padrão (*standard cells*) ES2 na tecnologia 0.7 μm. O gasto considerável de área é característico do projeto em *standard cells*, não só pelo tamanho das células, que são muito grandes, como pela estrutura em fila de células e canais de roteamento. Esta área pode ser reduzida em até 40% utilizando-se um projeto dedicado (*full-custom*).

Conforme explicado na seção II.4.4, as atribuições condicionais de sinal e instruções de atribuição de sinal são convertidas em equivalentes booleanos, dentre os quais *Flip-Flops* e *latches* podem ser instanciados ou inferidos. Dependendo do tipo de cada sinal o Synopsys infere um *Flip-Flop* ou um registro. Como exemplo,

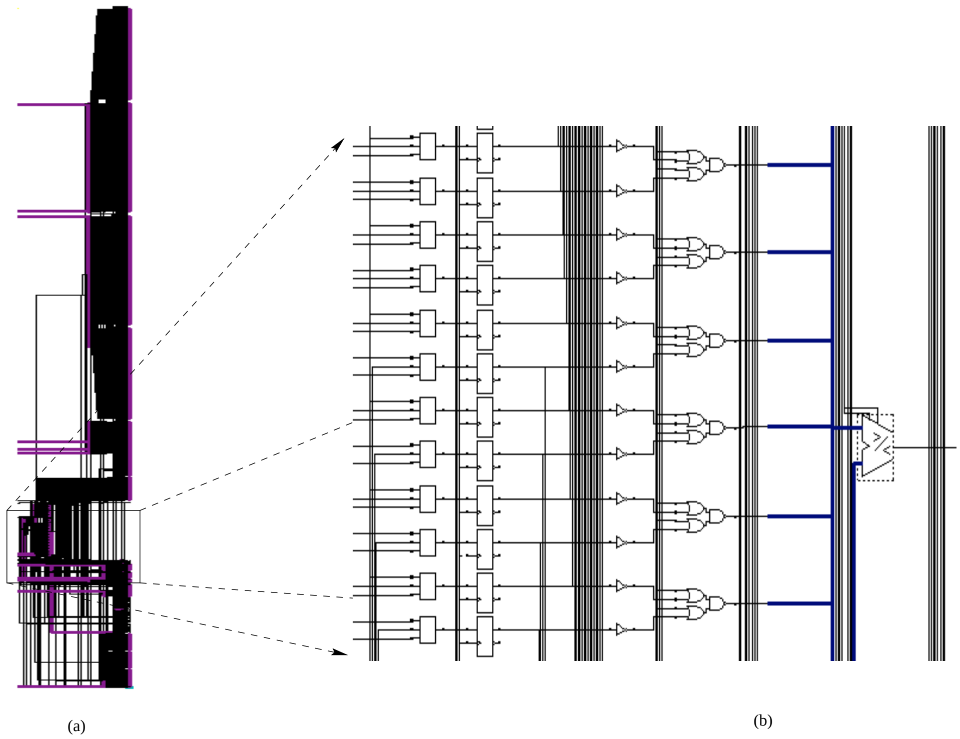


Figura IV.13: Esquema lógico do protocolo IPv6(a) e um detalhe do circuito sintetizado(b).

para o sinal `ipv6_recebe_aal5_mem` foi inferido um registro de 8 *Flip-Flops*, uma vez que este sinal possui o tipo *byte*. A tabela IV.1 mostra os *Flip-Flops* inferidos pelo Synopsys para cada atribuição de sinal e para cada variável pertencente à descrição sem dependências do protocolo IPv6. O circuito lógico resultante possui 3375 células, dentre as quais 1220 são *Flip-Flops*.

IV.7 Conclusão

Os resultados obtidos nas diversas etapas da metodologia aplicada ao protocolo IPv6 foram apresentados neste capítulo. A especificação em Estelle deste protocolo foi completamente simulada e validada usando o ambiente C.A.D. Estelle, que consiste de um conjunto de ferramentas desenvolvidas pelo GTA/COPPE/UFRJ.

Após o mapeamento da especificação em Estelle do protocolo IPv6 numa des-

Tabela IV.1: Registros inferidos na entidade IPv6.

Nome do Registro	Tipo	Quant.
AAL5_Unitdata_Req_aal5type_LLC_reg	Flip-Flop	24
AAL5_Unitdata_Req_aal5type_OUI1_reg	Flip-Flop	8
AAL5_Unitdata_Req_aal5type_OUI2_reg	Flip-Flop	16
AAL5_Unitdata_Req_aal5type_TIPO_reg	Flip-Flop	16
AAL5_Unitdata_Req_cod_reg	Flip-Flop	1
AAL5_Unitdata_Req_dados_reg	Flip-Flop	8
AAL5_Unitdata_Req_destaddr_reg	Flip-Flop	128
AAL5_Unitdata_Req_flowlabel_reg	Flip-Flop	24
AAL5_Unitdata_Req_hoplimit_reg	Flip-Flop	8
AAL5_Unitdata_Req_nextheader_reg	Flip-Flop	8
AAL5_Unitdata_Req_payload_length_reg	Flip-Flop	32
AAL5_Unitdata_Req_priority_reg	Flip-Flop	4
AAL5_Unitdata_Req_sourceaddr_reg	Flip-Flop	128
AAL5_Unitdata_Req_tam_reg	Flip-Flop	32
AAL5_Unitdata_Req_version_reg	Flip-Flop	4
IPv6_Data_Ind_cod_reg	Flip-Flop	1
IPv6_Data_Ind_ipdest_reg	Flip-Flop	128
IPv6_Data_Ind_ipsrc_reg	Flip-Flop	128
IPv6_Data_Ind_sdu_reg	Flip-Flop	8
aal5_pdu_reg	Flip-Flop	32
buffer_LLC_reg	Flip-Flop	24
buffer_OUI1_reg	Flip-Flop	8
buffer_OUI2_reg	Flip-Flop	16
buffer_TIPO_reg	Flip-Flop	16
buffer_destaddr_reg	Flip-Flop	128
buffer_flowlabel_reg	Flip-Flop	24
buffer_hoplimit_reg	Flip-Flop	8
buffer_priority_reg	Flip-Flop	4
buffer_sourceaddr_reg	Flip-Flop	128
buffer_version_reg	Flip-Flop	4
i_reg	Flip-Flop	32
ipv6_envia_aal5_mem_cod_reg	Flip-Flop	1
ipv6_envia_usu_mem_cod_reg	Flip-Flop	1
ipv6_recebe_aal5_mem_cod_reg	Flip-Flop	1
ipv6_recebe_aal5_mem_reg	Flip-Flop	8
ipv6habilitado_reg	Flip-Flop	1
ipv6preparado_reg	Flip-Flop	1
ready_ackin1_reg	Flip-Flop	1
ready_ackin2_reg	Flip-Flop	1
sdu_usu_reg	Flip-Flop	32
state_reg	Flip-Flop	32
usu_envia_mem_cod_reg	Flip-Flop	1
usu_envia_mem_reg	Flip-Flop	8
varind_validout1_reg	Flip-Flop	1
varind_validout2_reg	Flip-Flop	1

crição VHDL, foram feitas simulações num ambiente de C.A.D. de projeto de circuitos que incluíram troca de mensagens entre entidades, mostrando que o protocolo está de acordo com sua respectiva especificação funcional e que seu mapeamento foi corretamente efetuado.

Em seguida, utilizou-se o algoritmo de *scheduling* DLS a fim de se obter uma descrição sem dependências, de dados e de controle, de modo a ser passível de ser sintetizável logicamente pela ferramenta *design_analyzer* existente no Synopsys. Vale lembrar que, após a utilização deste algoritmo no protocolo IPv6 não se obteve uma descrição estrutural, mas sim uma descrição ainda comportamental sem dependências, visto que a descrição comportamental do protocolo IPv6 não possui subsistemas (componentes). Esta descrição comportamental resultou numa máquina com 21 estados. Antes de se realizar a síntese lógica do protocolo IPv6, sua descrição sem dependências foi simulada com a mesma seqüência de troca de mensagens das descrições comportamentais, ou seja, troca de mensagens entre as entidades IPv6 e AAL5. Assim, verificou-se a validade da utilização do algoritmo.

Durante a síntese lógica da descrição sem dependências do protocolo IPv6 envolvendo apenas a transferência de datagramas completos, adotou-se a partição funcional do sistema mostrada na figura IV.10, com o objetivo de evitar que a ferramenta de síntese lógica considere o *buffer* como uma máquina de estados. O Synopsys converte a descrição VHDL original em uma descrição lógica interna a ser otimizada.

O próximo capítulo apresentará as considerações finais sobre este trabalho.

Capítulo V

Conclusões

COM o surgimento de redes de alta velocidade e de aplicações multimídias, novos requisitos de desempenho são impostos aos sistemas de processamento de dados, refletindo-se em modificações no projeto de protocolos, que devem ser mais simples e mais eficientes de modo a permitir operações de alta velocidade e suportar os tráfegos multimídias. Entretanto, o alto desempenho de subsistemas de comunicação é obtido não apenas com a otimização em *software* dos protocolos mas também com sua implementação em *hardware*.

Neste trabalho é apresentada uma metodologia para a implementação em *hardware*, de maneira semi-automática, de protocolos destinados a sistemas de comunicação de alta velocidade. A metodologia é baseada no uso do mapeamento Estelle/VHDL proposto por Pirmez [11] e no uso da ferramenta de síntese lógica Synopsys [26].

Esta metodologia foi aplicada na nova versão do protocolo IP, oficialmente chamado de IPv6 dentro da abordagem "IP over ATM". Tal escolha foi estimulada pelo fato deste protocolo estar relacionado com as mudanças que se fazem necessárias para se atingir os novos requisitos de desempenho impostos pelas novas aplicações.

Atualmente, as redes locais (LANs) começam a se mostrar limitadas em termos de desempenho devido ao crescente número de estações interconectadas e ao surgimento de novas aplicações que exigem grande banda passante. A tecnologia ATM é capaz de fornecer uma alta banda passante e de suportar, de maneira integrada, diferentes tipos de Tráfego (voz, vídeo e dados). A simples implantação de novas LANs baseadas na tecnologia ATM mostra-se, no entanto, inviável dada a enorme base instalada. Como conseqüência, é necessário uma migração suave para as redes ATM, com soluções intermediárias, que permitam que as aplicações existentes possam continuar sendo utilizadas. A abordagem "IP over ATM" é um dos caminhos na implementação da emulação de LANs sobre redes ATM, possibitando então, um aumento no desempenho e na eficiência das redes locais. Esta abordagem é bastante promissora, uma vez que o protocolo IPv6 pode ser implementado diretamente sobre a tecnologia ATM.

Com este trabalho, mostrou-se que é possível a implementação em *hardware* de um protocolo de comunicação a partir de uma descrição formal de alto nível. Como um primeiro estudo, este trabalho optou por especificar em Estelle apenas um subgrupo do protocolo IPv6, de modo a se analisar a complexidade do *hardware* gerado ao final da metodologia empregada.

Os resultados obtidos mostraram que a maior limitação envolvida no projeto é o tamanho dos *buffers* para o armazenamento de dados que no caso do protocolo IPv6 chega a 9180 octetos. Para isto, foi adotada uma partição funcional do projeto onde a memória é implementada em separado da parte de controle. A síntese lógica do controle é efetuada pelo Synopsys mostrando que a metodologia proposta pode ser implementada sem o recurso a ferramentas de síntese de alto nível. Isto se deve ao fato de que protocolos podem em geral ser modelados de forma eficiente por máquinas de estados finitas, dispensando a descrição RTL na síntese lógica.

V.1 Trabalhos Futuros

Este trabalho direcionou para a implementação em *hardware* uma pequena parte do protocolo IPv6. Após a aplicação da metodologia empregada, este trabalho sugere os seguintes itens como trabalhos futuros:

- automatizar, através da criação de uma ferramenta, o processo de mapeamento Estelle/VHDL de modo a garantir a consistência, efetivamente, do mapeamento. Uma vez que o mapeamento ainda é manual, este trabalho tentou garantir a consistência do mapeamento através de simulações excessivas envolvendo a ferramenta de simulação do Synopsys (*vhdldbx*), facilitado pelo número pequeno de transições existentes na especificação do protocolo IPv6;

- utilizar a ferramenta de síntese de alto nível pertencente ao Synopsys, o *behaviour_compiler*), na tradução da descrição comportamental do protocolo IPv6 na sua descrição estrutural equivalente. O uso desta ferramenta tem a vantagem de envolver todos os passos da síntese de alto nível, indo desde o macro_*scheduling* até a geração da arquitetura. A não-disponibilidade desta ferramenta ao longo do desenvolvimento deste trabalho e a própria natureza do sistema sintetizado, possuindo um mínimo de paralelismo, resultaram na escolha de se empregar, somente, o algoritmo de *scheduling* DLS como etapa fundamental da síntese de alto nível;

- Dentre as funções básicas do protocolo IPv6 não foi considerado o algoritmo de roteamento de datagramas. Este trabalho sugere especificar formalmente não só o procedimento realizado pelo protocolo IPv6 para rotear datagramas como as interações deste protocolo com o protocolo ICMPv6, visando a implementação em *hardware*. Não se deve esquecer que a tecnologia TCP/IP continua evasiva. Diferente do modelo OSI - que descreve formalmente os serviços de cada camada, a interface usada pelas camadas adjacentes para troca de informações e o protocolo que define regras de comunicação para cada uma das camadas - os documentos que descrevem a tecnologia TCP/IP continuam a analisar seus protocolos independentemente, sem considerar como múltiplos protocolos operam em conjunto. A maior dificuldade no entendimento dos protocolos reside nas interações entre eles;

- realizar a síntese lógica da descrição comportamental em VHDL do protocolo IPv6 que envolve o algoritmo de fragmentação de datagramas. Neste trabalho foi incluído este algoritmo de fragmentação numa descrição comportamental em VHDL. Porém foi realizada apenas sua simulação;

- realizar a geração de *layout*, através da ferramenta Opus [27], do circuito lógico sintetizado neste trabalho. Com este último passo pertencente à metodologia empregada, obtém-se com uma maior precisão o tamanho da área do circuito VLSI bem como o desempenho obtido por este circuito, de modo a analisar o ganho de velocidade com relação à implementação do protocolo IPv6 em *hardware*.

Referências Bibliográficas

[1] Zitterbart, M., "Parallel Protocols Implementations for High Speed Networks", *SBT/IEEE International Telecommunications Symposium*, pp. 11.5.1–11.5.5, Set 1990.

[2] Duarte, O. C. B., Baginski, L. F., "Um modelo de implementação de alto desempenho para protocolos de comunicação estruturados em camadas", *RBC - Revista Brasileira de Computação*, vol. 6, no. 4, Abr 1991.

[3] Huang, Y., Guan, S., "A Refined Cut-Through Buffer Management Scheme for Layered Protocols Stack", *IEEE Communications Magazine*, Mar 1994.

[4] Krishnakumar, A. S., Sabnani, K., "VLSI Implementations of Communication Protocols - A Survey", *IEEE journal on Selected Areas in Communications*, vol. 7, no. 7, Set 1989.

[5] King, P., "Formalization of Protocol Engineering Concepts", *IEEE transactions on Computers*, vol. 40, no. 4, pp. 387–403, Abr 1991.

[6] Vissers, C. et al., "Formal Description Techniques", *Procceding of IEEE*, vol. 71, no. 12, pp. 1356–1364, Dez 1983.

[7] Bochmann, G., Sunshine, C., "Formal Methods in Communication Protocol Design", *IEEE transactions on Communications*, vol. 28, no. 4, pp. 624–631, Abr 1980.

[8] Budkowski, S., Dembinski, P., "An Introduction to Estelle: A Specification Language for Distributed Systems", *Computer Network and ISDN System, North-Holland*, vol. 14, pp. 3–23, 1987.

[9] IEEE Std 1076-1987, *IEEE Standard VHDL Language Reference Manual*, março 1988.

[10] Ashenden, P. J., *The VHDL Cookbook.* Computer Science Dept. of University of Adelaide, 1990.

[11] Pirmez, L., *Uma Metodologia para a Implementação de Sistemas Distribuídos em Hardware a partir de uma Descrição Formal.* Tese de Doutorado, Universidade Federal do Rio de Janeiro, PEE/COPPE/UFRJ, Rio de Janeiro, Brasil, 1996.

[12] Bradner, S., Mankin, A., *The Recommendation for the IP Next Generation Protocol.* RFC 1752, Harvard University, ISI, janeiro 1995.

[13] Deering, S., Hinden, R., *Internet Protocol Version 6 (IPv6) Specification.* RFC 1883, Xerox PARC, Ipsilon Networks, dezembro 1995.

[14] Stallings, W., "IPv6: The New Internet Protocol", *IEEE Communications Magazine*, Jul 1996.

[15] Le Boudec, J-Y., "The Asynchronous Transfer Mode: a tutorial", *Computer Network and ISDN System, North-Holland*, vol. 24, pp. 279–309, 1992.

[16] Laubach, M., *Classical IP and ARP over ATM.* RFC 1577, Hewlett-Packard Laboratories, janeiro 1994.

[17] L. Pirmez, M. Rahmouni, P. Kission, A. Pedroza, A. C. Mesquita e A. A. Jerraya, "Analysis of different protocol description styles in vhdl to high-level synthesis", in *EURODAC'96*, Geneve, 1996.

[18] Carli, E., Lima, R. N. B., Pedroza, A.C.P. et al., "Implementação em Hardware de Protocolos para Redes de Alta Velocidade a partir de uma Descrição Formal", in *II Simpósio Franco-Brasileiro de Sistemas Informáticos Distribuídos, SFBSID'97*, Fortaleza, CE, pp. 211–220, Nov 1997.

[19] Kloos, C. D. et al., "VHDL Generation from a timed extension of the formal description technique LOTOS within the FORMAT project", *Microprocessing and Microprograming, North-Holland*, vol. 38, pp. 589–596, 1993.

[20] Wytrebowicz, J., "Hardware Specification Generated from Estelle", in *Fifteenth International Symposium on Protocol Specification, Testing and Verification, IFIP WG 6.1*, Varsow, Polônia, 1995.

[21] L. Pirmez, A. Pedroza e A. C. Mesquita, "A methodology to the implementation of distributed system in hardware from a formal description", in *Fifteenth International Symposium on Protocol Specification, Testing and Verification*, Varsow, Poland, IFIP WG 6.1, 1995.

[22] Cruz, R., "*Um Compilador para a Linguagem Estelle*", Tese de Mestrado, Universidade Federal do Rio de Janeiro, PEE/COPPE/UFRJ, Rio de Janeiro, Brasil, 1991.

[23] Bernardo Filho, O., "*Verificador para Estelle*", Tese de Mestrado, Universidade Federal do Rio de Janeiro, PEE/COPPE/UFRJ, Rio de Janeiro, Brasil, 1992.

[24] Soares da Silva, G., "*Geração Automática de Sequências de Testes para Protocolos de Comunicação*", Tese de Mestrado, Universidade Federal do Rio de Janeiro, PEE/COPPE/UFRJ, Rio de Janeiro, Brasil, 1992.

[25] Resende Loureiro, F., "*Um Simulador para a Linguagem Estelle*", Tese de Mestrado, Universidade Federal do Rio de Janeiro, PEE/COPPE/UFRJ, Rio de Janeiro, Brasil, 1995.

[26] Synopsys, Inc., *Synopsys Online Documentation, v1997.01*, 1997.

[27] Cadence Design Systems, Inc., *Cadence Online Library*, 1994.

[28] Courtiat, J. P., "Estelle *: A powerful Dialect of Estelle for OSI Protocol description", in *International Workshop on Protocol Specification, Verification and Testing*, Toulouse-Moissac, França, 1988.

[29] Jerraya, A. A., Aichouchi, M., Berrebi, E. et al., *AMICAL: Interative Architectural Synthesis Based on VHDL.* AMICAL Tutorial, Internal Report INPG/TIMA, abril 1994.

[30] Rahmouni, M., O'Brien, K., "A Loop-Based Scheduling Scheduling Algorithm for Hardware Description Languages", *Journal of Parallel Processing Letters*, vol. 4, no. 3, no. 3, pp. 351–364, 1994.

[31] O'Brien, K., Rahmouni, M., Jerraya, A. A., "DLS: A Scheduling Algorithm for High-Level Synthesis in VHDL", in *EDAC'93*, Paris, Franca, 1993.

[32] Gajski, Daniel D. et al., *High-Level Synthesis - Introduction to Chip and System Design*. University of California, Irvine: Kluver Academic Publishers, 1992.

[33] Camposano, R., "Path-Based Scheduling for Synthesis", *IEEE Transactions on CAD*, vol. 10, pp. 85–93, Jan 1991.

[34] Camposano, R., Bergamaschi, R. A., "Synthesis Using Path-Based Scheduling: Algoritms And Exercises", in *27th DAC*, Orlando, pp. 450–455, Jun 1990.

[35] Postel, J. (ed), *Transmission Control Protocol*. RFC 793, setembro 1981.

[36] Postel, J. (ed), *Internet Protocol - DARPA Internet Program Protocol Specification*. RFC 791, USC/Information Sciences Institute, setembro 1981.

[37] Newman, P., "ATM Local Area Networks", *IEEE Communications Magazine*, Mar 1994.

[38] Biagioni, E., Cooper, E. et al., "Designing a Pratical ATM LAN", *IEEE Network Magazine*, Mar 1993.

[39] The ATM Forum, LAN Emulation SWG Drafting Group, *LAN Emulation Over ATM Specification, Version 1.0*, Jan 1995.

[40] Chao, H. J., Ghosal, D. et al., "IP over ATM Local Area Networks", *IEEE Communications Magazine*, Ago 1994.

[41] Brazdziunas, C., *IPng Support for ATM Services*. RFC 1680, Bellcore, Ago 1994.

[42] Partridge, C., *Using the Flow Label Field in IPv6*. RFC 1809, BBN Systems and Technologies, Jun 1995.

[43] RFC 1881, IAB and IESG, *IPv6 Address Allocation Management*, Dez 1995.

[44] Hinden, R., Deering, S., *IP version 6 Addressing Architecture.* RFC 1884, Ipsilon Networks, Xerox PARC, Dez 1995.

[45] Conta, A., Deering, S., *Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification.* RFC 1885, Digital Equipament Corporation, Xerox PARC, Dez 1995.

[46] Thomson, S., Huitema, C., *DNS Extensions to support IP version 6.* RFC 1886, Bellcore, INRIA, Dez 1995.

[47] Gilligan, R., Nordmark, E., *Transition Mechanisms for IPv6 Hosts and Routers.* RFC 1933, Sun Microsystems, Inc., abril 1996.

[48] Vickers, B. J., Suda, T., "Connectionless Service for Public ATM Networks", *IEEE Communications Magazine*, Ago 1994.

[49] Comer, D. E., *Internetworking with TCP/IP*, vol. 1. New Jersey: Prentice Hall, 3 ed., 1995.

[50] Soares, Luiz Fernando G., *Redes de Computadores: das LANs, MANs e WANs às redes ATM.* Rio de Janeiro: Ed. Campus, 2 ed., 1995.

[51] Atkinson, R., *Default IP MTU for use over ATM AAL5.* RFC 1626, Naval Research Laboratory, Mai 1994.

[52] Heinanen, J., *Multiprotocol Encapsulation over ATM Adaptation Layer 5.* RFC 1483, Telecom Finland, julho 1993.

[53] Thomas, Stephen A., *IPng and the TCP/IP protocolos: implementing the next generation internet.* New York: Wiley Computer Publishing, 1996.

[54] Perez, M., Liaw, F. et al., *ATM Signaling Support for IP over ATM.* RFC 1755, ISI, FORE System, Inc., Motorola Codex, Ascom Timeplex, Inc., Fev 1995.

[55] Braden, R. (ed), *Requirements for Internet Hosts – Communication Layers.* RFC 1122, Internet Engineering Task Force, Out 1989.

Apêndice A

Especificação Estelle do Protocolo IPv6

Abaixo encontra-se a especificação completa em Estelle do protocolo IPv6. Nas seções seguintes são mostrados os resultados de algumas simulações realizadas utilizando-se o simulador Estelle pertencente ao ambiente C.A.D. Estelle.

```
(*                  Subgrupo do protocolo IPv6.                        *)
(* Esta especificacao envolve, alem da entrega de datagramas completos, *)
(* a fragmentacao e a concatenacao dos mesmos.                          *)

(* Inicio: 20/07/97 *)
(* Termino: 10/09/97 *)

specification IPv6 systemactivity;
default individual queue;
timescale seconds;

{$ external NetUserUDP2}
  type
  string = array [0..255] of char;

  function  KeyPressed : boolean;  primitive;
  function  ReadKey    : char;     primitive;
  procedure WriteLn;              primitive;
  procedure WriteString (s : string); primitive;
  procedure ReadString  (VAR s : string); primitive;
  procedure WriteInt    (i : integer);    primitive;
  procedure ReadInt     (i : integer);    primitive;
  procedure ClrScr;                       primitive;
  procedure Copy (str : string; inx : integer; len : integer; VAR result: string);        primitive;
  function Lenght (VAR str : string) : integer;           primitive;
  procedure Delete (VAR str : string; inx : integer; len : integer);              primitive;
  procedure Concat (s1 : string; s2 : string; VAR dest : string);                 primitive;
  function Random (i : integer) : integer; primitive;
  const
```

```
   MTU = 70; (* tamanho maximo que o datagrama deve ter *)
   N   = 9; (* numero maximo do buffer de fragmentos *)
  type
   destino      = integer;

   IP_PDU_type1 = RECORD
LLC          : integer;
LLCOUI1      : integer;
OUI2         : integer;
TIPO         : integer;
VerPriFlow   : array [0..1] of integer;
PayLength    : integer;
NextHop      : integer;
SourceAddr   : array [0..7] of integer;
DestAddr     : array [0..7] of integer;
dados        : array [0..49] of char;
   END;

   IP_PDU_type2 = RECORD
LLC          : integer;
LLCOUI1      : integer;
OUI2         : integer;
TIPO         : integer;
VerPriFlow   : array [0..1] of integer;
PayLength    : integer;
NextHop      : integer;
SourceAddr   : array [0..7] of integer;
DestAddr     : array [0..7] of integer;
Next2Res1    : integer;
FragoffRes2M : integer;
Id           : array [0..1] of integer;
dados        : array [0..49] of char;
   END;

   IP_PDU_type3 = RECORD
VerPriFlow   : array [0..1] of integer;
PayLength    : integer;
NextHop      : integer;
SourceAddr   : array [0..7] of integer;
DestAddr     : array [0..7] of integer;
Dados        : array [0..49] of char;
   END;

   IP_PDU_type4 = RECORD
VerPriFlow   : array [0..1] of integer;
PayLength    : integer;
NextHop      : integer;
SourceAddr   : array [0..7] of integer;
DestAddr     : array [0..7] of integer;
Next2Res1    : integer;
FragoffRes2M : integer;
Id           : array [0..1] of integer;
dados        : array [0..49] of char;
   END;

   SDU_type1 = RECORD
ipsrc : array [0..7] of char;
ipdest : array [0..7] of char;
FL     : integer;
Pr     : integer;
HP     : integer;
SDU    : array [0..49] of char;

   END;
```

```
   SDU_type2 = RECORD
ipsrc : array [0..7] of char;
ipdest : array [0..7] of char;
SDU    : array [0..49] of char;
   END;

   nomemaq       = array [0..9] of char;
   maqpos        = array [0..5] of nomemaq;
   enderecoIPbin  = array [0..127] of char;
   enderecoIPhex  = array [0..7] of char;
   enderecopos    = array [0..5] of enderecoIPhex;
   stringbit3     = array [0..2] of char;
   stringbit4     = array [0..3] of char;
   stringbit8     = array [0..7] of char;
   stringbit13    = array [0..12] of char;
   stringbit16    = array [0..15] of char;
   stringbit24    = array [0..23] of char;
   stringbit32    = array [0..31] of char;
   stringbit48    = array [0..47] of char;
   stringbit50    = array [0..49] of char;
   ident          = array [0..N] of stringbit48;
   dadosrec       = array [0..N] of stringbit50;
   addr_aux       = array [0..7] of integer;
   verpriflow_aux = array [0..1] of integer;

(* definicao dos canais *)

channel medium_interface (user,provider);
  by user:
      AAL_Unitdata1_Req (prox_no : destino ; PDU1 : IP_PDU_type1);
      AAL_Unitdata2_Req (prox_no : destino ; PDU2 : IP_PDU_type2);
  by provider:
      AAL_Unitdata1_Ind (PDU3 : IP_PDU_type3);
      AAL_Unitdata2_Ind (PDU4 : IP_PDU_type4);

channel IP_Interface (user,provider);
  by user:
      IP_Data_Req (SDU1 : SDU_type1);
  by provider:
      IP_Data_Ind (SDU2 : SDU_type2);

channel SYS_Interface (user,provider);
   by user:
       ENVIAR;

(* definicao do modulo AAL5 *)

module AAL5_type activity;
  ip AAL5: array [0..5] of medium_interface (provider);
end;

body AAL5_body for AAL5_type;
 var
  IP_PDU_aux3 : IP_PDU_type3;
  IP_PDU_aux4 : IP_PDU_type4;
  dest        : integer;
  i           : integer;
  num         : integer;

procedure VerificaDest (int : integer; var dest : integer);
begin
  case int of
    13659 : dest := 0;
```

```
    13639 : dest := 1;
    13633 : dest := 2;
    13651 : dest := 3;
    13634 : dest := 4;
    13640 : dest := 5;
  end;
end;
  initialize
    name AAL5aberto:
    begin
    end;
trans
   when AAL5[0].AAL_Unitdata1_Req
   name AAL0Enviando_dat:
   begin
     WriteString (' PDU1.dados :');
     WriteString (PDU1.dados);
     WriteLn;
     VerificaDest (prox_no,dest);
     IP_PDU_aux3.VerPriFlow[0] := PDU1.VerPriFlow[0];
     IP_PDU_aux3.VerPriFlow[1] := PDU1.VerPriFlow[1];
     IP_PDU_aux3.PayLength := PDU1.PayLength;
     IP_PDU_aux3.NextHop := PDU1.NextHop;
     for i:= 0 to 7 do
     begin
       IP_PDU_aux3.SourceAddr[i] := PDU1.SourceAddr[i];
       IP_PDU_aux3.DestAddr[i] := PDU1.DestAddr[i];
     end;
     Copy (PDU1.dados,0,50,IP_PDU_aux3.dados);
     output AAL5[dest].AAL_Unitdata1_Ind (IP_PDU_aux3);
   end;

   when AAL5[0].AAL_Unitdata2_Req
   name AAL0Enviando_frag:
   begin
     WriteString (' PDU2.dados :');
     WriteString (PDU2.dados);
     WriteLn;
     VerificaDest (prox_no,dest);
     IP_PDU_aux4.VerPriFlow[0] := PDU2.VerPriFlow[0];
     IP_PDU_aux4.VerPriFlow[1] := PDU2.VerPriFlow[1];
     IP_PDU_aux4.PayLength := PDU2.PayLength;
     IP_PDU_aux4.NextHop := PDU2.NextHop;
     for i:= 0 to 7 do
     begin
       IP_PDU_aux4.SourceAddr[i] := PDU2.SourceAddr[i];
       IP_PDU_aux4.DestAddr[i] := PDU2.DestAddr[i];
     end;
     IP_PDU_aux4.Next2Res1 := PDU2.Next2Res1;
     IP_PDU_aux4.FragoffRes2M := PDU2.FragoffRes2M;
     IP_PDU_aux4.Id[0] := PDU2.Id[0];
     IP_PDU_aux4.Id[1] := PDU2.Id[1];
     Copy (PDU2.dados,0,50,IP_PDU_aux4.dados);
     num := Random (100);
     WriteInt (num);
     if num > 3 then output AAL5[dest].AAL_Unitdata2_Ind (IP_PDU_aux4);
   end;

   when AAL5[1].AAL_Unitdata1_Req
   name AAL1Enviando_dat:
   begin
     WriteString (' PDU1.dados :');
     WriteString (PDU1.dados);
     WriteLn;
```

```
  VerificaDest (prox_no,dest);
  IP_PDU_aux3.VerPriFlow[0] := PDU1.VerPriFlow[0];
  IP_PDU_aux3.VerPriFlow[1] := PDU1.VerPriFlow[1];
  IP_PDU_aux3.PayLength := PDU1.PayLength;
  IP_PDU_aux3.NextHop := PDU1.NextHop;
  for i:= 0 to 7 do
  begin
    IP_PDU_aux3.SourceAddr[i] := PDU1.SourceAddr[i];
    IP_PDU_aux3.DestAddr[i] := PDU1.DestAddr[i];
  end;
  Copy (PDU1.dados,0,50,IP_PDU_aux3.dados);
  output AAL5[dest].AAL_Unitdata1_Ind (IP_PDU_aux3);
end;

when AAL5[1].AAL_Unitdata2_Req
name AAL1Enviando_frag:
begin
  WriteString (' PDU2.dados :');
  WriteString (PDU2.dados);
  WriteLn;
  VerificaDest (prox_no,dest);
  IP_PDU_aux4.VerPriFlow[0] := PDU2.VerPriFlow[0];
  IP_PDU_aux4.VerPriFlow[1] := PDU2.VerPriFlow[1];
  IP_PDU_aux4.PayLength := PDU2.PayLength;
  IP_PDU_aux4.NextHop := PDU2.NextHop;
  for i:= 0 to 7 do
  begin
    IP_PDU_aux4.SourceAddr[i] := PDU2.SourceAddr[i];
    IP_PDU_aux4.DestAddr[i] := PDU2.DestAddr[i];
  end;
  IP_PDU_aux4.Next2Res1 := PDU2.Next2Res1;
  IP_PDU_aux4.FragoffRes2M := PDU2.FragoffRes2M;
  IP_PDU_aux4.Id[0] := PDU2.Id[0];
  IP_PDU_aux4.Id[1] := PDU2.Id[1];
  Copy (PDU2.dados,0,50,IP_PDU_aux4.dados);
  num := Random (100);
  WriteInt (num);
  if num > 3 then output AAL5[dest].AAL_Unitdata2_Ind (IP_PDU_aux4);
end;

when AAL5[2].AAL_Unitdata1_Req
name AAL2Enviando_dat:
begin
  WriteString (' PDU1.dados :');
  WriteString (PDU1.dados);
  WriteLn;
  VerificaDest (prox_no,dest);
  IP_PDU_aux3.VerPriFlow[0] := PDU1.VerPriFlow[0];
  IP_PDU_aux3.VerPriFlow[1] := PDU1.VerPriFlow[1];
  IP_PDU_aux3.PayLength := PDU1.PayLength;
  IP_PDU_aux3.NextHop := PDU1.NextHop;
  for i:= 0 to 7 do
  begin
    IP_PDU_aux3.SourceAddr[i] := PDU1.SourceAddr[i];
    IP_PDU_aux3.DestAddr[i] := PDU1.DestAddr[i];
  end;
  Copy (PDU1.dados,0,50,IP_PDU_aux3.dados);
  output AAL5[dest].AAL_Unitdata1_Ind (IP_PDU_aux3);
end;

when AAL5[2].AAL_Unitdata2_Req
name AAL2Enviando_frag:
begin
  WriteString (' PDU2.dados :');
```

```
  WriteString (PDU2.dados);
  WriteLn;
  VerificaDest (prox_no,dest);
  IP_PDU_aux4.VerPriFlow[0] := PDU2.VerPriFlow[0];
  IP_PDU_aux4.VerPriFlow[1] := PDU2.VerPriFlow[1];
  IP_PDU_aux4.PayLength := PDU2.PayLength;
  IP_PDU_aux4.NextHop := PDU2.NextHop;
  for i:= 0 to 7 do
  begin
    IP_PDU_aux4.SourceAddr[i] := PDU2.SourceAddr[i];
    IP_PDU_aux4.DestAddr[i] := PDU2.DestAddr[i];
  end;
  IP_PDU_aux4.Next2Res1 := PDU2.Next2Res1;
  IP_PDU_aux4.FragoffRes2M := PDU2.FragoffRes2M;
  IP_PDU_aux4.Id[0] := PDU2.Id[0];
  IP_PDU_aux4.Id[1] := PDU2.Id[1];
  Copy (PDU2.dados,0,50,IP_PDU_aux4.dados);
  num := Random (100);
  WriteInt (num);
  if num > 3 then output AAL5[dest].AAL_Unitdata2_Ind (IP_PDU_aux4);
end;

when AAL5[3].AAL_Unitdata1_Req
name AAL3Enviando_dat:
begin
  WriteString (' PDU1.dados :');
  WriteString (PDU1.dados);
  WriteLn;
  VerificaDest (prox_no,dest);
  IP_PDU_aux3.VerPriFlow[0] := PDU1.VerPriFlow[0];
  IP_PDU_aux3.VerPriFlow[1] := PDU1.VerPriFlow[1];
  IP_PDU_aux3.PayLength := PDU1.PayLength;
  IP_PDU_aux3.NextHop := PDU1.NextHop;
  for i:= 0 to 7 do
  begin
    IP_PDU_aux3.SourceAddr[i] := PDU1.SourceAddr[i];
    IP_PDU_aux3.DestAddr[i] := PDU1.DestAddr[i];
  end;
  Copy (PDU1.dados,0,50,IP_PDU_aux3.dados);
  output AAL5[dest].AAL_Unitdata1_Ind (IP_PDU_aux3);
end;

when AAL5[3].AAL_Unitdata2_Req
name AAL3Enviando_frag:
begin
  WriteString (' PDU2.dados :');
  WriteString (PDU2.dados);
  WriteLn;
  VerificaDest (prox_no,dest);
  IP_PDU_aux4.VerPriFlow[0] := PDU2.VerPriFlow[0];
  IP_PDU_aux4.VerPriFlow[1] := PDU2.VerPriFlow[1];
  IP_PDU_aux4.PayLength := PDU2.PayLength;
  IP_PDU_aux4.NextHop := PDU2.NextHop;
  for i:= 0 to 7 do
  begin
    IP_PDU_aux4.SourceAddr[i] := PDU2.SourceAddr[i];
    IP_PDU_aux4.DestAddr[i] := PDU2.DestAddr[i];
  end;
  IP_PDU_aux4.Next2Res1 := PDU2.Next2Res1;
  IP_PDU_aux4.FragoffRes2M := PDU2.FragoffRes2M;
  IP_PDU_aux4.Id[0] := PDU2.Id[0];
  IP_PDU_aux4.Id[1] := PDU2.Id[1];
  Copy (PDU2.dados,0,50,IP_PDU_aux4.dados);
  num := Random (100);
```

```
  WriteInt (num);
  if num > 3 then output AAL5[dest].AAL_Unitdata2_Ind (IP_PDU_aux4);
end;

when AAL5[4].AAL_Unitdata1_Req
name AAL4Enviando_dat:
begin
  WriteString (' PDU1.dados :');
  WriteString (PDU1.dados);
  WriteLn;
  VerificaDest (prox_no,dest);
  IP_PDU_aux3.VerPriFlow[0] := PDU1.VerPriFlow[0];
  IP_PDU_aux3.VerPriFlow[1] := PDU1.VerPriFlow[1];
  IP_PDU_aux3.PayLength := PDU1.PayLength;
  IP_PDU_aux3.NextHop := PDU1.NextHop;
  for i:= 0 to 7 do
  begin
    IP_PDU_aux3.SourceAddr[i] := PDU1.SourceAddr[i];
    IP_PDU_aux3.DestAddr[i] := PDU1.DestAddr[i];
  end;
  Copy (PDU1.dados,0,50,IP_PDU_aux3.dados);
  output AAL5[dest].AAL_Unitdata1_Ind (IP_PDU_aux3);
end;

when AAL5[4].AAL_Unitdata2_Req
name AAL4Enviando_frag:
begin
  WriteString (' PDU2.dados :');
  WriteString (PDU2.dados);
  WriteLn;
  VerificaDest (prox_no,dest);
  IP_PDU_aux4.VerPriFlow[0] := PDU2.VerPriFlow[0];
  IP_PDU_aux4.VerPriFlow[1] := PDU2.VerPriFlow[1];
  IP_PDU_aux4.PayLength := PDU2.PayLength;
  IP_PDU_aux4.NextHop := PDU2.NextHop;
  for i:= 0 to 7 do
  begin
    IP_PDU_aux4.SourceAddr[i] := PDU2.SourceAddr[i];
    IP_PDU_aux4.DestAddr[i] := PDU2.DestAddr[i];
  end;
  IP_PDU_aux4.Next2Res1 := PDU2.Next2Res1;
  IP_PDU_aux4.FragoffRes2M := PDU2.FragoffRes2M;
  IP_PDU_aux4.Id[0] := PDU2.Id[0];
  IP_PDU_aux4.Id[1] := PDU2.Id[1];
  Copy (PDU2.dados,0,50,IP_PDU_aux4.dados);
  num := Random (100);
  WriteInt (num);
  if num > 3 then output AAL5[dest].AAL_Unitdata2_Ind (IP_PDU_aux4);
end;

when AAL5[5].AAL_Unitdata1_Req
name AAL5Enviando_dat:
begin
  WriteString (' PDU1.dados :');
  WriteString (PDU1.dados);
  WriteLn;
  VerificaDest (prox_no,dest);
  IP_PDU_aux3.VerPriFlow[0] := PDU1.VerPriFlow[0];
  IP_PDU_aux3.VerPriFlow[1] := PDU1.VerPriFlow[1];
  IP_PDU_aux3.PayLength := PDU1.PayLength;
  IP_PDU_aux3.NextHop := PDU1.NextHop;
  for i:= 0 to 7 do
  begin
    IP_PDU_aux3.SourceAddr[i] := PDU1.SourceAddr[i];
```

```
      IP_PDU_aux3.DestAddr[i] := PDU1.DestAddr[i];
    end;
    Copy (PDU1.dados,0,50,IP_PDU_aux3.dados);
    output AAL5[dest].AAL_Unitdata1_Ind (IP_PDU_aux3);
  end;

  when AAL5[5].AAL_Unitdata2_Req
  name AAL5Enviando_frag:
  begin
    WriteString (' PDU2.dados :');
    WriteString (PDU2.dados);
    WriteLn;
    VerificaDest (prox_no,dest);
    IP_PDU_aux4.VerPriFlow[0] := PDU2.VerPriFlow[0];
    IP_PDU_aux4.VerPriFlow[1] := PDU2.VerPriFlow[1];
    IP_PDU_aux4.PayLength := PDU2.PayLength;
    IP_PDU_aux4.NextHop := PDU2.NextHop;
    for i:= 0 to 7 do
    begin
      IP_PDU_aux4.SourceAddr[i] := PDU2.SourceAddr[i];
      IP_PDU_aux4.DestAddr[i] := PDU2.DestAddr[i];
    end;
    IP_PDU_aux4.Next2Res1 := PDU2.Next2Res1;
    IP_PDU_aux4.FragoffRes2M := PDU2.FragoffRes2M;
    IP_PDU_aux4.Id[0] := PDU2.Id[0];
    IP_PDU_aux4.Id[1] := PDU2.Id[1];
    Copy (PDU2.dados,0,50,IP_PDU_aux4.dados);
    num := Random (100);
    WriteInt (num);
    if num > 3 then output AAL5[dest].AAL_Unitdata2_Ind (IP_PDU_aux4);
  end;

end; (* AAL5_body *)

module IP_type activity (aux_id_est : integer);
  ip UIPCEP: IP_Interface (provider);
     MIPCEP: medium_interface (user);
end;

body IP_body for IP_type;
 const
  P = 10; (* tempo de espera *)
 var
  id_est             : integer;
  IP_PDU_aux1        : IP_PDU_type1;
  IP_PDU_aux2        : IP_PDU_type2;
  SDU_aux2           : SDU_type2;
  ipsrcbin           : enderecoIPbin;
  ipdestbin          : enderecoIPbin;
  M                  : integer;
  NFB                : integer;
  NFB_aux            : integer;
  len                : integer;
  Next2bin           : stringbit8;
  Res1bin            : stringbit8;
  Fragoff            : integer;
  posini             : integer;
  posfim             : integer;
  dados_env          : integer;
  s                  : stringbit4;
  id_aux             : integer;
  len_aux            : integer;
  identificacao      : stringbit48;
  Fragoff_aux        : integer;
```

```
  PayLength_pri     : array [0..N] of integer;
  tamanho_dados_pri : array [0..N] of integer;
  tamanho_dados_ult : array [0..N] of integer;
  timer             : array [0..N] of integer;
  descartado        : boolean;
  existe            : array [0..N] of stringbit48;
  chegou_algum      : integer;
  item_aux          : integer;
  Pr                : integer;
  FL                : integer;
  HP                : integer;
  ident_aux         : ident;
  sourceaddr1       : addr_aux;
  sourceaddr2       : addr_aux;
  destaddr1         : addr_aux;
  destaddr2         : addr_aux;
  verpriflow1       : verpriflow_aux;
  verpriflow2       : verpriflow_aux;
  Idbin             : stringbit32;
  Fragoffbin        : stringbit13;
  Res2Mbin          : stringbit3;
  dadosrec_aux      : dadosrec;
  FragoffRes2M_aux  : integer;
  i                 : integer;
  NextHop_aux       : integer;
  auxMbin           : array [0..0] of char;
  auxMbin_aux       : char;
  inicio            : integer;
  prox_no_aux       : destino;

state
  Repouso;
(* Espaco destinado aos procedures e as funcoes necessarias *)
(* A funcao case soh aceita como variavel de entrada tipos pre-definidos,  *)
(* como char e integer.                                                    *)

procedure TabelaHexBin (hex: char; var bin : stringbit4);
begin
  case hex of
   '0' : bin := '0000';
   '1' : bin := '0001';
   '2' : bin := '0010';
   '3' : bin := '0011';
   '4' : bin := '0100';
   '5' : bin := '0101';
   '6' : bin := '0110';
   '7' : bin := '0111';
   '8' : bin := '1000';
   '9' : bin := '1001';
   'A' : bin := '1010';
   'B' : bin := '1011';
   'C' : bin := '1100';
   'D' : bin := '1101';
   'E' : bin := '1110';
   'F' : bin := '1111';
 end;
end;

procedure TabelaBinHex (bin : stringbit4; var hex : char);
var
 aux : array [0..0] of char;
 aux1 : char;
begin
  if bin = '0000' then hex := '0'
```

```
  else
   if bin = '0001' then hex := '1'
   else
    if bin = '0010' then hex := '2'
    else
     if bin = '0011' then hex := '3'
     else
      if bin = '0100' then hex := '4'
      else
       if bin = '0101' then hex := '5'
       else
if bin = '0110' then hex := '6'
else
 if bin = '0111' then hex := '7'
 else
  if bin = '1000' then hex := '8'
  else
   if bin = '1001' then hex := '9'
   else
    if bin = '1010' then hex := 'A'
    else
     if bin = '1011' then hex := 'B'
     else
      if bin = '1100' then hex := 'C'
      else
       if bin = '1101' then hex := 'D'
       else
if bin = '1110' then hex := 'E'
else
 if bin = '1111' then hex := 'F';
end;

procedure ConvIPhexIPbin (SDU_aux1 : SDU_type1; var End2IP: enderecoIPbin; var End3IP : enderecoIPbin);
var
 nome3bin   : stringbit4;
 nome2bin   : stringbit4;
 i,j,i_aux  : integer;
 len        : integer;
begin
 i_aux := 0;
 while i_aux <= 95 do
  begin
    End2IP[i_aux] := '0';
    End3IP[i_aux] := '0';
    i_aux := i_aux + 1;
  end;
 for i := 0 to 7 do
  begin
    TabelaHexBin (SDU_aux1.ipsrc[i],nome2bin);
    TabelaHexBin (SDU_aux1.ipdest[i],nome3bin);
    for j := 0 to 3 do
     begin
       End2IP[i_aux] := nome2bin[j];
       End3IP[i_aux] := nome3bin[j];
       i_aux := i_aux + 1;
     end;
  end;
end;

procedure ConvIPbinIPhex (End4IP : enderecoIPbin; End5IP : enderecoIPbin; var SDU_aux2 : SDU_type2);
  var
   i,i_aux,j : integer;
   nome3bin  : stringbit4;
   nome4bin  : stringbit4;
```

```
   numhex    : char;
   numhex1   : char;
  begin
    i := 96;
    j := 0;
    while i <= 127 do
     begin
       for i_aux := 0 to 3 do
begin
  nome3bin[i_aux] := End4IP[i + i_aux];
  nome4bin[i_aux] := End5IP[i + i_aux];
end;
       TabelaBinHex (nome3bin,numhex);
       TabelaBinHex (nome4bin,numhex1);
       SDU_aux2.ipsrc[j] := numhex;
       SDU_aux2.ipdest[j] := numhex1;
       j := j + 1;
       i := i + 4;
     end;
  end;

procedure ConvIPbinIPint (nome3 : enderecoIPbin; var inteiro : addr_aux);
var
 aux : array [0..0] of char;
 aux1 : char;
 i,j,i_aux,soma1,soma2,j_aux : integer;
 K1, K2 : integer;
begin
 soma1 := 0;
 soma2 := 0;
 j := 97;
 K1 := 16384;
 K2 := 16384;
 for i := 0 to 5 do
  begin
    inteiro[i] := 0;
  end;
 while (j < 127) do
  begin
    while (j <= 111) do
     begin
       aux1 := nome3[j];
       aux[0] := aux1;
       if aux = '1' then
       soma1 := soma1 + K1;
       K1 := K1 div 2;
       j := j + 1;
     end;
    j := j + 1;
    aux1 := nome3[j];
    aux[0] := aux1;
    if aux = '1' then
    soma2 := soma2 + K2;
    K2 := K2 div 2;
  end;
 inteiro[6] := soma1;
 inteiro[7] := soma2;
end;

procedure ConvIPintIPbin (enderecoint : addr_aux; var enderecobin : enderecoIPbin);
  var
   ind      : integer;
   ind_aux  : integer;
   int1     : integer;
```

```
   int2     : integer;
   int1_aux : integer;
   int2_aux : integer;
  begin
    for ind := 0 to 95 do
     begin
       enderecobin[ind] := '0';
     end;
    int1 := enderecoint[6];
    int2 := enderecoint[7];
    if int1 = 0 then
     begin
       for ind := 96 to 111 do
begin
  enderecobin[ind] := '0';
end;
     end
    else
     begin
       ind_aux := 111;
       while int1 > 1 do
begin
  int1_aux := int1 mod 2;
  if int1_aux = 1 then enderecobin[ind_aux] := '1'
  else enderecobin[ind_aux] := '0';
  int1 := int1 div 2;
  ind_aux := ind_aux - 1;
end;
       enderecobin[ind_aux] := '1';
       while ind_aux <> 96 do
begin
  ind_aux := ind_aux - 1;
  enderecobin[ind_aux] := '0';
end;
     end;
    if int2 = 0 then
     begin
       for ind := 112 to 127 do
begin
  enderecobin[ind] := '0';
end;
     end
    else
     begin
       ind_aux := 127;
       while int2 > 1 do
begin
  int2_aux := int2 mod 2;
  if int2_aux = 1 then enderecobin[ind_aux] := '1'
  else enderecobin[ind_aux] := '0';
  int2 := int2 div 2;
  ind_aux := ind_aux - 1;
end;
       enderecobin[ind_aux] := '1';
       while ind_aux <> 112 do
begin
  ind_aux := ind_aux - 1;
  enderecobin[ind_aux] := '0';
end;
     end;
  end;

procedure StoVerPriFlow (Campo2 : integer; Campo3 : integer; var Campos123 : verpriflow_aux);
var
```

```
 aux : array [0..0] of char;
 aux1 : char;
 Version       : stringbit4;
 Campo2_aux    : stringbit4;
 Campo3_aux    : stringbit24;
 VerPriFlowbin : stringbit32;
 aux2,aux3     : stringbit16;
 i1,i2,i3,i4,indice1,indice2,K1,K2,K3,K4,soma3,soma4 : integer;
begin
 Version := '0110';
 i1 := Pr;
 i2 := FL;
 i3 := 1;
 i4 := 1;
 K3 := 16384;
 K4 := 16384;
 soma3 := 0;
 soma4 := 0;
 if i1 = 0 then Campo2_aux := '0000'
 else
  begin
    indice1 := 3;
    while i1 > 1 do
      begin
K1 := i1 mod 2;
if K1 = 1 then Campo2_aux[indice1] := '1'
else
Campo2_aux[indice1] := '0';
i1 := i1 div 2;
indice1 := indice1 - 1;
      end;
    Campo2_aux[indice1] := '1';
    while indice1 <> 0 do
      begin
indice1 := indice1 - 1;
Campo2_aux[indice1] := '0';
      end;
  end;
 if i2 = 0 then Campo3_aux := '000000000000000000000000'
 else
  begin
    indice2 := 23;
    while i2 > 1 do
      begin
K2 := i2 mod 2;
if K2 = 1 then Campo3_aux[indice2] := '1'
else
Campo3_aux[indice2] := '0';
i2 := i2 div 2;
indice2 := indice2 - 1;
      end;
    Campo3_aux[indice2] := '1';
    while indice2 <> 0 do
      begin
indice2 := indice2 - 1;
Campo3_aux[indice2] := '0';
      end;
  end;
 Concat (Version,Campo2_aux,VerPriFlowbin);
 Concat (VerPriFlowbin,Campo3_aux,VerPriFlowbin);
 Copy (VerPriFlowbin,0,16,aux2);
 Copy (VerpriFlowbin,16,16,aux3);
 while i3 <= 15 do
   begin
```

```
    aux1 := aux2[i3];
    aux[0] := aux1;
    if aux = '1' then
    soma3 := soma3 + K3;
    K3 := K3 div 2;
    i3 := i3 + 1;
  end;
 Campos123[0] := soma3;
 while i4 <= 15 do
  begin
    aux1 := aux3[i4];
    aux[0] := aux1;
    if aux = '1' then
    soma4 := soma4 + K4;
    K4 := K4 div 2;
    i4 := i4 + 1;
  end;
 Campos123[1] := soma4;
end;

procedure RclVerPriFlow (inteiro : integer; var s : stringbit4);
  var
   indice : integer;
   K      : integer;
   out    : stringbit16;
   begin
     indice := 15;
     while inteiro > 1 do
      begin
K := inteiro mod 2;
if K = 1 then out[indice] := '1'
else out[indice] := '0';
inteiro := inteiro div 2;
indice := indice - 1;
      end;
      out[indice] := '1';
      while indice <> 0 do
       begin
 indice := indice - 1;
 out[indice] := '0';
       end;
      Copy (out,0,4,s);
   end;

procedure StoNextHop (SDU_aux1 : SDU_type1; var NextHop_aux : integer);
var
  len,len_aux            : integer;
  Hopbin,NextHeader      : stringbit8;
  indice,i,i2,K,K2,soma  : integer;
  aux                    : stringbit16;
  aux1                   : array [0..0] of char;
  aux2                   : char;
 begin
   i := HP;
   i2 := 1;
   K2 := 16384;
   soma := 0;
   indice := 7;
   len := Lenght (SDU_aux1.SDU);
   len_aux := len + 40;
   if len_aux > MTU then
     NextHeader := '00101100'
   else
     NextHeader := '00000110';
```

```
   while i > 1 do
     begin
       K := i mod 2;
       if K = 1 then Hopbin[indice] := '1'
       else
       Hopbin[indice] := '0';
       i := i div 2;
       indice := indice - 1;
     end;
   Hopbin[indice] := '1';
   while indice <> 0 do
     begin
       indice := indice - 1;
       Hopbin := '0';
     end;
   Concat (NextHeader,Hopbin,aux);
   while i2 <= 15 do
     begin
       aux2 := aux[i2];
       aux1[0] := aux2;
       if aux1 = '1' then
       soma := soma + K;
       K := K div 2;
       i2 := i2 + 1;
     end;
   NextHop_aux := soma;
 end;

procedure StoNext2Res1 (nome4 : stringbit8; nome5 : stringbit8; var IP_PDU_aux2 : IP_PDU_type2);
  var
   aux : array [0..0] of char;
   aux1 : char;
   Next2Res1bin  : stringbit16;
   indice,soma,K : integer;
  begin
    K := 16384;
    indice := 1;
    soma := 0;
    Concat (nome4,nome5,Next2Res1bin);
    while indice <= 15 do
      begin
aux1 := Next2Res1bin[indice];
aux[0] := aux1;
if aux = '1' then
soma := soma + K;
K := K div 2;
indice := indice + 1;
      end;
    IP_PDU_aux2.Next2Res1 := soma;
  end;

procedure DetFragoffRes2M (int : integer; int2 : integer; var IP_PDU_aux2 : IP_PDU_type2);
  var
   Fragoffbin      : stringbit13;
   FragoffRes2Mbin : stringbit16;
   Res2Mbin        : stringbit3;
   indice          : integer;
   indice2         : integer;
   soma            : integer;
   K               : integer;
   K2              : integer;
   aux             : array [0..0] of char;
   aux1            : char;
  begin
```

```
    K2 := 16384;
    indice2 := 1;
    soma := 0;
    Res2Mbin[0] := '0';
    Res2Mbin[1] := '0';
    if int2 = 1 then Res2Mbin[2] := '1';
    if int2 = 0 then Res2Mbin[2] := '0';
    if int = 0 then
    Fragoffbin := '0000000000000'
    else
begin
  indice := 12;
  while int > 1 do
   begin
     K := int mod 2;
     if K = 1 then Fragoffbin[indice] := '1'
     else
     Fragoffbin[indice] := '0';
     int := int div 2;
     indice := indice - 1;
   end;
  Fragoffbin[indice] := '1';
  while indice <> 0 do
    begin
      indice := indice - 1;
      Fragoffbin[indice] := '0';
    end;
end;
   Concat(Fragoffbin,Res2Mbin,FragoffRes2Mbin);
   WriteString ('FragoffRes2Mbin: ');
   WriteString (FragoffRes2Mbin);
   WriteLn;
   while indice2 <= 15 do
    begin
      aux1 := FragoffRes2Mbin[indice2];
      aux[0] := aux1;
      if aux = '1' then
      soma := soma + K2;
      K2 := K2 div 2;
      indice2 := indice2 + 1;
    end;
   IP_PDU_aux2.FragoffRes2M := soma;
  end;

procedure RclFragoffRes2M (ent : integer; var Fragoffbin : stringbit13; var Res2Mbin : stringbit3);
  var
   FragoffRes2Mbin : stringbit16;
   indice : integer;
   K      : integer;
begin
     indice := 15;
     while ent > 1 do
     begin
       K := ent mod 2;
       if K = 1 then FragoffRes2Mbin[indice] := '1'
       else FragoffRes2Mbin[indice] := '0';
       ent := ent div 2;
       indice := indice - 1;
     end;
     FragoffRes2Mbin[indice] := '1';
     while indice <> 0 do
     begin
       indice := indice - 1;
       FragoffRes2Mbin[indice] := '0';
```

```
      end;
      Copy (FragoffRes2Mbin,0,13,Fragoffbin);
      Copy (FragoffRes2Mbin,13,3,Res2Mbin);
end;

procedure Verificabuffer (identificacao : stringbit48; var item_aux : integer);
  var
   indice : integer;
   indice2 : integer;
   associado : boolean;
   begin
     indice := 0;
     associado := false;
     indice2 := 0;
     while (indice <= N) do
     begin
       if ident_aux[indice] = identificacao then
       begin
 item_aux := indice;
 associado := true;
 indice := 11;
       end
       else
       indice := indice + 1;
     end;
     while (associado = false) do
     begin
       if (ident_aux[indice2] = '') then
begin
  ident_aux[indice2] := identificacao;
  item_aux := indice2;
  associado := true;
  timer[indice2] := 1;
  chegou_algum := chegou_algum + 1;
  WriteString ('chegou_algum = :');
  WriteInt (chegou_algum);
end
       else
       begin
 indice2 := indice2 + 1;
 associado := false;
       end;
     end;
end;

procedure ConvIdintIdbin (IP_PDU_aux4 : IP_PDU_type4; var id_bin : stringbit32);
  var
   indice1     : integer;
   indice2     : integer;
   indice_aux1 : integer;
   indice_aux2 : integer;
   id_bin1     : stringbit16;
   id_bin2     : stringbit16;
   begin
     indice1 := 15;
     while IP_PDU_aux4.Id[0] > 1 do
     begin
       indice_aux1 := IP_PDU_aux4.Id[0] mod 2;
       if indice_aux1 = 1 then id_bin1[indice1] := '1'
       else id_bin1[indice1] := '0';
       IP_PDU_aux4.Id[0] := IP_PDU_aux4.Id[0] div 2;
       indice1 := indice1 - 1;
     end;
     id_bin1[indice1] := '1';
```

```
    while indice1 <> 0 do
    begin
      indice1 := indice1 - 1;
      id_bin1[indice1] := '0';
    end;
    indice2 := 15;
    while IP_PDU_aux4.Id[1] > 1 do
    begin
      indice_aux2 := IP_PDU_aux4.Id[1] mod 2;
      if indice_aux2 = 1 then id_bin2[indice2] := '1'
      else id_bin2[indice2] := '0';
      IP_PDU_aux4.Id[1] := IP_PDU_aux4.Id[1] div 2;
      indice2 := indice2 - 1;
    end;
    id_bin2[indice2] := '1';
    while indice2 <> 0 do
    begin
      indice2 := indice2 - 1;
      id_bin2[indice2] := '0';
    end;
    Concat (id_bin1,id_bin2,id_bin);
   end;

procedure ConvFragoffbintoint (Fragoffbin : stringbit13; var Fragoff : integer);
  var
   aux : array [0..0] of char;
   aux1 : char;
   soma : integer;
   K : integer;
   j : integer;
   begin
    soma := 0;
    K := 4096; (* 2 ** 12 => 13 bits *)
    j := 0;
    while (j <= 12) do
    begin
      aux1 := Fragoffbin[j];
      aux[0] := aux1;
      if aux = '1' then
      soma := soma + K;
      K := K div 2;
      j := j + 1;
    end;
    Fragoff := soma;
   end;

function VerificaTamMsg (SDU_aux1 : SDU_type1) : boolean;
  var
   len_aux : integer;
  begin
   WriteString (' Mensagem : ');
   WriteString (SDU_aux1.SDU);
   WriteLn;
   len := Lenght (SDU_aux1.SDU);
   WriteString (' Tamanho dos dados: ');
   WriteInt (len);
   WriteLn;
   len_aux := len + 40;
   if len_aux > MTU then
    VerificaTamMsg := true
   else
    VerificaTamMsg := false;
  end;
```

```
function VerificaChegada (chegou_algum_aux : integer) : boolean;
begin
  if chegou_algum_aux > 0 then VerificaChegada := true
  else VerificaChegada := false;
end;

procedure VerificaDescarte (identificacao_aux : stringbit48; var descartado_aux : boolean);
var
 i : integer;
begin
  descartado_aux := false;
  i := 0;
  while i <= N do
   begin
     if existe[i] = identificacao then
      begin
descartado_aux := true;
WriteString ('Este Fragmento pertence a um datagrama descartado.');
WriteString ('Descartando Fragmento...');
i := 11;
      end
     else
      i := i + 1;
   end;
end;

procedure Determina_prox_no (SDU_aux1 : SDU_type1 ; campo_org : integer ; campo_dest : integer);
var
 len_aux : integer;
begin
  if campo_dest = campo_org then
   begin
     len := Lenght (SDU_aux1.SDU);
     len_aux := len + 40;
     if len_aux > MTU then
      begin
prox_no_aux := IP_PDU_aux2.DestAddr[7];
WriteString ('A maq destino se localiza na rede local');
Writeln;
WriteString ('prox_no :');
WriteInt (prox_no_aux);
Writeln;
      end
     else
      begin
prox_no_aux := IP_PDU_aux1.DestAddr[7];
WriteString ('A maq destino se localiza na rede local');
Writeln;
WriteString ('prox_no :');
WriteInt (prox_no_aux);
Writeln;
      end;
   end
   else
     WriteString ('A maq destino nao se localiza na rede local');
     Writeln;
     (* prox_no seria um roteador *)
end;

initialize to Repouso
    begin
      id_est := aux_id_est;
      chegou_algum := 0;
      Next2bin := '00000110';
```

```
      Res1bin  := '00000000';
      id_aux := 2;
      for inicio := 0 to N do
       begin
 existe[inicio] := '';
 PayLength_pri[inicio] := 0;
 descartado := false;
 tamanho_dados_pri[inicio] := 0;
 timer[inicio] := 0;
 ident_aux[inicio] := '';
       end;
    end;
trans
  from Repouso
   when UIPCEP.IP_Data_Req
   provided VerificaTamMsg(SDU1)
   to same
   name Frag:
   begin
     IP_PDU_aux2.LLC := 2730;
     IP_PDU_aux2.LLCOUI1 := 768;
     IP_PDU_aux2.OUI2 := 0;
     IP_PDU_aux2.TIPO := 2048;
     IP_PDU_aux2.Id[0] := id_aux;
     IP_PDU_aux2.Id[1] := id_aux;
     id_aux := id_aux + 1;
     WriteString ('IP_PDU_aux2.Id[0] :');
     WriteInt (IP_PDU_aux2.Id[0]);
     WriteLn;
     WriteString ('IP_PDU_aux2.Id[1] :');
     WriteInt (IP_PDU_aux2.Id[1]);
     WriteLn;
     ConvIPhexIPbin(SDU1,ipsrcbin,ipdestbin);
     ConvIPbinIPint(ipsrcbin,sourceaddr2);
     for i := 0 to 7 do
     begin
       IP_PDU_aux2.SourceAddr[i] := sourceaddr2[i];
       WriteString ('SourceAddr :');
       WriteInt (IP_PDU_aux2.SourceAddr[i]);
       WriteLn;
     end;
     ConvIPbinIPint(ipdestbin,destaddr2);
     for i := 0 to 7 do
     begin
       IP_PDU_aux2.DestAddr[i] := destaddr2[i];
       WriteString ('DestAddr :');
       WriteInt (IP_PDU_aux2.DestAddr[i]);
       WriteLn;
     end;
     Determina_prox_no (SDU1,IP_PDU_aux2.SourceAddr[6],IP_PDU_aux2.DestAddr[6]);
     Pr := SDU1.Pr;
     FL := SDU1.FL;
     StoVerPriFlow(Pr,FL,verpriflow2);
     IP_PDU_aux2.VerPriFlow[0] := verpriflow2[0];
     IP_PDU_aux2.VerPriFlow[1] := verpriflow2[1];
     HP := SDU1.HP;
     StoNextHop(SDU1,NextHop_aux);
     IP_PDU_aux2.NextHop := NextHop_aux;
     StoNext2Res1(Next2bin,Res1bin,IP_PDU_aux2);
     M := 1;
     posini := 0;
     Fragoff := 0;
     dados_env := 0;
     NFB := (MTU - 48) div 8;
```

```
     NFB_aux := NFB * 8;
     posfim := NFB_aux;
     while dados_env < len do
      begin
WriteString ('Fragoff: ');
WriteInt (Fragoff);
WriteLn;
IP_PDU_aux2.PayLength := NFB_aux + 8;
Copy (SDU1.SDU,posini,posfim,IP_PDU_aux2.dados);
DetFragoffRes2M (Fragoff,M,IP_PDU_aux2);
output MIPCEP.AAL_Unitdata2_Req (prox_no_aux,IP_PDU_aux2);
Fragoff := Fragoff + NFB;
dados_env := dados_env + NFB_aux;
if dados_env < len then
 begin
   posini := dados_env;
   NFB_aux := len - dados_env;
   if NFB_aux > (NFB * 8) then
    begin
      NFB_aux := NFB * 8;
      M := 1;
    end
   else
    begin
      M := 0;
    end;
   posfim := NFB_aux;
 end
 else
 WriteString ('Acabou a Frag.');
      end;
   end;

   provided NOT(VerificaTamMsg(SDU1))
   to same
   name Datgint:
   begin
     WriteString (' ip da maqlocal: ');
     WriteString (SDU1.ipsrc);
     WriteLn;
     WriteString (' ip da maqremota: ');
     WriteString (SDU1.ipdest);
     WriteLn;
     ConvIPhexIPbin (SDU1,ipsrcbin,ipdestbin);
     ConvIPbinIPint (ipsrcbin,sourceaddr1);
     for i := 0 to 7 do
     begin
       IP_PDU_aux1.SourceAddr[i] := sourceaddr1[i];
     end;
     ConvIPbinIPint (ipdestbin,destaddr1);
     for i := 0 to 7 do
     begin
       IP_PDU_aux1.DestAddr[i] := destaddr1[i];
     end;
     Determina_prox_no (SDU1,IP_PDU_aux1.SourceAddr[6],IP_PDU_aux1.DestAddr[6]);
     Pr := SDU1.Pr;
     FL := SDU1.FL;
     StoVerPriFlow (Pr,FL,verpriflow1);
     IP_PDU_aux1.VerPriFlow[0] := verpriflow1[0];
     IP_PDU_aux1.VerPriFlow[1] := verpriflow1[1];
     IP_PDU_aux1.PayLength := len;
     HP := SDU1.HP;
     StoNextHop(SDU1,NextHop_aux);
     IP_PDU_aux1.NextHop := NextHop_aux;
```

```
   IP_PDU_aux1.dados := SDU1.SDU;
   IP_PDU_aux1.LLC := 2730;
   IP_PDU_aux1.LLCOUI1 := 768;
   IP_PDU_aux1.OUI2 := 0;
   IP_PDU_aux1.TIPO := 2048;
   output MIPCEP.AAL_Unitdata1_Req (prox_no_aux,IP_PDU_aux1);
 end;

from Repouso to Repouso
when MIPCEP.AAL_Unitdata1_Ind
name Recdatg:
begin
  RclVerPriFlow (PDU3.VerPriFlow[0],s);
  if s = '0110' then
  begin
   WriteString ('Versao correta do datagrama. ');
   Writeln;
   if PDU3.DestAddr[7] = id_est then
   begin
     WriteString ('Este datagrama estah corretamente destinado.');
     Writeln;
     WriteString ('id_est :');
     WriteInt (id_est);
     Writeln;
   end
   else
   WriteString ('Este datagrama veio erroneamente para este destino.');
   (* descartando o datagrama *)
   for i := 0 to 7 do
   begin
     sourceaddr1[i] := PDU3.SourceAddr[i];
   end;
   ConvIPintIPbin (sourceaddr1,ipsrcbin);
   for i := 0 to 7 do
   begin
     destaddr1[i] := PDU3.DestAddr[i];
   end;
   ConvIPintIPbin (destaddr1,ipdestbin);
   ConvIPbinIPhex (ipsrcbin,ipdestbin,SDU_aux2);
   SDU_aux2.SDU := PDU3.dados;
   WriteString (' SDU_aux2: ');
   WriteString (SDU_aux2.SDU);
   WriteLn;
   WriteString (SDU_aux2.ipsrc);
   WriteLn;
   WriteString (SDU_aux2.ipdest);
   WriteLn;
   WriteString (SDU_aux2.SDU);
   WriteLn;
   output UIPCEP.IP_Data_Ind (SDU_aux2);
  end
  else
   WriteString ('Versao incorreta do datagrama. ');
   (* descartar datagrama *);
end;

from Repouso
 when MIPCEP.AAL_Unitdata2_Ind
  to Repouso
  name RecFrag:
   begin
     RclVerPriFlow (PDU4.VerPriFlow[0],s);
     WriteLn;
     WriteString (' PDU4.dados: ');
```

```
       WriteString (PDU4.dados);
       WriteLn;
if s = '0110' then
 begin
   WriteString ('Versao correta do Frag.');
   if PDU4.DestAddr[7] = id_est then
   begin
     WriteString ('Este fragmento estah corretamente destinado.');
     Writeln;
     WriteString ('id_est :');
     WriteInt (id_est);
     Writeln;
   end
   else
   WriteString ('Este fragmento nao pertence a este destino.');
   for i := 0 to 7 do
    begin
      sourceaddr2[i] := PDU4.SourceAddr[i];
    end;
   ConvIPintIPbin (sourceaddr2,ipsrcbin);
   for i := 0 to 7 do
    begin
      destaddr2[i] := PDU4.DestAddr[i];
    end;
   ConvIPintIPbin (destaddr2,ipdestbin);
   ConvIPbinIPhex (ipsrcbin,ipdestbin,SDU_aux2);
   ConvIdintIdbin (PDU4,Idbin);
   WriteString ('idbin :');
   WriteString (Idbin);
   WriteLn;
   FragoffRes2M_aux := PDU4.FragoffRes2M;
   RclFragoffRes2M (FragoffRes2M_aux,Fragoffbin,Res2Mbin);
   WriteString ('Fragoffbin :');
   WriteString (Fragoffbin);
   WriteLn;
   WriteString ('Res2Mbin :');
   WriteString (Res2Mbin);
   WriteLn;
   auxMbin_aux := Res2Mbin[2];
   auxMbin[0] := auxMbin_aux;
   Concat (SDU_aux2.ipsrc,SDU_aux2.ipdest,identificacao);
   Concat (identificacao,Idbin,identificacao);
   WriteString ('identificacao :');
   WriteString (identificacao);
   WriteLn;
   VerificaDescarte (identificacao,descartado);
   if descartado = false then
    begin
      Verificabuffer (identificacao,item_aux);
      WriteString ('item_aux :');
      WriteInt (item_aux);
      if ((Fragoffbin = '0000000000000') and (auxMbin = '1')) then
       begin
 WriteString (' identificacao: ');
 WriteString (identificacao);
 WriteString (' ident_aux[item_aux]: ');
 WriteString (ident_aux[item_aux]);
 WriteLn;
 PayLength_pri[item_aux] := PDU4.PayLength;
 tamanho_dados_pri[item_aux] := PDU4.PayLength - 8;
 Copy (PDU4.dados,0,tamanho_dados_pri[item_aux],dadosrec_aux[item_aux]);
 WriteString (' dadosrec_aux[item_aux] :');
 WriteString (dadosrec_aux[item_aux]);
 WriteLn;
```

```
 Delete (identificacao,0,48);
 WriteString (' identificacao deletada: ');
 WriteString (identificacao);
 WriteLn;
       end
      else
       begin
 Concat (dadosrec_aux[item_aux],PDU4.dados,dadosrec_aux[item_aux]);
 if auxMbin = '0' then
  begin
    tamanho_dados_ult[item_aux] := PDU4.PayLength - 8;
    ConvFragoffbintoint (Fragoffbin,Fragoff_aux);
    len_aux := Lenght (dadosrec_aux[item_aux]);
    if (PayLength_pri[item_aux] - tamanho_dados_pri[item_aux] -
    8 + (8 * Fragoff_aux) + tamanho_dados_ult[item_aux]) = len_aux then
     begin
       Copy (dadosrec_aux[item_aux],0,len_aux,SDU_aux2.SDU);
       chegou_algum := chegou_algum - 1;
       output UIPCEP.IP_Data_Ind (SDU_aux2);
       Delete (ident_aux[item_aux],0,48);
       Delete (dadosrec_aux[item_aux],0,50);
       PayLength_pri[item_aux] := 0;
       tamanho_dados_pri[item_aux] := 0;
       tamanho_dados_ult[item_aux] := 0;
       timer[item_aux] := 0;
     end
    else
       WriteString (' Erro. Nem todos os fragmentos foram recebidos.');
       WriteString ('dadosrec :');
       WriteString (dadosrec_aux[item_aux]);
       WriteLn;
  end; (* do auxMbin *)
       end; (* do Fragoffbin = '00..' *)
    end (* do descartado *)
   else
    WriteString ('Mais um fragmento sendo descartado.');
 end (* do s *)
else
 begin
   WriteString (' Versao incorreta do datagrama.');
 end;
     end;

   from Repouso
    provided VerificaChegada (chegou_algum)
    to same
  (*   delay (P) *)
     name DaTempo:
     begin
       for i := 0 to N do
begin
  if timer[i] > 0 then
   begin
     timer[i] := timer[i] + P;
     WriteString ('O valor do timer eh:');
     WriteInt (timer[i]);
     WriteLn;
     if timer[i] >= 60 then
      begin
chegou_algum := chegou_algum - 1;
existe[i] := ident_aux[i];
Delete (ident_aux[i],0,48);
Delete (dadosrec_aux[i],0,50);
timer[i] := 0;
```

```
WriteString ('Tempo estourado para o dat: ');
WriteInt (i);
WriteLn;
WriteString ('Descartando o restante do datagrama.');
      end;
   end;
end;
     end;


end; (* IP_body *)

module User_type activity;
  ip UCEP: IP_Interface (user);
     USYS: SYS_Interface (provider);
end;
body User_body for User_type;
  ip SYS : SYS_Interface (user);
     USER: SYS_Interface (provider);
var
 SDU_aux1 : SDU_type1;
 maqlocal,maqremota : nomemaq;
 mp : maqpos;
 IPpos : enderecopos;
 ipsrc,ipdest : enderecoIPhex;
 FL  : integer;
 Pr  : integer;
 HP  : integer;
 SDU : stringbit50;

state
     S1;
(* Espaco destinado aos procedures e as funcoes necessarias *)

procedure AssociamaqaoIP (nome:nomemaq; var EndIP:enderecoIPhex);
var
  indice:integer;
  associado:boolean;
  begin
   indice:=0;
   associado:=false;
   while (associado=false) do
    begin
     if (mp[indice]=nome) then
       begin
 EndIP:=IPpos[indice];
 associado := true;
       end
     else
 indice := indice+1;
    end;
  end;

procedure AssociaIPamaq (EndIP : enderecoIPhex; var nome : nomemaq);
  var
   indice : integer;
   associado : boolean;
   begin
     indice := 0;
     associado := false;
     while (associado = false) do
     begin
       if IPpos[indice] = EndIP then
       begin
```

```
 nome := mp[indice];
 associado := true;
       end
       else
       indice := indice + 1;
     end;
   end;
(* Fim dos procedures e das funcoes *)
  initialize to S1
   name inicia_usuario:
    begin
     connect SYS to USER;
     mp[0]:= 'buzios';
     mp[1]:= 'recreio';
     mp[2]:= 'alpha' ;
     mp[3]:= 'beta' ;
     mp[4]:= 'parati' ;
     mp[5]:= 'lumiar' ;
     IPpos[0]:= '72A43547' ;
     IPpos[1]:= '72A4355B' ;
     IPpos[2]:= '72A43541' ;
     IPpos[3]:= '72A43542' ;
     IPpos[4]:= '72A43548' ;
     IPpos[5]:= '72A43553' ;
    end;

trans
    when USYS.ENVIAR
    name Iniciotrans:
    begin
       output SYS.ENVIAR;
    end;

trans
    from S1 to S1
    when USER.ENVIAR
    name InicioEnvia:
    begin
      WriteString ('Entre com a maquina local: ');
      ReadString (maqlocal);
      Writeln;
      WriteString ('Entre com a maquina remota: ');
      ReadString (maqremota);
      Writeln;
      WriteString ('Flow-label: ');
      ReadInt (FL);
      Writeln;
      WriteString ('Prioridade: ');
      ReadInt (Pr);
      Writeln;
      WriteString ('Hoplimit: ');
      ReadInt (HP);
      Writeln;
      WriteString ('Entre com os dados: ');
      ReadString (SDU);
      AssociamaqaoIP (maqlocal,ipsrc);
      WriteString ('IP da maqlocal: ');
      AssociamaqaoIP (maqremota,ipdest);
      WriteString ('IP da maqremota: ');
      SDU_aux1.ipsrc := ipsrc;
      SDU_aux1.ipdest := ipdest;
      SDU_aux1.FL := FL;
      SDU_aux1.Pr := Pr;
      SDU_aux1.HP := HP;
```

```
      SDU_aux1.SDU := SDU;
      WriteString (SDU_aux1.ipsrc);
      WriteLn;
      WriteString (SDU_aux1.ipdest);
      WriteLn;
      WriteInt (SDU_aux1.FL);
      WriteLn;
      WriteInt (SDU_aux1.Pr);
      WriteLn;
      WriteInt (SDU_aux1.HP);
      WriteLn;
      WriteString (SDU_aux1.SDU);
      output UCEP.IP_Data_Req (SDU_aux1);
    end;

    from S1 to S1
    when UCEP.IP_Data_Ind
    name RecDat:
    begin
      AssociaIPamaq (SDU2.ipsrc,maqremota);
      AssociaIPamaq (SDU2.ipdest,maqlocal);
      WriteString ('Chegou um datagrama da maquina :');
      WriteString (maqremota);
      WriteLn;
      WriteString ('para a maquina: ');
      WriteString (maqlocal);
      WriteLn;
      WriteString (' com os dados: ');
      WriteString (SDU2.SDU);
    end;

end; (* User_body *)
    modvar
 IP_Recreio, IP_Buzios, IP_Alpha,
 IP_Lumiar, IP_Beta, IP_Parati : IP_type;
 MEIO       : AAL5_type;
 Recreio, Buzios, Alpha,
 Lumiar, Beta, Parati : User_type;

    initialize (* IP_protocol *)
       name iniciaprot:
       begin
init MEIO       with AAL5_body;
init IP_Recreio with IP_body(13659);
init IP_Buzios  with IP_body(13639);
init IP_Alpha   with IP_body(13633);
init IP_Lumiar  with IP_body(13651);
init IP_Beta    with IP_body(13634);
init IP_Parati  with IP_body(13640);
init Recreio    with User_body;
init Buzios     with User_body;
init Alpha      with User_body;
init Lumiar     with User_body;
init Beta       with User_body;
init Parati     with User_body;
connect IP_Recreio.MIPCEP to MEIO.AAL5[0];
connect IP_Buzios.MIPCEP to MEIO.AAL5[1];
connect IP_Alpha.MIPCEP to MEIO.AAL5[2];
connect IP_Lumiar.MIPCEP to MEIO.AAL5[3];
connect IP_Beta.MIPCEP to MEIO.AAL5[4];
connect IP_Parati.MIPCEP to MEIO.AAL5[5];
connect Recreio.UCEP to IP_Recreio.UIPCEP;
connect Buzios.UCEP to IP_Buzios.UIPCEP;
connect Alpha.UCEP to IP_Alpha.UIPCEP;
```

```
 connect Lumiar.UCEP to IP_Lumiar.UIPCEP;
 connect Beta.UCEP to IP_Beta.UIPCEP;
 connect Parati.UCEP to IP_Parati.UIPCEP;
       end;
end.  (* IPv6 protocol *)
```

A.1 Algumas Simulações

A.1.1 Simulação A

O objetivo desta simulação foi transferir um datagrama completo de uma máquina qualquer a outra máquina. Nesta simulação foram escolhidas as máquinas `Búzios` e `Parati`.

```
================================================================
     | BUZIOS[ 0]                | INICIOTRANSaot        |     0
     |                           |-----------------------|
     |  --> SYS[ 0]              | ENVIAR                |
================================================================
     | BUZIOS[ 0]                | INICIOENVIAaov        |     0
     |                           |-----------------------|
     |  --> UCEP[ 0]             | IPDATAREQ             |
================================================================
     | IPBUZIOS[ 0]              | DATGINTanl            |     0
     |                           |-----------------------|
     |  --> MIPCEP[ 0]           | AALUNITDATA1REQ       |
================================================================
     | MEIO[ 0]                  | AAL1ENVIANDODATafp    |     0
     |                           |-----------------------|
     |  --> AAL5[ 5]             | AALUNITDATA1IND       |
================================================================
     | IPPARATI[ 0]              | RECDATGann            |     0
     |                           |-----------------------|
     |  --> UIPCEP[ 0]           | IPDATAIND             |
================================================================
     | PARATI[ 0]                | RECDATaox             |     0
```

A.1.2 Simulação B

Nesta simulação foi realizada a transferência de um datagrama em fragmentos de uma máquina qualquer a outra máquina. Neste caso foram escolhidas as máquinas `Alpha` e `Recreio`. Não houve perda de fragmentos.

```
===========================================================
    | ALPHA[ 0]               | INICIOTRANSaot       |     0
    |                         |----------------------|
    |  --> SYS[ 0]            | ENVIAR               |
===========================================================
    | ALPHA[ 0]               | INICIOENVIAaov       |     0
    |                         |----------------------|
    |  --> UCEP[ 0]           | IPDATAREQ            |
===========================================================
    | IPALPHA[ 0]             | FRAGanj              |     0
    |                         |----------------------|
    |  --> MIPCEP[ 0]         | AALUNITDATA2REQ      |
    |                         |----------------------|
    |  --> MIPCEP[ 0]         | AALUNITDATA2REQ      |
===========================================================
    | MEIO[ 0]                | AAL2ENVIANDOFRAGafv  |     0
    |                         |----------------------|
    |  --> AAL5[ 0]           | AALUNITDATA2IND      |
===========================================================
    | IPRECREIO[ 0]           | RECFRAGanp           |     0
===========================================================
    | MEIO[ 0]                | AAL2ENVIANDOFRAGafv  |     0
    |                         |----------------------|
    |  --> AAL5[ 0]           | AALUNITDATA2IND      |
===========================================================
    | IPRECREIO[ 0]           | RECFRAGanp           |     0
    |                         |----------------------|
    |  --> UIPCEP[ 0]         | IPDATAIND            |
===========================================================
    | RECREIO[ 0]             | RECDATaox            |     0
```

A.1.3 Simulação C

O objetivo desta simulação foi transferir diversos datagramas em fragmentos provenientes de diferentes máquinas, destinados a uma mesma máquina. Neste caso foram escolhidas as máquinas `Lumiar` e `Beta` como máquinas origem de datagramas e a máquina `Recreio` como máquina de destino dos datagramas. Nesta simulação houve a perda do último fragmento proveniente da máquina `Lumiar` no meio de comunicação, resultando no disparo de um *timer*. Os fragmentos anteriormente recebidos referentes à este datagrama são então descartados.

```
===========================================================
    | LUMIAR[ 0]              | INICIOTRANSaot       |     0
    |                         |----------------------|
    |  --> SYS[ 0]            | ENVIAR               |
===========================================================
    | LUMIAR[ 0]              | INICIOENVIAaov       |     0
    |                         |----------------------|
    |  --> UCEP[ 0]           | IPDATAREQ            |
===========================================================
    | IPLUMIAR[ 0]            | FRAGanj              |     0
    |                         |----------------------|
    |  --> MIPCEP[ 0]         | AALUNITDATA2REQ      |
```

```
     |                          |---------------------|
     |  --> MIPCEP[ 0]          | AALUNITDATA2REQ     |
=================================================================
     | MEIO[ 0]                 | AAL3ENVIANDOFRAGafz  |     0
     |                          |---------------------|
     |  --> AAL5[ 0]            | AALUNITDATA2IND     |
=================================================================
     | IPRECREIO[ 0]            | RECFRAGanp          |     0
=================================================================
     | BETA[ 0]                 | INICIOTRANSaot      |     0
     |                          |---------------------|
     |  --> SYS[ 0]             | ENVIAR              |
=================================================================
     | BETA[ 0]                 | INICIOENVIAaov      |     0
     |                          |---------------------|
     |  --> UCEP[ 0]            | IPDATAREQ           |
=================================================================
     | IPBETA[ 0]               | FRAGanj             |     0
     |                          |---------------------|
     |  --> MIPCEP[ 0]          | AALUNITDATA2REQ     |
     |                          |---------------------|
     |  --> MIPCEP[ 0]          | AALUNITDATA2REQ     |
     |                          |---------------------|
     |  --> MIPCEP[ 0]          | AALUNITDATA2REQ     |
=================================================================
     | MEIO[ 0]                 | AAL4ENVIANDOFRAGagd  |     0
     |                          |---------------------|
     |  --> AAL5[ 0]            | AALUNITDATA2IND     |
=================================================================
     | IPRECREIO[ 0]            | RECFRAGanp          |     0
=================================================================
     | MEIO[ 0]                 | AAL3ENVIANDOFRAGafz  |     0
=================================================================
     | MEIO[ 0]                 | AAL4ENVIANDOFRAGagd  |     0
     |                          |---------------------|
     |  --> AAL5[ 0]            | AALUNITDATA2IND     |
=================================================================
     | IPRECREIO[ 0]            | RECFRAGanp          |     0
=================================================================
     | MEIO[ 0]                 | AAL4ENVIANDOFRAGagd  |     0
     |                          |---------------------|
     |  --> AAL5[ 0]            | AALUNITDATA2IND     |
=================================================================
     | IPRECREIO[ 0]            | RECFRAGanp          |     0
     |                          |---------------------|
     |  --> UIPCEP[ 0]          | IPDATAIND           |
=================================================================
     | RECREIO[ 0]              | RECDATaox           |     0
=================================================================
     | IPRECREIO[ 0]            | DATEMPOanr          |     0
=================================================================
     | IPRECREIO[ 0]            | DATEMPOanr          |     0
=================================================================
     | IPRECREIO[ 0]            | DATEMPOanr          |     0
=================================================================
     | IPRECREIO[ 0]            | DATEMPOanr          |     0
=================================================================
     | IPRECREIO[ 0]            | DATEMPOanr          |     0
=================================================================
     | IPRECREIO[ 0]            | DATEMPOanr          |     0
```

A.1.4 Simulação D

Nesta simulação houve a transferência de 3 datagramas, em fragmentos, provenientes de máquinas diferentes destinados a uma mesma máquina. Foram escolhidas as máquinas `Recreio`, `Beta` e `Alpha` como máquinas de origem e a máquina `Búzios` como máquina de destino. Nenhum dos fragmentos foi perdido no meio de comunicação.

```
===============================================================
     | RECREIO[ 0]                | INICIOTRANSaot       |     0
     |                            |----------------------|
     |  --> SYS[ 0]               | ENVIAR               |
===============================================================
     | RECREIO[ 0]                | INICIOENVIAaov       |     0
     |                            |----------------------|
     |  --> UCEP[ 0]              | IPDATAREQ            |
===============================================================
     | IPRECREIO[ 0]              | FRAGanj              |     0
     |                            |----------------------|
     |  --> MIPCEP[ 0]            | AALUNITDATA2REQ      |
     |                            |----------------------|
     |  --> MIPCEP[ 0]            | AALUNITDATA2REQ      |
===============================================================
     | MEIO[ 0]                   | AAL0ENVIANDOFRAGafn  |     0
     |                            |----------------------|
     |  --> AAL5[ 1]              | AALUNITDATA2IND      |
===============================================================
     | IPBUZIOS[ 0]               | RECFRAGanp           |     0
===============================================================
     | BETA[ 0]                   | INICIOTRANSaot       |     0
     |                            |----------------------|
     |  --> SYS[ 0]               | ENVIAR               |
===============================================================
     | BETA[ 0]                   | INICIOENVIAaov       |     0
     |                            |----------------------|
     |  --> UCEP[ 0]              | IPDATAREQ            |
===============================================================
     | IPBETA[ 0]                 | FRAGanj              |     0
     |                            |----------------------|
     |  --> MIPCEP[ 0]            | AALUNITDATA2REQ      |
     |                            |----------------------|
     |  --> MIPCEP[ 0]            | AALUNITDATA2REQ      |
     |                            |----------------------|
     |  --> MIPCEP[ 0]            | AALUNITDATA2REQ      |
===============================================================
     | MEIO[ 0]                   | AAL4ENVIANDOFRAGagd  |     0
     |                            |----------------------|
     |  --> AAL5[ 1]              | AALUNITDATA2IND      |
===============================================================
     | IPBUZIOS[ 0]               | RECFRAGanp           |     0
===============================================================
     | ALPHA[ 0]                  | INICIOTRANSaot       |     0
     |                            |----------------------|
     |  --> SYS[ 0]               | ENVIAR               |
===============================================================
     | ALPHA[ 0]                  | INICIOENVIAaov       |     0
     |                            |----------------------|
     |  --> UCEP[ 0]              | IPDATAREQ            |
```

```
===============================================================
     | IPALPHA[ 0]              | FRAGanj              |     0
     |                          |----------------------|
     |  --> MIPCEP[ 0]          | AALUNITDATA2REQ      |
     |                          |----------------------|
     |  --> MIPCEP[ 0]          | AALUNITDATA2REQ      |
     |                          |----------------------|
     |  --> MIPCEP[ 0]          | AALUNITDATA2REQ      |
===============================================================
     | MEIO[ 0]                 | AAL2ENVIANDOFRAGafv  |     0
     |                          |----------------------|
     |  --> AAL5[ 1]            | AALUNITDATA2IND      |
===============================================================
     | IPBUZIOS[ 0]             | RECFRAGanp           |     0
===============================================================
     | MEIO[ 0]                 | AAL4ENVIANDOFRAGagd  |     0
     |                          |----------------------|
     |  --> AAL5[ 1]            | AALUNITDATA2IND      |
===============================================================
     | IPBUZIOS[ 0]             | RECFRAGanp           |     0
===============================================================
     | MEIO[ 0]                 | AAL2ENVIANDOFRAGafv  |     0
     |                          |----------------------|
     |  --> AAL5[ 1]            | AALUNITDATA2IND      |
===============================================================
     | IPBUZIOS[ 0]             | RECFRAGanp           |     0
===============================================================
     | MEIO[ 0]                 | AAL0ENVIANDOFRAGafn  |     0
     |                          |----------------------|
     |  --> AAL5[ 1]            | AALUNITDATA2IND      |
===============================================================
     | IPBUZIOS[ 0]             | RECFRAGanp           |     0
     |                          |----------------------|
     |  --> UIPCEP[ 0]          | IPDATAIND            |
===============================================================
     | BUZIOS[ 0]               | RECDATaox            |     0
===============================================================
     | MEIO[ 0]                 | AAL2ENVIANDOFRAGafv  |     0
     |                          |----------------------|
     |  --> AAL5[ 1]            | AALUNITDATA2IND      |
===============================================================
     | IPBUZIOS[ 0]             | RECFRAGanp           |     0
     |                          |----------------------|
     |  --> UIPCEP[ 0]          | IPDATAIND            |
===============================================================
     | BUZIOS[ 0]               | RECDATaox            |     0
===============================================================
     | MEIO[ 0]                 | AAL4ENVIANDOFRAGagd  |     0
     |                          |----------------------|
     |  --> AAL5[ 1]            | AALUNITDATA2IND      |
===============================================================
     | IPBUZIOS[ 0]             | RECFRAGanp           |     0
     |                          |----------------------|
     |  --> UIPCEP[ 0]          | IPDATAIND            |
===============================================================
     | BUZIOS[ 0]               | RECDATaox            |     0
```

Apêndice B

Descrições Comportamentais em VHDL do Protocolo IPv6

Neste apêndice encontram-se as descrições comportamentais do protocolo IPv6 envolvendo a transferência de datagramas completos bem como a transferência de datagramas completos e fragmentados. A síntese lógica foi realizada apenas na descrição comportamental do protocolo IPv6 que envolve a transferência de datagramas completos.

B.1 Transferência de datagramas completos

Nesta seção encontra-se a descrição comportamental do protocolo IPv6 com os *buffers* para o armazenamento de dados bem como sua descrição comportamental sem os *buffers*. Além disso, a descrição comportamental que abrange apenas os *buffers* para o armazenamento de dados é apresentada.

B.1.1 Descrição com os *Buffers*

```
-- modulo IPv6sub escrito em VHDL.
-- Nesta versao sao transferidos datagramas completos.

LIBRARY std;
  use std.standard.all;

LIBRARY work;
  use work.processador_types.all;


ENTITY IPv6_ENTITY_TYPE IS
  port (start: in bit;
        clk: in bit;
        --   Sinais de Controle de modo a deixar a entidade IP
        --   habilitada a receber os dados do testbench
        ipv6preparado : out bit;
        testpreparado : in bit;
        --   Sinais de controle de modo a deixar a entidade AAL5
        --   apta a receber os dados do IPv6.
        ipv6habilitado : out bit;
        aal5preparado : in bit;
        --   AAL5_Unitdata_Req
        AAL5_Unitdata_Req_cod : out bit;
        AAL5_Unitdata_Req_version : out bit_4;
        AAL5_Unitdata_Req_priority : out bit_4;
        AAL5_Unitdata_Req_flowlabel : out byte_3;
        AAL5_Unitdata_Req_payload_length : out integer;
        AAL5_Unitdata_Req_nextheader : out byte;
        AAL5_Unitdata_Req_hoplimit : out byte;
        AAL5_Unitdata_Req_dados : out character;
        AAL5_Unitdata_Req_aal5type_LLC : out byte_3;
        AAL5_Unitdata_Req_aal5type_OUI1 : out byte;
        AAL5_Unitdata_Req_aal5type_OUI2 : out  byte_2;
        AAL5_Unitdata_Req_aal5type_TIPO : out byte_2;
        AAL5_Unitdata_Req_sourceaddr : out end_ipv6;
        AAL5_Unitdata_Req_destaddr : out end_ipv6;
        AAL5_Unitdata_Req_tam : out integer;
        --   VarInd
        varind_ackout1 : in bit;
        varind_ackout2 : in bit;
        varind_validout1 : out bit;
        varind_validout2 : out bit;
        --   Ready
        ready_validin1 : in bit;
        ready_validin2 : in bit;
        ready_ackin1 : out bit;
        ready_ackin2 : out bit;
        --   AAL5_Unitdata_Ind
        AAL5_Unitdata_Ind_cod : in bit;
        AAL5_Unitdata_Ind_version : in bit_4;
        AAL5_Unitdata_Ind_priority : in bit_4;
        AAL5_Unitdata_Ind_flowlabel : in byte_3;
        AAL5_Unitdata_Ind_payload_length : in integer;
        AAL5_Unitdata_Ind_nextheader : in byte;
        AAL5_Unitdata_Ind_hoplimit : in byte;
        AAL5_Unitdata_Ind_dados : in character;
        AAL5_Unitdata_Ind_sourceaddr : in end_ipv6;
        AAL5_Unitdata_Ind_destaddr : in end_ipv6;
        -- AAL5_Unitdata_Ind_tam : in integer;
        -- Esta primitiva nao precisa ter o tam, pois
        -- o tamanho e dado pelo campo do cabecalho IP.
        --   IPv6_Data_Req
```

```
        IPv6_Data_Req_cod : in bit;
        IPv6_Data_Req_ipsrc : in end_ipv6;
        IPv6_Data_Req_ipdest : in end_ipv6;
        IPv6_Data_Req_flowlabel : in byte_3;
        IPv6_Data_Req_priority : in bit_4;
        IPv6_Data_Req_hoplimit : in byte;
        IPv6_Data_Req_sdu : in character;
        IPv6_Data_Req_tam : in integer;
        --   IPv6_Data_Ind
        IPv6_Data_Ind_cod : out bit;
        IPv6_Data_Ind_ipsrc : out end_ipv6;
        IPv6_Data_Ind_ipdest : out end_ipv6;
        IPv6_Data_Ind_sdu : out character);

END IPv6_ENTITY_TYPE;

ARCHITECTURE IPv6_ENTITY_BODY OF IPv6_ENTITY_TYPE IS

  TYPE tam_msg_usu IS array (0 to 34) of character;


BEGIN  --  IPv6_ENTITY_BODY
 -----------------------------------------------------------------------------
 --  Process:
 --  Purpose:
 --  Inputs:
 --  Outputs:
 -----------------------------------------------------------------------------
 PROCESS

  VARIABLE      buffer_LLC       : byte_3;
  VARIABLE      buffer_OUI1      : byte;
  VARIABLE      buffer_OUI2, buffer_TIPO         : byte_2;
  VARIABLE      i, sdu_usu, aal5_pdu     : integer;
  VARIABLE      buffer_version, buffer_priority  : bit_4;
  VARIABLE      buffer_flowlabel         : byte_3;
  VARIABLE      buffer_payload_length    : integer;
  VARIABLE      buffer_usuipv6_sdu, buffer_aal5ipv6_pdu : tam_msg_usu;
  VARIABLE      buffer_hoplimit, buffer_nextheader       : byte;
  VARIABLE      buffer_sourceaddr, buffer_destaddr  : end_ipv6;


  PROCEDURE LermsgUserIPv6 is
  begin
    i := 0;
    lendomsgusu : WHILE (i < sdu_usu) LOOP
    WAIT UNTIL (ready_validin1 = '1');
    buffer_usuipv6_sdu(i) := IPv6_Data_Req_sdu;
    ready_ackin1 <= '1';
    WAIT UNTIL (ready_validin1 = '0');
    ready_ackin1 <= '0';
    i := i + 1;
    END LOOP lendomsgusu;
    end LermsgUserIPv6;

  PROCEDURE EnviarmsgIPv6aal5 is
  begin
    i := 0;
    enviandomsgaal5 : WHILE (i < sdu_usu) LOOP
    IF (varind_ackout1 /= '0') THEN
    WAIT UNTIL varind_ackout1 = '0';
    END IF;
    AAL5_Unitdata_Req_dados <= buffer_usuipv6_sdu(i);
    varind_validout1 <= '1';
```

```
  WAIT UNTIL (varind_ackout1 = '1');
  varind_validout1 <= '0';
  i := i + 1;
  END LOOP enviandomsgaal5;
  end EnviarmsgIPv6aal5;

PROCEDURE Lermsgaal5IPv6 is
  begin
    i := 0;
    lendomsgaal5 : WHILE (i < aal5_pdu) LOOP
    WAIT UNTIL (ready_validin2 = '1');
    buffer_aal5ipv6_pdu(i) := AAL5_Unitdata_Ind_dados;
    ready_ackin2 <= '1';
    WAIT UNTIL (ready_validin2 = '0');
    ready_ackin2 <= '0';
    i := i + 1;
    END LOOP lendomsgaal5;
  end Lermsgaal5IPv6;

PROCEDURE EnviarmsgIPv6User is
  begin
    i := 0;
    enviandomsguser : WHILE (i < aal5_pdu) LOOP
    IF ((varind_ackout2 /= '0')) THEN
      WAIT UNTIL varind_ackout2 = '0';
    END IF;
    IPv6_Data_Ind_sdu <= buffer_aal5ipv6_pdu(i);
    varind_validout2 <= '1';
    WAIT UNTIL (varind_ackout2 = '1');
    varind_validout2 <= '0';
    i := i + 1;
    END LOOP enviandomsguser;
  end EnviarmsgIPv6User;

PROCEDURE inicio is
  begin
    varind_validout1 <= '0';
    varind_validout2 <= '0';
    ready_ackin1 <= '0';
    ready_ackin2 <= '0';
    ipv6habilitado <= '0';
  end inicio;


  begin --   process
    IPv6_Data_Ind_cod <= '0';
    AAL5_Unitdata_Req_cod <= '0';
    ipv6preparado <= '0';
    WAIT UNTIL (start = '1' and clk'event and clk = '1');
    MEF : LOOP
    ipv6preparado <= '1';
    esperandoativacao : WHILE (IPv6_Data_Req_cod /= '1' AND AAL5_Unitdata_Ind_cod /= '1') LOOP
    WAIT UNTIL (clk'event and clk = '1');
    END LOOP esperandoativacao;
    ipv6preparado <= '0';
    inicio;
    IF ((IPv6_Data_Req_cod = '1')) THEN
      sdu_usu := IPv6_Data_Req_tam;
      buffer_version := "0110";
      buffer_LLC := "101010101010101000000011";
      buffer_OUI1 := "00000000";
      buffer_OUI2 := "0000000000000000";
      buffer_TIPO := "1000011011011101";
      buffer_priority := IPv6_Data_Req_priority;
```

```
      buffer_flowlabel := IPv6_Data_Req_flowlabel;
      buffer_hoplimit := IPv6_Data_Req_hoplimit;
      buffer_sourceaddr := IPv6_Data_Req_ipsrc;
      buffer_destaddr := IPv6_Data_Req_ipdest;
      LermsgUserIPv6;
      AAL5_Unitdata_Req_cod <= '1';
      AAL5_Unitdata_Req_tam <= sdu_usu;

      WAIT UNTIL aal5preparado = '1';

      buffer_nextheader := "00000110";
      buffer_payload_length := sdu_usu;
      AAL5_Unitdata_Req_aal5type_LLC <= buffer_LLC;
      AAL5_Unitdata_Req_aal5type_OUI1 <= buffer_OUI1;
      AAL5_Unitdata_Req_aal5type_OUI2 <= buffer_OUI2;
      AAL5_Unitdata_Req_aal5type_TIPO <= buffer_TIPO;
      AAL5_Unitdata_Req_version <= buffer_version;
      AAL5_Unitdata_Req_priority <= buffer_priority;
      AAL5_Unitdata_Req_flowlabel <= buffer_flowlabel;
      AAL5_Unitdata_Req_payload_length <= buffer_payload_length;
      AAL5_Unitdata_Req_nextheader <= buffer_nextheader;
      AAL5_Unitdata_Req_hoplimit <= buffer_hoplimit;
      AAL5_Unitdata_Req_sourceaddr <= buffer_sourceaddr;
      AAL5_Unitdata_Req_destaddr <= buffer_destaddr;
      EnviarmsgIPv6aal5;
    ELSIF ((AAL5_Unitdata_Ind_cod = '1')) THEN
        ipv6habilitado <= '1';
        buffer_version := AAL5_Unitdata_Ind_version;
        IF ((buffer_version = "0110")) THEN
           buffer_priority := AAL5_Unitdata_Ind_priority;
           buffer_flowlabel := AAL5_Unitdata_Ind_flowlabel;
           buffer_payload_length := AAL5_Unitdata_Ind_payload_length;
           buffer_hoplimit := AAL5_Unitdata_Ind_hoplimit;
           buffer_nextheader := AAL5_Unitdata_Ind_nextheader;
           aal5_pdu := buffer_payload_length;
           buffer_sourceaddr := AAL5_Unitdata_Ind_sourceaddr;
           buffer_destaddr := AAL5_Unitdata_Ind_destaddr;
           Lermsgaal5IPv6;
           IPv6_Data_Ind_cod <= '1';
           WAIT UNTIL testpreparado = '1';
           IPv6_Data_Ind_ipsrc <= buffer_sourceaddr;
           IPv6_Data_Ind_ipdest <= buffer_destaddr;
           EnviarmsgIPv6User;
        END IF;
    END IF;
  WAIT UNTIL (clk'event and clk = '1');
  IPv6_Data_Ind_cod <= '0';
  AAL5_Unitdata_Req_cod <= '0';
  ipv6habilitado <= '0';
  ipv6preparado <= '0';
  END LOOP MEF;
 end process;
END IPv6_ENTITY_BODY;
```

B.1.2 Descrição sem os *Buffers*

```
-- modulo IPv6sub escrito em VHDL.
-- Nesta versao sao transferidos datagramas completos.
-- Os buffers ficaram em outra descricao.

LIBRARY std;
  use std.standard.all;

LIBRARY work;
  use work.processador_types.all;


ENTITY IPv6_ENTITY_SEMBUFFER_TYPE IS
  port (start: in bit;
        clk: in bit;
        --   Sinais de Controle de modo a deixar a entidade IP
        --   habilitada a receber os dados do testbench
        ipv6preparado : out bit;
        testpreparado : in bit;
        --   Sinais de controle de modo a deixar a entidade AAL5
        --   apta a receber os dados do IPv6.
        ipv6habilitado : out bit;
        aal5preparado : in bit;
        --   AAL5_Unitdata_Req
        AAL5_Unitdata_Req_cod : out bit;
        AAL5_Unitdata_Req_version : out bit_4;
        AAL5_Unitdata_Req_priority : out bit_4;
        AAL5_Unitdata_Req_flowlabel : out byte_3;
        AAL5_Unitdata_Req_payload_length : out integer;
        AAL5_Unitdata_Req_nextheader : out byte;
        AAL5_Unitdata_Req_hoplimit : out byte;
        AAL5_Unitdata_Req_dados : out character;
        AAL5_Unitdata_Req_aal5type_LLC : out byte_3;
        AAL5_Unitdata_Req_aal5type_OUI1 : out byte;
        AAL5_Unitdata_Req_aal5type_OUI2 : out  byte_2;
        AAL5_Unitdata_Req_aal5type_TIPO : out byte_2;
        AAL5_Unitdata_Req_sourceaddr : out end_ipv6;
        AAL5_Unitdata_Req_destaddr : out end_ipv6;
        AAL5_Unitdata_Req_tam : out integer;
        --   VarInd
        varind_ackout1 : in bit;
        varind_ackout2 : in bit;
        varind_validout1 : out bit;
        varind_validout2 : out bit;
        --   Ready
        ready_validin1 : in bit;
        ready_validin2 : in bit;
        ready_ackin1 : out bit;
        ready_ackin2 : out bit;
        --   AAL5_Unitdata_Ind
        AAL5_Unitdata_Ind_cod : in bit;
        AAL5_Unitdata_Ind_version : in bit_4;
        AAL5_Unitdata_Ind_priority : in bit_4;
        AAL5_Unitdata_Ind_flowlabel : in byte_3;
        AAL5_Unitdata_Ind_payload_length : in integer;
        AAL5_Unitdata_Ind_nextheader : in byte;
        AAL5_Unitdata_Ind_hoplimit : in byte;
        AAL5_Unitdata_Ind_dados : in character;
        AAL5_Unitdata_Ind_sourceaddr : in end_ipv6;
        AAL5_Unitdata_Ind_destaddr : in end_ipv6;
        -- AAL5_Unitdata_Ind_tam : in integer;
        -- Esta primitiva nao precisa ter o tam, pois
        -- o tamanho e dado pelo campo do cabecalho IP.
```

```
        --   IPv6_Data_Req
        IPv6_Data_Req_cod : in bit;
        IPv6_Data_Req_ipsrc : in end_ipv6;
        IPv6_Data_Req_ipdest : in end_ipv6;
        IPv6_Data_Req_flowlabel : in byte_3;
        IPv6_Data_Req_priority : in bit_4;
        IPv6_Data_Req_hoplimit : in byte;
        IPv6_Data_Req_sdu : in character;
        IPv6_Data_Req_tam : in integer;
        --   IPv6_Data_Ind
        IPv6_Data_Ind_cod : out bit;
        IPv6_Data_Ind_ipsrc : out end_ipv6;
        IPv6_Data_Ind_ipdest : out end_ipv6;
        IPv6_Data_Ind_sdu : out character;
        -- Sinais adicionados para a comunic. c/ memoria
        usu_envia_mem : out character;
        usu_envia_mem_cod : out bit;
        ipv6_envia_aal5_mem : in character;
        ipv6_envia_aal5_mem_cod : out bit;
        ipv6_recebe_aal5_mem : out character;
        ipv6_recebe_aal5_mem_cod : out bit;
        ipv6_envia_usu_mem : in character;
        ipv6_envia_usu_mem_cod : out bit;
        mem_autorizando : in bit;
        mem_autorizando_usu : in bit);


END IPv6_ENTITY_SEMBUFFER_TYPE;

ARCHITECTURE IPv6_ENTITY_SEMBUFFER_BODY OF IPv6_ENTITY_SEMBUFFER_TYPE IS

  -- TYPE tam_msg_usu IS array (0 to 34) of character;


BEGIN  --  IPv6_ENTITY_BODY
 -------------------------------------------------------------------------------
 --  Process:
 --  Purpose:
 --  Inputs:
 --  Outputs:
 -------------------------------------------------------------------------------
 PROCESS

  VARIABLE      buffer_LLC       : byte_3;
  VARIABLE      buffer_OUI1      : byte;
  VARIABLE      buffer_OUI2, buffer_TIPO         : byte_2;
  VARIABLE      i, sdu_usu, aal5_pdu     : integer;
  VARIABLE      buffer_version, buffer_priority  : bit_4;
  VARIABLE      buffer_flowlabel         : byte_3;
  VARIABLE      buffer_payload_length    : integer;
  VARIABLE      buffer_hoplimit, buffer_nextheader       : byte;
  VARIABLE      buffer_sourceaddr, buffer_destaddr  : end_ipv6;


  PROCEDURE LermsgUserIPv6 is
  begin
    i := 0;
    lendomsgusu : WHILE (i < sdu_usu) LOOP
    WAIT UNTIL (ready_validin1 = '1');
    usu_envia_mem_cod <= '1';
    usu_envia_mem <= IPv6_Data_Req_sdu;
    ready_ackin1 <= '1';
    WAIT UNTIL (ready_validin1 = '0');
    usu_envia_mem_cod <= '0';
```

```
  ready_ackin1 <= '0';
  i := i + 1;
  END LOOP lendomsgusu;
  end LermsgUserIPv6;

PROCEDURE EnviarmsgIPv6aal5 is
begin
  i := 0;
  enviandomsgaal5 : WHILE (i < sdu_usu) LOOP
  ipv6_envia_aal5_mem_cod <= '1';
  WAIT UNTIL mem_autorizando = '1';
  IF (varind_ackout1 /= '0') THEN
    WAIT UNTIL varind_ackout1 = '0';
  END IF;
  AAL5_Unitdata_Req_dados <= ipv6_envia_aal5_mem;
  ipv6_envia_aal5_mem_cod <= '0';
  WAIT UNTIL mem_autorizando = '0';
  varind_validout1 <= '1';
  WAIT UNTIL (varind_ackout1 = '1');
  varind_validout1 <= '0';
  i := i + 1;
  END LOOP enviandomsgaal5;
  end EnviarmsgIPv6aal5;

PROCEDURE Lermsgaal5IPv6 is
  begin
    i := 0;
    lendomsgaal5 : WHILE (i < aal5_pdu) LOOP
    WAIT UNTIL (ready_validin2 = '1');
    ipv6_recebe_aal5_mem_cod <= '1';
    ipv6_recebe_aal5_mem <= AAL5_Unitdata_Ind_dados;
    ready_ackin2 <= '1';
    WAIT UNTIL (ready_validin2 = '0');
    ipv6_recebe_aal5_mem_cod <= '0';
    ready_ackin2 <= '0';
    i := i + 1;
    END LOOP lendomsgaal5;
  end Lermsgaal5IPv6;

PROCEDURE EnviarmsgIPv6User is
  begin
    i := 0;
    enviandomsguser : WHILE (i < aal5_pdu) LOOP
    ipv6_envia_usu_mem_cod <= '1';
    WAIT UNTIL mem_autorizando_usu = '1';
    IF ((varind_ackout2 /= '0')) THEN
      WAIT UNTIL varind_ackout2 = '0';
    END IF;
    IPv6_Data_Ind_sdu <= ipv6_envia_usu_mem;
    ipv6_envia_usu_mem_cod <= '0';
    WAIT UNTIL mem_autorizando_usu = '0';
    varind_validout2 <= '1';
    WAIT UNTIL (varind_ackout2 = '1');
    varind_validout2 <= '0';
    i := i + 1;
    END LOOP enviandomsguser;
  end EnviarmsgIPv6User;

PROCEDURE inicio is
  begin
    varind_validout1 <= '0';
    varind_validout2 <= '0';
    ready_ackin1 <= '0';
    ready_ackin2 <= '0';
```

```
    ipv6habilitado <= '0';
  end inicio;


  begin --   process
    IPv6_Data_Ind_cod <= '0';
    AAL5_Unitdata_Req_cod <= '0';
    ipv6preparado <= '0';
    usu_envia_mem_cod <= '0';
    ipv6_envia_aal5_mem_cod <= '0';
    ipv6_recebe_aal5_mem_cod <= '0';
    ipv6_envia_usu_mem_cod <= '0';

    WAIT UNTIL (start = '1' and clk'event and clk = '1');
    MEF : LOOP
    ipv6preparado <= '1';
    esperandoativacao : WHILE (IPv6_Data_Req_cod /= '1' AND AAL5_Unitdata_Ind_cod /= '1') LOOP
    WAIT UNTIL (clk'event and clk = '1');
    END LOOP esperandoativacao;
    ipv6preparado <= '0';
    inicio;
    IF ((IPv6_Data_Req_cod = '1')) THEN
      sdu_usu := IPv6_Data_Req_tam;
      buffer_version := "0110";
      buffer_LLC := "101010101010101000000011";
      buffer_OUI1 := "00000000";
      buffer_OUI2 := "0000000000000000";
      buffer_TIPO := "1000011011011101";
      buffer_priority := IPv6_Data_Req_priority;
      buffer_flowlabel := IPv6_Data_Req_flowlabel;
      buffer_hoplimit := IPv6_Data_Req_hoplimit;
      buffer_sourceaddr := IPv6_Data_Req_ipsrc;
      buffer_destaddr := IPv6_Data_Req_ipdest;
      LermsgUserIPv6;
      AAL5_Unitdata_Req_cod <= '1';
      AAL5_Unitdata_Req_tam <= sdu_usu;

      WAIT UNTIL aal5preparado = '1';

      buffer_nextheader := "00000110";
      buffer_payload_length := sdu_usu;
      AAL5_Unitdata_Req_aal5type_LLC <= buffer_LLC;
      AAL5_Unitdata_Req_aal5type_OUI1 <= buffer_OUI1;
      AAL5_Unitdata_Req_aal5type_OUI2 <= buffer_OUI2;
      AAL5_Unitdata_Req_aal5type_TIPO <= buffer_TIPO;
      AAL5_Unitdata_Req_version <= buffer_version;
      AAL5_Unitdata_Req_priority <= buffer_priority;
      AAL5_Unitdata_Req_flowlabel <= buffer_flowlabel;
      AAL5_Unitdata_Req_payload_length <= buffer_payload_length;
      AAL5_Unitdata_Req_nextheader <= buffer_nextheader;
      AAL5_Unitdata_Req_hoplimit <= buffer_hoplimit;
      AAL5_Unitdata_Req_sourceaddr <= buffer_sourceaddr;
      AAL5_Unitdata_Req_destaddr <= buffer_destaddr;
      EnviarmsgIPv6aal5;
    ELSIF ((AAL5_Unitdata_Ind_cod = '1')) THEN
        ipv6habilitado <= '1';
        buffer_version := AAL5_Unitdata_Ind_version;
        IF ((buffer_version = "0110")) THEN
           buffer_priority := AAL5_Unitdata_Ind_priority;
           buffer_flowlabel := AAL5_Unitdata_Ind_flowlabel;
           buffer_payload_length := AAL5_Unitdata_Ind_payload_length;
           buffer_hoplimit := AAL5_Unitdata_Ind_hoplimit;
           buffer_nextheader := AAL5_Unitdata_Ind_nextheader;
           aal5_pdu := buffer_payload_length;
```

```
          buffer_sourceaddr := AAL5_Unitdata_Ind_sourceaddr;
          buffer_destaddr := AAL5_Unitdata_Ind_destaddr;
          Lermsgaal5IPv6;
          IPv6_Data_Ind_cod <= '1';
          WAIT UNTIL testpreparado = '1';
          IPv6_Data_Ind_ipsrc <= buffer_sourceaddr;
          IPv6_Data_Ind_ipdest <= buffer_destaddr;
          EnviarmsgIPv6User;
        END IF;
      END IF;
    WAIT UNTIL (clk'event and clk = '1');
    IPv6_Data_Ind_cod <= '0';
    AAL5_Unitdata_Req_cod <= '0';
    ipv6habilitado <= '0';
    ipv6preparado <= '0';
    END LOOP MEF;
   end process;
  END IPv6_ENTITY_SEMBUFFER_BODY;
```

B.1.3 Descrição Comportamental abrangendo apenas os *Buffers*

```
-- modulo de memoria. Neste modulo
-- estao contidos os buffers necessarios
-- que foram retirados das entidades ipv6.
-- Data: 05/03/98

LIBRARY std;
    use std.standard.all;

LIBRARY work;
    use work.processador_types.all;

ENTITY MEMORIA_ENTITY_TYPE IS
  port (start : in bit;
        clk : in bit;
        -- Sinais para a comunicacao c/ o IPv6
        mem_rec_usu : in character;
        mem_rec_usu_cod : in bit;
        mem_envia_aal5 : out character;
        mem_envia_aal5_cod : in bit;
        mem_recebe_aal5 : in character;
        mem_recebe_aal5_cod : in bit;
        mem_envia_usu : out character;
        mem_envia_usu_cod : in bit;
        mem_enviando_aal5 : out bit;
        mem_enviando_usu : out bit;
        -- Sinais para o sincronismo c/ o IPv6
        IPv6_Data_Req_cod : in bit;
        AAL5_Unitdata_Ind_cod : in bit;
        IPv6_Data_Req_tam : in integer;
        AAL5_Unitdata_Ind_payload_length : in integer);

END MEMORIA_ENTITY_TYPE;

ARCHITECTURE MEMORIA_ENTITY_BODY OF MEMORIA_ENTITY_TYPE IS

     TYPE tam_msg_usu is array (0 to 34) of character;
```

```
begin -- MEMORIA_ENTITY_BODY

   process

      VARIABLE buffer_usuipv6_sdu, buffer_aal5ipv6_pdu : tam_msg_usu;
      VARIABLE cont_pos, i, cont_pdu : integer;

begin

      i := 0;
      mem_enviando_aal5 <= '0';
      mem_enviando_usu <= '0';
      WAIT UNTIL (start = '1' AND clk'event AND clk = '1');
      MEF : LOOP
      esperandoativacao: WHILE (IPv6_Data_Req_cod /= '1' AND AAL5_Unitdata_Ind_cod /= '1') LOOP
      WAIT UNTIL (clk'event AND clk = '1');
      END LOOP esperandoativacao;
      i := 0;
      IF (IPv6_Data_Req_cod = '1') THEN
        cont_pos := IPv6_Data_Req_tam;
        recebendomsgusu: WHILE (i < cont_pos) LOOP
        WAIT UNTIL mem_rec_usu_cod = '1';
        buffer_usuipv6_sdu(i) := mem_rec_usu;
        i := i + 1;
        WAIT UNTIL mem_rec_usu_cod = '0';
        END LOOP recebendomsgusu;
        i := 0;

        enviandomsgaal5 : WHILE (i < cont_pos) LOOP
        WAIT UNTIL mem_envia_aal5_cod = '1';
        mem_envia_aal5 <= buffer_usuipv6_sdu(i);
        mem_enviando_aal5 <= '1';
        i := i + 1;
        WAIT UNTIL mem_envia_aal5_cod = '0';
        mem_enviando_aal5 <= '0';
        END LOOP enviandomsgaal5;
      ELSIF (AAL5_Unitdata_Ind_cod = '1') THEN
        cont_pdu := AAL5_Unitdata_Ind_payload_length;
        recebendomsgaal5 : WHILE (i < cont_pdu) LOOP
        WAIT UNTIL mem_recebe_aal5_cod = '1';
        buffer_aal5ipv6_pdu(i) := mem_recebe_aal5;
        i := i + 1;
        WAIT UNTIL mem_recebe_aal5_cod = '0';
        END LOOP recebendomsgaal5;

        i := 0;
        enviandomsgusu : WHILE (i < cont_pdu) LOOP
        WAIT UNTIL mem_envia_usu_cod = '1';
        mem_envia_usu <= buffer_aal5ipv6_pdu(i);
        mem_enviando_usu <= '1';
        i := i + 1;
        WAIT UNTIL mem_envia_usu_cod = '0';
        mem_enviando_usu <= '0';
        END LOOP enviandomsgusu;
      END IF;
     WAIT UNTIL (clk'event AND clk = '1');
    END LOOP MEF;
  END process;
END MEMORIA_ENTITY_BODY;
```

B.2 Transferência de datagramas completos e fragmentados

```
-- modulo IPv6 escrito em VHDL.
-- Esta eh a versao com a fragmentacao.
-- Os sinais de Fragoffset e de identificacao foram
-- colocados como tipo inteiro (integer) pois
-- serao feitas operacoes aritmeticas com estes valores.
-- O procedimento de adicao (add) pode nao ser sintetizavel.


LIBRARY std;
  use std.standard.all;

LIBRARY work;
  use work.processador_types.all;


ENTITY IPv6_ENTITYFRAG_TYPE IS
  port (start: in bit;
        clk: in bit;
        --   Sinais de Controle de modo a deixar a entidade IP
        --   habilitada a receber os dados do testbench
        ipv6preparado : out bit;
        testpreparado : in bit;
        --   Sinais de controle de modo a deixar a entidade AAL5
        --   apta a receber os dados do IPv6.
        ipv6habilitado : out bit;
        aal5preparado : in bit;
        --   AAL5_Unitdata_Req
        AAL5_Unitdata_Req_cod : out bit;
        AAL5_Unitdata_Req_version : out bit_4;
        AAL5_Unitdata_Req_priority : out bit_4;
        AAL5_Unitdata_Req_flowlabel : out byte_3;
        AAL5_Unitdata_Req_payload_length : out integer;
        AAL5_Unitdata_Req_nextheader : out byte;
        AAL5_Unitdata_Req_hoplimit : out byte;
        AAL5_Unitdata_Req_dados : out character;
        AAL5_Unitdata_Req_aal5type_LLC : out byte_3;
        AAL5_Unitdata_Req_aal5type_OUI1 : out byte;
        AAL5_Unitdata_Req_aal5type_OUI2 : out  byte_2;
        AAL5_Unitdata_Req_aal5type_TIPO : out byte_2;
        AAL5_Unitdata_Req_sourceaddr : out end_ipv6;
        AAL5_Unitdata_Req_destaddr : out end_ipv6;
        AAL5_Unitdata_Req_nextheader2 : out byte;
        AAL5_Unitdata_Req_reserved : out byte;
        AAL5_Unitdata_Req_fragoffset : out integer;
        AAL5_Unitdata_Req_res : out bit_2;
        AAL5_Unitdata_Req_M : out bit;
        AAL5_Unitdata_Req_identification : out integer;
        AAL5_Unitdata_Req_tam : out integer;
        --   VarInd
        varind_ackout1 : in bit;
        varind_ackout2 : in bit;
        varind_validout1 : out bit;
        varind_validout2 : out bit;
        --   Ready
        ready_validin1 : in bit;
        ready_validin2 : in bit;
        ready_ackin1 : out bit;
        ready_ackin2 : out bit;
```

```
        --   AAL5_Unitdata_Ind
        AAL5_Unitdata_Ind_cod : in bit;
        AAL5_Unitdata_Ind_version : in bit_4;
        AAL5_Unitdata_Ind_priority : in bit_4;
        AAL5_Unitdata_Ind_flowlabel : in byte_3;
        AAL5_Unitdata_Ind_payload_length : in integer;
        AAL5_Unitdata_Ind_nextheader : in byte;
        AAL5_Unitdata_Ind_hoplimit : in byte;
        AAL5_Unitdata_Ind_dados : in character;
        AAL5_Unitdata_Ind_sourceaddr : in end_ipv6;
        AAL5_Unitdata_Ind_destaddr : in end_ipv6;
        -- AAL5_Unitdata_Ind_tam : in integer;
        -- Esta primitiva nao precisa ter o tam, pois
        -- o tamanho e dado pelo campo do cabecalho IP.
        AAL5_Unitdata_Ind_nextheader2 : in byte;
        AAL5_Unitdata_Ind_reserved : in byte;
        AAL5_Unitdata_Ind_fragoffset : in integer;
        AAL5_Unitdata_Ind_res : in bit_2;
        AAL5_Unitdata_Ind_M : in bit;
        AAL5_Unitdata_Ind_identification : in integer;
        --   IPv6_Data_Req
        IPv6_Data_Req_cod : in bit;
        IPv6_Data_Req_ipsrc : in end_ipv6;
        IPv6_Data_Req_ipdest : in end_ipv6;
        IPv6_Data_Req_flowlabel : in byte_3;
        IPv6_Data_Req_priority : in bit_4;
        IPv6_Data_Req_hoplimit : in byte;
        IPv6_Data_Req_sdu : in character;
        IPv6_Data_Req_tam : in integer;
        --   IPv6_Data_Ind
        IPv6_Data_Ind_cod : out bit;
        IPv6_Data_Ind_ipsrc : out end_ipv6;
        IPv6_Data_Ind_ipdest : out end_ipv6;
        IPv6_Data_Ind_sdu : out character);

END IPv6_ENTITYFRAG_TYPE;

ARCHITECTURE IPv6_ENTITYFRAG_BODY OF IPv6_ENTITYFRAG_TYPE IS

  TYPE tam_msg_usu IS array (0 to 34) of character;


BEGIN  --  IPv6_ENTITY_BODY
 -------------------------------------------------------------------------------
 --  Process:
 --  Purpose:
 --  Inputs:
 --  Outputs:
 -------------------------------------------------------------------------------
 PROCESS

  VARIABLE      buffer_LLC       : byte_3;
  VARIABLE      buffer_OUI1      : byte;
  VARIABLE      buffer_OUI2, buffer_TIPO         : byte_2;
  VARIABLE      i, sdu_usu, aal5_pdu     : integer;
  VARIABLE      buffer_version, buffer_priority  : bit_4;
  VARIABLE      buffer_flowlabel         : byte_3;
  VARIABLE      buffer_payload_length, buffer_payload_length_pri : integer;
  VARIABLE      buffer_usuipv6_sdu, buffer_aal5ipv6_pdu : tam_msg_usu;
  VARIABLE      buffer_hoplimit, buffer_nextheader       : byte;
  VARIABLE      buffer_sourceaddr, buffer_destaddr  : end_ipv6;
  VARIABLE      MTU, tam_dat, identif, buffer_fragoffset : integer;
  VARIABLE      buffer_nextheader2, reserved : byte;
  VARIABLE      res : bit_2;
```

```
VARIABLE      buffer_identif : integer;
VARIABLE      M : bit;
VARIABLE      posini, posfim, dados_env, NFB, NFB_aux, frag_aux : integer;
VARIABLE      tamanho_dados_pri, tamanho_dados_ult, concat_aux : integer;


PROCEDURE LermsgUserIPv6 is
begin
  i := 0;
  lendomsgusu : WHILE (i < sdu_usu) LOOP
  WAIT UNTIL (ready_validin1 = '1');
  buffer_usuipv6_sdu(i) := IPv6_Data_Req_sdu;
  ready_ackin1 <= '1';
  WAIT UNTIL (ready_validin1 = '0');
  ready_ackin1 <= '0';
  i := i + 1;
  END LOOP lendomsgusu;
  end LermsgUserIPv6;

PROCEDURE EnviarmsgIPv6aal5 is
begin
  i := 0;
  enviandomsgaal5 : WHILE (i < sdu_usu) LOOP
  IF (varind_ackout1 /= '0') THEN
  WAIT UNTIL varind_ackout1 = '0';
  END IF;
  AAL5_Unitdata_Req_dados <= buffer_usuipv6_sdu(i);
  varind_validout1 <= '1';
  WAIT UNTIL (varind_ackout1 = '1');
  varind_validout1 <= '0';
  i := i + 1;
  END LOOP enviandomsgaal5;
  end EnviarmsgIPv6aal5;

PROCEDURE EnviarIPv6fragaal5 is
begin
  enviandofragaal5 : WHILE (frag_aux < posfim) LOOP
  IF (varind_ackout1 /= '0') THEN
  WAIT UNTIL varind_ackout1 = '0';
  END IF;
  AAL5_Unitdata_Req_dados <= buffer_usuipv6_sdu(posini);
  varind_validout1 <= '1';
  WAIT UNTIL (varind_ackout1 = '1');
  varind_validout1 <= '0';
  frag_aux := frag_aux + 1;
  posini := posini + 1;
  END LOOP enviandofragaal5;
  end EnviarIPv6fragaal5;

PROCEDURE Lermsgaal5IPv6 is
  begin
    i := 0;
    lendomsgaal5 : WHILE (i < aal5_pdu) LOOP
    WAIT UNTIL (ready_validin2 = '1');
    buffer_aal5ipv6_pdu(i) := AAL5_Unitdata_Ind_dados;
    ready_ackin2 <= '1';
    WAIT UNTIL (ready_validin2 = '0');
    ready_ackin2 <= '0';
    i := i + 1;
    END LOOP lendomsgaal5;
  end Lermsgaal5IPv6;

PROCEDURE Lerfragaal5IPv6 is
  begin
```

```
    i := 0;
    lendomsgaal5 : WHILE (i < aal5_pdu) LOOP
    WAIT UNTIL (ready_validin2 = '1');
    buffer_aal5ipv6_pdu(concat_aux) := AAL5_Unitdata_Ind_dados;
    ready_ackin2 <= '1';
    WAIT UNTIL (ready_validin2 = '0');
    ready_ackin2 <= '0';
    i := i + 1;
    concat_aux := concat_aux + 1;
    END LOOP lendomsgaal5;
  end Lerfragaal5IPv6;

PROCEDURE EnviarmsgIPv6User is
  begin
    i := 0;
    enviandomsguser : WHILE (i < aal5_pdu) LOOP
    IF ((varind_ackout2 /= '0')) THEN
      WAIT UNTIL varind_ackout2 = '0';
    END IF;
    IPv6_Data_Ind_sdu <= buffer_aal5ipv6_pdu(i);
    varind_validout2 <= '1';
    WAIT UNTIL (varind_ackout2 = '1');
    varind_validout2 <= '0';
    i := i + 1;
    END LOOP enviandomsguser;
  end EnviarmsgIPv6User;

PROCEDURE inicio is
  begin
    varind_validout1 <= '0';
    varind_validout2 <= '0';
    ready_ackin1 <= '0';
    ready_ackin2 <= '0';
    ipv6habilitado <= '0';
    MTU := 70;
    identif := 1;
  end inicio;



  begin --   process
    IPv6_Data_Ind_cod <= '0';
    AAL5_Unitdata_Req_cod <= '0';
    ipv6preparado <= '0';
    WAIT UNTIL (start = '1' and clk'event and clk = '1');
    MEF : LOOP
    ipv6preparado <= '1';
    esperandoativacao : WHILE (IPv6_Data_Req_cod /= '1' AND AAL5_Unitdata_Ind_cod /= '1') LOOP
    WAIT UNTIL (clk'event and clk = '1');
    END LOOP esperandoativacao;
    ipv6preparado <= '0';
    inicio;
    IF (IPv6_Data_Req_cod = '1') THEN
      sdu_usu := IPv6_Data_Req_tam;
      tam_dat := sdu_usu + 40;
        buffer_version := "0110";
        buffer_LLC := "101010101010101000000011";
        buffer_OUI1 := "00000000";
        buffer_OUI2 := "0000000000000000";
        buffer_TIPO := "1000011011011101";
        buffer_priority := IPv6_Data_Req_priority;
        buffer_flowlabel := IPv6_Data_Req_flowlabel;
        buffer_hoplimit := IPv6_Data_Req_hoplimit;
        buffer_sourceaddr := IPv6_Data_Req_ipsrc;
        buffer_destaddr := IPv6_Data_Req_ipdest;
```

```
  LermsgUserIPv6;
IF (tam_dat > MTU) THEN
  buffer_identif := identif;
  identif := identif + 1;
  buffer_nextheader2 := "00000110";
  reserved := "00000000";
  res := "00";
  M := '1';
  posini := 0;
  frag_aux := 0;
  buffer_fragoffset := 0;
  dados_env := 0;
  NFB := (MTU - 48) / 8;
  NFB_aux := NFB * 8;
  posfim := NFB_aux;
  fragmentacao: WHILE (dados_env < sdu_usu) LOOP
     AAL5_Unitdata_Req_cod <= '1';
     AAL5_Unitdata_Req_tam <= posfim;
     buffer_payload_length := NFB_aux + 8;

     -- WAIT UNTIL aal5preparado = '1';

     buffer_nextheader := "00101100";
     AAL5_Unitdata_Req_aal5type_LLC <= buffer_LLC;
     AAL5_Unitdata_Req_aal5type_OUI1 <= buffer_OUI1;
     AAL5_Unitdata_Req_aal5type_OUI2 <= buffer_OUI2;
     AAL5_Unitdata_Req_aal5type_TIPO <= buffer_TIPO;
     AAL5_Unitdata_Req_version <= buffer_version;
     AAL5_Unitdata_Req_priority <= buffer_priority;
     AAL5_Unitdata_Req_flowlabel <= buffer_flowlabel;
     AAL5_Unitdata_Req_payload_length <= buffer_payload_length;
     AAL5_Unitdata_Req_nextheader <= buffer_nextheader;
     AAL5_Unitdata_Req_hoplimit <= buffer_hoplimit;
     AAL5_Unitdata_Req_sourceaddr <= buffer_sourceaddr;
     AAL5_Unitdata_Req_destaddr <= buffer_destaddr;
     AAL5_Unitdata_Req_nextheader2 <= buffer_nextheader2;
     AAL5_Unitdata_Req_reserved <= reserved;
     AAL5_Unitdata_Req_fragoffset <= buffer_fragoffset;
     AAL5_Unitdata_Req_res <= res;
     AAL5_Unitdata_Req_M <= M;
     AAL5_Unitdata_Req_identification <= buffer_identif;
     WAIT UNTIL aal5preparado = '1';
     EnviarIPv6fragaal5;
     AAL5_Unitdata_Req_cod <= '0';
     WAIT UNTIL aal5preparado = '0';
     frag_aux := 0;
     buffer_fragoffset := buffer_fragoffset + NFB;
     dados_env := dados_env + NFB_aux;
     IF (dados_env < sdu_usu) THEN
       posini := dados_env;
       NFB_aux := sdu_usu - dados_env;
       IF (NFB_aux > (NFB * 8)) THEN
         NFB_aux := NFB * 8;
         M := '1';
       ELSE
         M := '0';
       END IF;
       posfim := NFB_aux;
     END IF;
  END LOOP fragmentacao;
ELSE
  AAL5_Unitdata_Req_cod <= '1';
  AAL5_Unitdata_Req_tam <= sdu_usu;
```

```
      WAIT UNTIL aal5preparado = '1';

      buffer_nextheader := "00000110";
      buffer_payload_length := sdu_usu;
      AAL5_Unitdata_Req_aal5type_LLC <= buffer_LLC;
      AAL5_Unitdata_Req_aal5type_OUI1 <= buffer_OUI1;
      AAL5_Unitdata_Req_aal5type_OUI2 <= buffer_OUI2;
      AAL5_Unitdata_Req_aal5type_TIPO <= buffer_TIPO;
      AAL5_Unitdata_Req_version <= buffer_version;
      AAL5_Unitdata_Req_priority <= buffer_priority;
      AAL5_Unitdata_Req_flowlabel <= buffer_flowlabel;
      AAL5_Unitdata_Req_payload_length <= buffer_payload_length;
      AAL5_Unitdata_Req_nextheader <= buffer_nextheader;
      AAL5_Unitdata_Req_hoplimit <= buffer_hoplimit;
      AAL5_Unitdata_Req_sourceaddr <= buffer_sourceaddr;
      AAL5_Unitdata_Req_destaddr <= buffer_destaddr;
      EnviarmsgIPv6aal5;
    END IF;
  ELSIF (AAL5_Unitdata_Ind_cod = '1') THEN
      ipv6habilitado <= '1';
      buffer_version := AAL5_Unitdata_Ind_version;
      IF (buffer_version = "0110") THEN
         buffer_priority := AAL5_Unitdata_Ind_priority;
         buffer_flowlabel := AAL5_Unitdata_Ind_flowlabel;
         buffer_payload_length := AAL5_Unitdata_Ind_payload_length;
         buffer_nextheader := AAL5_Unitdata_Ind_nextheader;
         IF (buffer_nextheader = "00101100") THEN
           buffer_nextheader2 := AAL5_Unitdata_Ind_nextheader2;
           buffer_fragoffset := AAL5_Unitdata_Ind_fragoffset;
           reserved := AAL5_Unitdata_Ind_reserved;
           res := AAL5_Unitdata_Ind_res;
           M := AAL5_Unitdata_Ind_M;
           buffer_identif := AAL5_Unitdata_Ind_identification;
           aal5_pdu := buffer_payload_length - 8;
           IF (buffer_fragoffset = 0 AND M = '1') THEN
             concat_aux := 0;
             buffer_payload_length_pri := buffer_payload_length;
             tamanho_dados_pri := buffer_payload_length - 8;
             Lerfragaal5IPv6;
           ELSE
             Lerfragaal5IPv6;
             IF (M = '0') THEN
               tamanho_dados_ult := buffer_payload_length -8;
               IF (buffer_payload_length_pri - tamanho_dados_pri - 8
                  + (8 * buffer_fragoffset) + tamanho_dados_ult)
                  = concat_aux THEN
                 buffer_destaddr := AAL5_Unitdata_Ind_destaddr;
                 buffer_sourceaddr := AAL5_Unitdata_Ind_sourceaddr;
                 IPv6_Data_Ind_cod <= '1';
                 WAIT UNTIL testpreparado = '1';
                 IPv6_Data_Ind_ipsrc <= buffer_sourceaddr;
                 IPv6_Data_Ind_ipdest <= buffer_destaddr;
                 aal5_pdu := concat_aux;
                 EnviarmsgIPv6User;
               END IF;
             END IF;
           END IF;
         ELSE
          buffer_hoplimit := AAL5_Unitdata_Ind_hoplimit;
          aal5_pdu := buffer_payload_length;
          buffer_sourceaddr := AAL5_Unitdata_Ind_sourceaddr;
          buffer_destaddr := AAL5_Unitdata_Ind_destaddr;
          Lermsgaal5IPv6;
          IPv6_Data_Ind_cod <= '1';
```

```
            WAIT UNTIL testpreparado = '1';
            IPv6_Data_Ind_ipsrc <= buffer_sourceaddr;
            IPv6_Data_Ind_ipdest <= buffer_destaddr;
            EnviarmsgIPv6User;
           END IF;
        END IF;
    END IF;
  WAIT UNTIL (clk'event and clk = '1');
  IPv6_Data_Ind_cod <= '0';
  AAL5_Unitdata_Req_cod <= '0';
  ipv6habilitado <= '0';
  ipv6preparado <= '0';
  END LOOP MEF;
 end process;
END IPv6_ENTITYFRAG_BODY;
```

Apêndice C

Descrição sem Dependências do Protocolo IPv6

Neste apêndice encontra-se a descrição sem dependências do Protocolo IPv6 sem os *buffers* para o armazenamento dos dados Esta descrição possui uma máquina de 21 estados.

```
-- modulo IPv6sub escrito em VHDL com
-- objetivo de sintese.
-- Data Inicio : 04/12/97
-- Data Termino :

LIBRARY std;
  use std.standard.all;

LIBRARY work;
  use work.processador_types.all;


ENTITY IPv6_ENTITY_SEMBUFFER_TYPE_SINT IS
  port (start: in bit;
        clk: in bit;
        --   Sinais de Controle de modo a deixar a entidade IP
        --   habilitada a receber os dados do testbench
        ipv6preparado : out bit;
        testpreparado : in bit;
        --   Sinais de controle de modo a deixar a entidade AAL5
        --   apta a receber os dados do IPv6.
        ipv6habilitado : out bit;
        aal5preparado : in bit;
        --   AAL5_Unitdata_Req
        AAL5_Unitdata_Req_cod : out bit;
        AAL5_Unitdata_Req_version : out bit_4;
        AAL5_Unitdata_Req_priority : out bit_4;
        AAL5_Unitdata_Req_flowlabel : out byte_3;
        AAL5_Unitdata_Req_payload_length : out integer;
        AAL5_Unitdata_Req_nextheader : out byte;
```

```
        AAL5_Unitdata_Req_hoplimit : out byte;
        AAL5_Unitdata_Req_dados : out character;
        AAL5_Unitdata_Req_aal5type_LLC : out byte_3;
        AAL5_Unitdata_Req_aal5type_OUI1 : out byte;
        AAL5_Unitdata_Req_aal5type_OUI2 : out  byte_2;
        AAL5_Unitdata_Req_aal5type_TIPO : out byte_2;
        AAL5_Unitdata_Req_sourceaddr : out end_ipv6;
        AAL5_Unitdata_Req_destaddr : out end_ipv6;
        AAL5_Unitdata_Req_tam : out integer;
        --   VarInd
        varind_ackout1 : in bit;
        varind_ackout2 : in bit;
        varind_validout1 : out bit;
        varind_validout2 : out bit;
        --   Ready
        ready_validin1 : in bit;
        ready_validin2 : in bit;
        ready_ackin1 : out bit;
        ready_ackin2 : out bit;
        --   AAL5_Unitdata_Ind
        AAL5_Unitdata_Ind_cod : in bit;
        AAL5_Unitdata_Ind_version : in bit_4;
        AAL5_Unitdata_Ind_priority : in bit_4;
        AAL5_Unitdata_Ind_flowlabel : in byte_3;
        AAL5_Unitdata_Ind_payload_length : in integer;
        AAL5_Unitdata_Ind_nextheader : in byte;
        AAL5_Unitdata_Ind_hoplimit : in byte;
        AAL5_Unitdata_Ind_dados : in character;
        AAL5_Unitdata_Ind_sourceaddr : in end_ipv6;
        AAL5_Unitdata_Ind_destaddr : in end_ipv6;
        -- AAL5_Unitdata_Ind_tam : in integer;
        -- Esta primitiva nao precisa ter o tam, pois
        -- o tamanho e dado pelo campo do cabecalho IP.
        --   IPv6_Data_Req
        IPv6_Data_Req_cod : in bit;
        IPv6_Data_Req_ipsrc : in end_ipv6;
        IPv6_Data_Req_ipdest : in end_ipv6;
        IPv6_Data_Req_flowlabel : in byte_3;
        IPv6_Data_Req_priority : in bit_4;
        IPv6_Data_Req_hoplimit : in byte;
        IPv6_Data_Req_sdu : in character;
        IPv6_Data_Req_tam : in integer;
        --   IPv6_Data_Ind
        IPv6_Data_Ind_cod : out bit;
        IPv6_Data_Ind_ipsrc : out end_ipv6;
        IPv6_Data_Ind_ipdest : out end_ipv6;
        IPv6_Data_Ind_sdu : out character;
        -- Sinais para a comunicacao c/ a memoria
        usu_envia_mem : out character;
        usu_envia_mem_cod : out bit;
        ipv6_envia_aal5_mem : in character;
        ipv6_envia_aal5_mem_cod : out bit;
        ipv6_recebe_aal5_mem : out character;
        ipv6_recebe_aal5_mem_cod : out bit;
        ipv6_envia_usu_mem : in character;
        ipv6_envia_usu_mem_cod : out bit;
        mem_autorizando : in bit;
        mem_autorizando_usu : in bit);


END IPv6_ENTITY_SEMBUFFER_TYPE_SINT;

ARCHITECTURE IPv6_ENTITY_SEMBUFFER_BODY_SINT OF IPv6_ENTITY_SEMBUFFER_TYPE_SINT IS
```

```
-- TYPE tam_msg_usu IS array (0 to 3) of character;

  CONSTANT      S1              : integer := 1;
  CONSTANT      S2              : integer := 2;
  CONSTANT      S3              : integer := 3;
  CONSTANT      S4              : integer := 4;
  CONSTANT      S5              : integer := 5;
  CONSTANT      S6              : integer := 6;
  CONSTANT      S7              : integer := 7;
  CONSTANT      S8              : integer := 8;
  CONSTANT      S9              : integer := 9;
  CONSTANT      S10             : integer := 10;
  CONSTANT      S11             : integer := 11;
  CONSTANT      S12             : integer := 12;
  CONSTANT      S13             : integer := 13;
  CONSTANT      S14             : integer := 14;
  CONSTANT      S15             : integer := 15;
  CONSTANT      S16             : integer := 16;
  CONSTANT      S17             : integer := 17;
  CONSTANT      S18             : integer := 18;
  CONSTANT      S19             : integer := 19;
  CONSTANT      S20             : integer := 20;
  CONSTANT      S21             : integer := 21;
  SIGNAL  state : integer;


BEGIN  --  IPv6_ENTITY_BODY_SINT

  PROCESS

  VARIABLE      buffer_LLC       : byte_3;
  VARIABLE      buffer_OUI1      : byte;
  VARIABLE      buffer_OUI2, buffer_TIPO         : byte_2;
  VARIABLE      i, sdu_usu, aal5_pdu     : integer;
  VARIABLE      buffer_version, buffer_priority  : bit_4;
  VARIABLE      buffer_flowlabel         : byte_3;
  VARIABLE      buffer_payload_length    : integer;
  VARIABLE      buffer_hoplimit, buffer_nextheader       : byte;
  VARIABLE      buffer_sourceaddr, buffer_destaddr  : end_ipv6;

    PROCEDURE inicio is
      begin
        varind_validout1 <= '0';
        varind_validout2 <= '0';
        ready_ackin1 <= '0';
        ready_ackin2 <= '0';
        ipv6habilitado <= '0';
    end inicio;

    begin  -- process
      IPv6_Data_Ind_cod <= '0';
      AAL5_Unitdata_Req_cod <= '0';
      ipv6preparado <= '0';
      usu_envia_mem_cod <= '0';
      ipv6_envia_aal5_mem_cod <= '0';
      ipv6_recebe_aal5_mem_cod <= '0';
      ipv6_envia_usu_mem_cod <= '0';
      state <= S1;

      MEF : LOOP

      WAIT UNTIL clk'event AND clk = '1'; -- tem que ter este wait!

      CASE state IS
```

```
WHEN 1 =>   -- wait until start='1' and clk='1'
  IF (start /= '1') THEN
    state <= S1;
  ELSE
    ipv6preparado <= '1';
    IF (IPv6_Data_Req_cod /= '0' OR AAL5_Unitdata_Ind_cod /= '0') THEN
      ipv6preparado <= '0';
      inicio;
      IF (IPv6_Data_Req_cod = '1') THEN
        sdu_usu := IPv6_Data_Req_tam;
        buffer_version := "0110";
        buffer_LLC := "101010101010101000000011";
        buffer_OUI1 := "00000000";
        buffer_OUI2 := "0000000000000000";
        buffer_TIPO := "1000011011011101";
        buffer_priority := IPv6_Data_Req_priority;
        buffer_flowlabel := IPv6_Data_Req_flowlabel;
        buffer_hoplimit := IPv6_Data_Req_hoplimit;
        buffer_sourceaddr := IPv6_Data_Req_ipsrc;
        buffer_destaddr := IPv6_Data_Req_ipdest;
        i := 0;
        state <= S2;
      ELSIF (AAL5_Unitdata_Ind_cod = '1') THEN
        ipv6habilitado <= '1';
        buffer_version := AAL5_Unitdata_Ind_version;
        state <= S3;
      END IF;
    END IF;
  END IF;
WHEN 2 => -- LermsguserIPv6
  IF (i < sdu_usu) THEN
    state <= S4;
  ELSE
    state <= S5;
  END IF;
WHEN 3 =>
  IF (buffer_version = "0110") THEN
    buffer_priority := AAL5_Unitdata_Ind_priority;
    buffer_flowlabel := AAL5_Unitdata_Ind_flowlabel;
    buffer_payload_length := AAL5_Unitdata_Ind_payload_length;
    buffer_hoplimit := AAL5_Unitdata_Ind_hoplimit;
    buffer_nextheader := AAL5_Unitdata_Ind_nextheader;
    aal5_pdu := buffer_payload_length;
    buffer_sourceaddr := AAL5_Unitdata_Ind_sourceaddr;
    buffer_destaddr := AAL5_Unitdata_Ind_destaddr;
    i := 0;
    state <= S6;
  ELSE
    IPv6_Data_Ind_cod <= '0';
    AAL5_Unitdata_Req_cod <= '0';
    ipv6habilitado <= '0';
    ipv6preparado <= '0';
    state <= S1;
  END IF;
WHEN 4 =>
  IF (ready_validin1 /= '1') THEN
    state <= S4;
  ELSE
    usu_envia_mem_cod <= '1';
    usu_envia_mem <= IPv6_Data_Req_sdu;
    ready_ackin1 <= '1';
    state <= S7;
  END IF;
WHEN 5 =>
```

```
  AAL5_Unitdata_Req_cod <= '1';
  AAL5_Unitdata_Req_tam <= sdu_usu;
  state <= S8;
WHEN 6 =>
  IF (i < aal5_pdu) THEN
    state <= S9;
  ELSE
    IPv6_Data_Ind_cod <= '1';
    state <= S10;
  END IF;
WHEN 7 =>
  IF (ready_validin1 /= '0') THEN
    state <= S7;
  ELSE
    usu_envia_mem_cod <= '0';
    ready_ackin1 <= '0';
    i := i + 1;
    state <= S2;
  END IF;
WHEN 8 =>
  IF (aal5preparado /= '1') THEN
    state <= S8;
  ELSE
    buffer_nextheader := "00000110";
    buffer_payload_length := sdu_usu;
    AAL5_Unitdata_Req_aal5type_LLC <= buffer_LLC;
    AAL5_Unitdata_Req_aal5type_OUI1 <= buffer_OUI1;
    AAL5_Unitdata_Req_aal5type_OUI2 <= buffer_OUI2;
    AAL5_Unitdata_Req_aal5type_TIPO <= buffer_TIPO;
    AAL5_Unitdata_Req_version <= buffer_version;
    AAL5_Unitdata_Req_priority <= buffer_priority;
    AAL5_Unitdata_Req_flowlabel <= buffer_flowlabel;
    AAL5_Unitdata_Req_payload_length <= buffer_payload_length;
    AAL5_Unitdata_Req_nextheader <= buffer_nextheader;
    AAL5_Unitdata_Req_hoplimit <= buffer_hoplimit;
    AAL5_Unitdata_Req_sourceaddr <= buffer_sourceaddr;
    AAL5_Unitdata_Req_destaddr <= buffer_destaddr;
    i := 0;
    state <= S11;
  END IF;
WHEN 9 =>
  IF (ready_validin2 /= '1') THEN
    state <= S9;
  ELSE
    ipv6_recebe_aal5_mem_cod <= '1';
    ipv6_recebe_aal5_mem <= AAL5_Unitdata_Ind_dados;
    ready_ackin2 <= '1';
    state <= S12;
  END IF;
WHEN 10 =>
  IF (testpreparado /= '1') THEN
    state <= S10;
  ELSE
    IPv6_Data_Ind_ipsrc <= buffer_sourceaddr;
    IPv6_Data_Ind_ipdest <= buffer_destaddr;
    i := 0;
    state <= S13;
  END IF;
WHEN 11 =>
  IF (i < sdu_usu) THEN
    ipv6_envia_aal5_mem_cod <= '1';
    state <= S14;
  ELSE
    IPv6_Data_Ind_cod <= '0';
```

```
      AAL5_Unitdata_Req_cod <= '0';
      ipv6habilitado <= '0';
      ipv6preparado <= '0';
      state <= S1;
    END IF;
  WHEN 12 =>
    IF (ready_validin2 /= '0') THEN
      state <= S12;
    ELSE
      ipv6_recebe_aal5_mem_cod <= '0';
      ready_ackin2 <= '0';
      i := i + 1;
      state <= S6;
    END IF;
  WHEN 13 =>
    IF (i < aal5_pdu) THEN
      ipv6_envia_usu_mem_cod <= '1';
      state <= S15;
    ELSE
      IPv6_Data_Ind_cod <= '0';
      AAL5_Unitdata_Req_cod <= '0';
      ipv6habilitado <= '0';
      ipv6preparado <= '0';
      state <= S1;
    END IF;
  WHEN 14 =>
    IF (mem_autorizando /= '1') THEN
      state <= S14;
    ELSE
      IF (varind_ackout1 /= '0') THEN
        state <= S16;
      ELSE
        AAL5_Unitdata_Req_dados <= ipv6_envia_aal5_mem;
        ipv6_envia_aal5_mem_cod <= '0';
        state <= S17;
      END IF;
    END IF;
  WHEN 15 =>
    IF (mem_autorizando_usu /= '1') THEN
      state <= S15;
    ELSE
      IF (varind_ackout2 /= '0') THEN
        state <= S18;
      ELSE
        IPv6_Data_Ind_sdu <= ipv6_envia_usu_mem;
        ipv6_envia_usu_mem_cod <= '0';
        state <= S19;
      END IF;
    END IF;
  WHEN 16 =>
    IF (varind_ackout1 /= '0') THEN
      state <= S16;
    ELSE
      AAL5_Unitdata_Req_dados <= ipv6_envia_aal5_mem;
      ipv6_envia_aal5_mem_cod <= '0';
      state <= S17;
    END IF;
  WHEN 17 =>
    IF (mem_autorizando /= '0') THEN
      state <= S17;
    ELSE
      varind_validout1 <= '1';
      state <= S20;
    END IF;
```

```
      WHEN 18 =>
        IF (varind_ackout2 /= '0') THEN
          state <= S18;
        ELSE
          IPv6_Data_Ind_sdu <= ipv6_envia_usu_mem;
          ipv6_envia_usu_mem_cod <= '0';
          state <= S19;
        END IF;
      WHEN 19 =>
        IF (mem_autorizando_usu /= '0') THEN
          state <= S19;
        ELSE
          varind_validout2 <= '1';
          state <= S21;
        END IF;
      WHEN 20 =>
        IF (varind_ackout1 /= '1') THEN
          state <= S20;
        ELSE
          varind_validout1 <= '0';
          i := i + 1;
          state <= S11;
        END IF;
      WHEN 21 =>
        IF (varind_ackout2 /= '1') THEN
          state <= S21;
        ELSE
          varind_validout2 <= '0';
          i := i + 1;
          state <= S13;
        END IF;
      WHEN others =>
    END CASE;

    -- WAIT UNTIL clk'event and clk = '1';

    END LOOP MEF;

  END PROCESS;

END IPv6_ENTITY_SEMBUFFER_BODY_SINT;
```