Mapeamento de Estelle em VHDL e Estilos de Descrição de protocolos de Comunicação para Síntese de Alto Nível

Luci Pirmez (1) (2), Aloysio de Castro P. Pedroza (1) (3),
Antonio Carneiro de Mesquita (1) Ahmed Amine Jerraya(4)

(1) Coppe/UFRJ - Programa de Engenharia Elétrica
Tel: +55 21 2605010
Po. Box 68504 - 21495 Rio de Janeiro RJ Brasil
E-mail: luci@barra.nce.ufrj.br
(2) NCE/UFRJ - Núcleo de Computação Eletrônica
(3) EE/UFRJ - Departamento de Eletrônica
(4) TIMA Laboratory
46,Avenue Félix Viallet
38031 Grenoble Cédex, France

Abstract

This paper presents[1] a methodology that efficiently transforms Estelle specifications into VHDL descriptions. This methodology obtains protocol descriptions suitable to be synthesized using AMICAL, a VHDL based behavioral synthesis tool. The goal is to develop a description style in VHDL to the communication protocols suitable for the high level synthesis in order to efficiently obtain synthesized architectures. It will be shown that, in order to efficiently obtain synthesized architectures, a number of constraints must be imposed on the writing style of VHDL. An example based on the specification of a high speed protocol is discussed.

Resumo

Este artigo apresenta uma metodologia que transforma eficientemente especificações Estelle em descrições VHDL. Esta metodologia obtém descrições de protocolos adequadas a síntese usando o AMICAL, uma ferramenta de síntese comportamental baseada em VHDL. O objetivo é desenvolver um estilo de descrição em VHDL para protocolos de comunicação adequados a síntese de alto nível a fim de obter arquiteturas eficientemente sintetizáveis. Também será mostrado as restrições impostas no estilo de descrição em VHDL a fim de obter estas arquiteturas.

Keywords

Protocolo , Síntese de Alto Nível, VLSI, VHDL

[1] Research supported by CAPES/COFECUB project.

I. INTRODUÇÃO

O surgimento de aplicações que exigem o uso de redes de computadores de alto desempenho e as mudanças na tecnologia disponivel para estas redes demandam modificações no projeto de protocolos. Entretanto, a obtenção de subsistemas de comunicação de alto desempenho não se limita somente à otimização de software e ao emprego do paralelismo para a construção de protocolos mas na implementação destes protocolos em hardware VLSI.

Tradicionalmente, os protocolos têm sido implementados inteiramente em software. Com o aumento dos requisitos de velocidade das redes atuais, tornou-se necessário o particionamento software/hardware, sendo cada uma destas partes endereçada a ambientes de projeto especificos. Uma especificação feita numa linguagem de descrição de hardware permite o uso de sistemas de CAD que incluem ferramentas de síntese e de simulação de projetos. Usando uma linguagem de descrição de hardware como o VHDL podemos imediatamente interfacear com estas ferramentas de síntese. Uma especificação descrita por uma linguagem de descrição de software como Estelle, pode facilmente modelar a hierarquia e a comunicação entre processos, mas não permite interfacear com ferramentas de síntese.

Vários trabalhos de pesquisa estão tentando fazer uma ligação entre uma especificação no nível de sistema e as ferramentas de hardware existentes. Um dos enfoques é traduzir uma especificação no nível de sistema para um ambiente da hardware como a tradução de LOTOS para VHDL [21] e a tradução de Estelle para VHDL [17]. Outro enfoque é criar uma forma intermediária como o SOLAR [22] que permita construir uma plataforma comum para diferentes linguagens como, por exemplo, SDL e VHDL [23]. Um terceiro enfoque é criar uma nova linguagem de especificação como o SpecCharts [24]. Contudo, estes trabalhos preocuparam-se apenas com problemas de simulação e não com os de síntese.

O projeto de protocolos requer ferramentas especificas de HLS (High-level Synthesis), geralmente denominadas de compiladores comportamentais dominados por fluxo de controle. Estas ferramentas são capazes de manipular estruturas complexas de controle incluindo handshaking sofisticados e sequências não estruturadas. Entretanto, a maioria destes compiladores apresenta um problema inevitável, os resultados da compilação dependem da qualidade da descrição de entrada pois uma especificação pode incluir pontos de sincronismo explícitos (como um wait em uma descrição VHDL) que restringem o papel da transformação e otimização normalmente executadas em uma descrição comportamental.

Neste artigo é apresentado uma metodologia para a tradução de especificações Estelle para descrições VHDL. Além disso, este trabalho discute o efeito da descrição de entrada na HLS quando é utilizado o VHDL. É utilizado como ambiente de trabalho o AMICAL, um compilador VHDL comportamental para máquinas predominantemente de controle de fluxo.

Com o objetivo de mostrar este efeito da descrição de entrada, será apresentado um protocolo de alta desempenho denominado de ABRACADABRA_HS.

A seção 2 deste artigo apresenta uma metodologia para o mapeamento de especificações em Estelle para VHDL. Na seção 3 são apresentados os diferentes estilos que podem ser utilizados para descrever um protocolo em VHDL. Na seção 4 é introduzido um protocolo de alto desempenho, o protocolo ABRACADABRA_HS. Os diferentes estilos de descrição são aplicados

no protocolo proposto e os resultados são apresentados. Por fim, a seção 5 destacará os principais resultados já obtidos, as perspectivas e considerações finais.

II. MAPEAMENTO DE ESTELLE PARA VHDL

A metodologia proposta inicia a partir de uma especificação corretamente compilada e simulada obtida com o uso de um ambiente adequado para o desenvolvimento de protocolos em Estelle. Em seguida, esta especificação é transformada numa descrição na linguagem VHDL, conforme ilustrado na Figure 1. Duas soluções são apontadas, uma para simulação [16] e outra para síntese. Como será visto mais tarde, algumas instruções Estelle não são encontradas em VHDL, tais como o mecanismo de filas, e por essa razão, requerem procedimentos específicos para tornar o mapeamento possível.

II.1 AS LINGUAGEM ESTELLE E VHDL

A linguagem Estelle [6] é uma técnica de especificação formal de sistemas concorrentes, em particular os protocolos de comunicação. Uma especificação Estelle possui três componentes básicos: módulos, pontos de interação e primitivas. Um módulo é especificado por um cabeçalho e um corpo. No cabeçalho de um módulo é definido seu atributo de classe, seus pontos de interação e suas variáveis exportadas. No corpo de um módulo é definido seu comportamento. A definição de um módulo corresponde a definição de um 'tipo' dentro de uma especificação. Uma especificação Estelle pode possuir várias instâncias de um mesmo módulo. Os pontos de interação correspondem a definição de canais de comunicação FIFO através dos quais uma informação é transmitida entre os módulos. da mesma forma que os módulos, os pontos de interação correspondem aos 'tipos' de canais de comunicação.

VHDL (Very High Speed Integrated Circuit (VHSIC) Hardware description Language) [9] é uma linguagem padrão desenvolvida pelo Departamento de Defesa Americano (DoD) para descrever sistemas eletrônicos digitais. VHDL suporta o projeto, a descrição e a simulação de estruturas de hardware em diferentes níveis de abstração, do arquitetural ao lógico. Esta linguagem foi projetada para ser independente de qualquer tecnologia específica, ambiente de projeto, ou métodos de projeto, e, consequentemente, possibilitar a integração de qualquer combinação de ambiente, tecnologia, e metodologia.

VHDL permite descrições estruturais e comportamentais. A descrição estrutural permite descrever as estruturas de um projeto, sua decomposição em sub-projetos e a interligação entre eles. A descrição comportamental permite a especificação de funções de um projeto utilizando estruturas familiares da linguagem de programação.

Em VHDL, uma entidade representa um módulo. A entidade pode ser usada como um componente em um projeto ou pode ser o módulo topo do projeto, isto é, o módulo cuja hierarquia é a de nível mais elevado.Uma entidade é composta por dois segmentos, uma *interface* (Entity) e um conjunto de um ou mais *corpos* (architecture). A *interface* define as características observáveis externamente, isto é, as portas de entrada/saída. Um *corpo* define a implementação de uma entidade.

II.2 ESTRATÉGIA DE MAPEAMENTO PARA SIMULAÇÃO

Uma *especificação* em Estelle é composta por um conjunto de módulos organizados de forma hierárquica. Cada *módulo* Estelle corresponde a uma *entidade* VHDL e sua hierarquia é obtida em VHDL através da instrução de declaração *component*. Em VHDL, cada instrução de declaração *component* declara um entidade e sua descrição é armazenada em bibliotecas. A instrução *component instantiation* do VHDL é responsável pela criação de instâncias de entidades VHDL (função *init* do Estelle) e pela conexão das portas de componentes com sinais reais ou com as portas de entidades (função *connect* e *attach* do Estelle). A ligação de uma *entidade* VHDL com sua declaração é obtida através de uma declaração, *configuration declaration*.

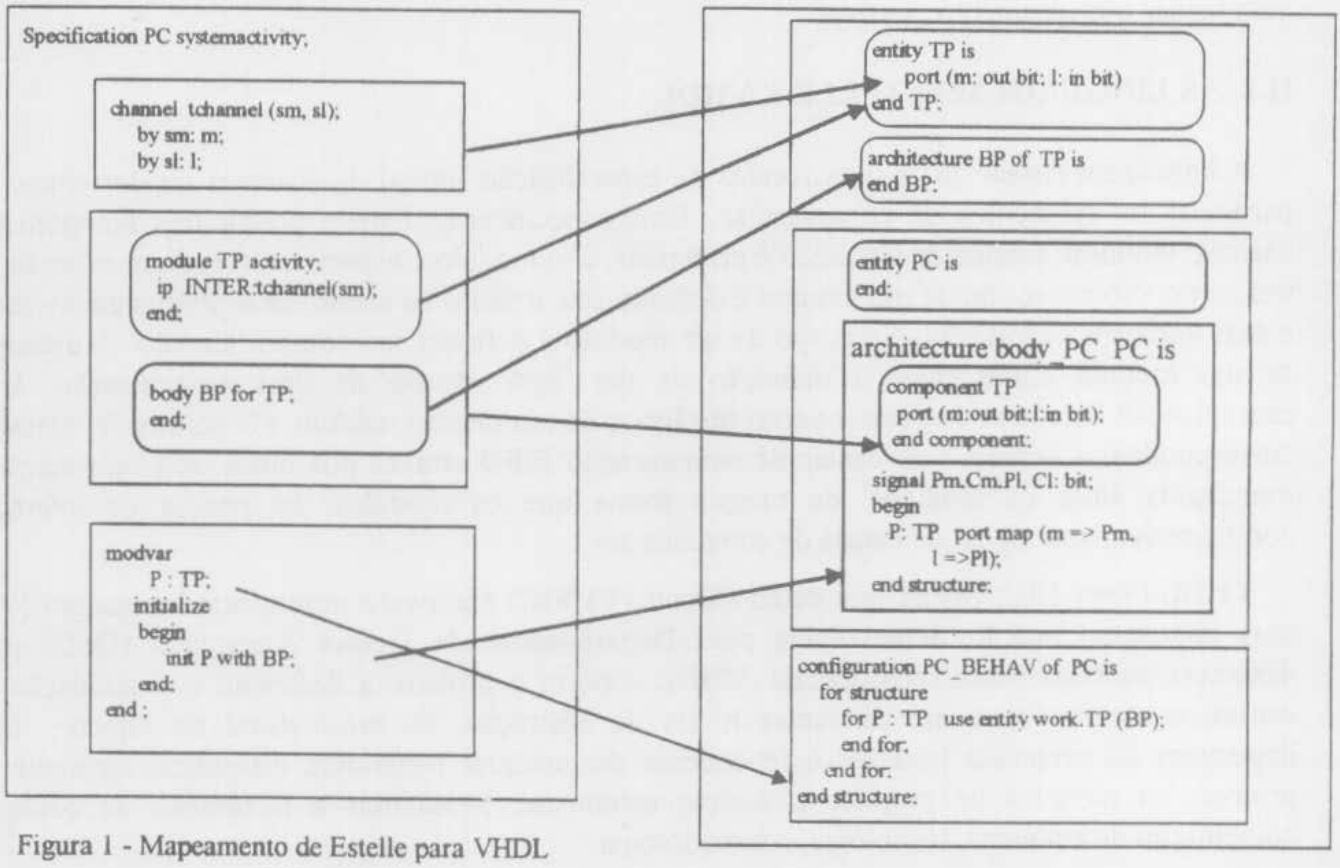


Figura 1 - Mapeamento de Estelle para VHDL

Cada módulo Estelle é composto por duas partes: o cabeçalho (*header*) e um ou mais corpos. A visão externa é definida no cabeçalho e o comportamento interno é descrito dentro do corpo. Cada *entidade* em VHDL é também composta de dois segmentos: uma interface (*entity*) e um conjunto de um ou mais corpos (*architecture*). Assim sendo, cada cabeçalho em Estelle (*module*) corresponderá a uma interface VHDL (*entity*) e cada corpo Estelle (*body*) corresponderá a um corpo VHDL (*architecture*). Os pontos de interação de cada tipo de módulo corresponderá as portas de uma entidade VHDL. As portas em VHDL podem ser dos seguintes tipos: *input*, *output* ou *input/output*. O tipo da porta em VHDL depende do conjunto de interações especificadas por um dado ponto de interação em Estelle.

Dois tipos de criação de módulos são permitidos em Estelle: estático e dinâmico. A opção dinâmica, que permite criar/destruir e conectar/desconectar módulos Estelle enquanto o protocolo está sendo executado, não será considerado neste trabalho.

Dois tipos de paralelismos entre instâncias de módulos podem ser especificados em Estelle: síncrono e assíncrono. Não existem conceitos equivalentes a estes em VHDL quando há mais de um módulo ativo em Estelle. Com o objetivo de obter uma conversão eficiente de Estelle para VHDL, a estratégia de mapeamento inclui as restrições definidas por Courtiat [11] para Estelle e que correspondem ao dialeto denominado Estelle*. Isto permite representar de uma forma assíncrona o paralelismo entre instâncias de módulo do tipo atividade. A restrições de Estelle* são:

- Não existem cláusulas *priority* associadas a qualquer transição definida pela cláusula *when*;
- As únicas instâncias de módulo ativas são as folhas da árvore de instâncias de módulos.

A semântica de um *módulo atividade* em Estelle é implementado em VHDL por uma entidade denominada HIERARCHY_QUEUE_ENTITY. Esta entidade deve selecionar uma instância de módulo para execução.

A linguagem Estelle tem um mecanismo interno de fila FIFO. Infelizmente, VHDL não tem um opção análoga. Uma solução possível é criar duas entidades em VHDL, denominadas de QUEUE_ENTITY e HIERARCHY_QUEUE_ENTITY. A entidade QUEUE_ENTITY será responsável somente pelo gerenciamento de filas de um módulo Estelle. É necessário em VHDL alocar sinais para tornar possível a comunicação entre entidades, conforme ilustrado na figura 2. A HIERARCHY_QUEUE_ENTITY será responsável pela seleção de uma entidade para execução.

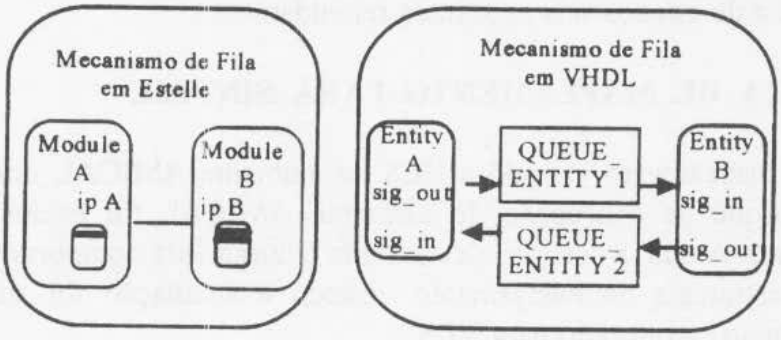


Figura 2 Mecanismo de Filas em Estelle e em VHDL

As variáveis exportadas não podem ser convertidas diretamente para VHDL. Para solucionar este problema, duas portas, uma de entrada e a outra de saída, são criadas nas entidades correspondentes ao módulo considerado e ao seu módulo pai. Deve ser observado que a comunicação entre entidades VHDL é permitido somente através de portas de entrada e saída.

O comportamento de um módulo 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 trivial e o conjunto obtido em VHDL é incluído na instrução process. As cláusulas de Estelle "**from** *state*", "**when** *interação-ponto.interação*", "**provided** *expression*", "**to** *state*" e "**output** *interação-ponto.interação(msg)*" são convertidas diretamente em VHDL para "**when** *state* = > ... *da estrutura case*", "**wait until** *condição*", "**if** *expression* **then** ...", "*state* := *state*

value" e "*interação = message*". As instruções em Estelle "**procedure** *name(lista de parâmetros)*" e "**function** *name(lista de parâmetros) : tipo do parâmetro de retorno*" são diretamente convertidas em declarações VHDL equivalentes.

A cláusula **delay** em Estelle é mais genérica que a cláusula similar em VHDL. A solução para este problema é restringir o uso desta cláusula em Estelle. A cláusula **delay** em Estelle sera usada somente na forma "**delay** *(T)*"ou "**delay** *(T,T)*".

O corpo de uma entidade VHDL será uma descrição comportamental obtida a partir da descrição da máquina de estado em Estelle. Entretanto, VHDL não tem o conceito de maquina de estados embutido na linguagem. Dessa forma, a construção de máquinas de estados em VHDL é obtida da seguinte forma:

- é definida uma variável *state* para armazenar o estado corrente da máquina de estados em VHDL. O conjunto de diferentes estados de um módulo Estelle formará a estrutura *case* em VHDL.
- para cada estado, as transições associadas com esse estado são agrupadas por interação. A instrução "**wait until** *condição_interação*" parará ou continuará a execução dependendo da cláusula *condição_interação* ser avaliada como verdadeira. Se mais de uma *condição_interação* pode ser avaliada como verdadeira em um mesmo estado, a instrução "**if .. then ... else ...**" é posta logo depois da instrução "**wait until** *condição_interações*". Transições expontâneas são executadas quando nenhuma interação for recebida em um dado estado. Estas transições são consideradas na cláusula "**else**" da instrução "**if .. then ... else ...**".

A máquina de estado resultante é inserida dentro da instrução *process* do VHDL permitindo, assim, que a máquina de estados seja executada repetidamente.

II.3 ESTRATÉGIA DE MAPEAMENTO PARA SÍNTESE

A estratégia de mapeamento visando a HLS no ambiente AMICAL difere daquela visando a simulação. Isto porque na concepção do ambiente AMICAL foi excluída a possibilidade de descrições estruturais. Assim, a descrição de entrada é puramente comportamental. Para contornar este problema, a estratégia de mapeamento visando a simulação foi adaptada segundo uma metodologia de projeto estruturado para VLSI.

II.3.1 O AMBIENTE AMICAL

O AMICAL é um sistema de síntese de alto nível que permite o uso da metodologia de projeto estruturado. Ele inicia com dois tipos de informações: uma especificação comportamental dada em VHDL e uma biblioteca de unidades funcionais.

A descrição comportamental aceita pelo AMICAL é formada por um único processo principal que pode conter subsistemas complexos utilizando chamadas a funções e/ou procedimentos. Entretanto, para cada procedimento ou função utilizada, a biblioteca deve incluir no mínimo uma unidade funcional capaz de executar a operação correspondente. Durante os diferentes passos envolvidos na síntese de alto nível, as unidades funcionais são usadas como caixas pretas que

podem executar uma lista de procedimentos. Porém, para completar a descrição no nível de transferência de registros, são necessários os detalhes das unidades funcionais.

II.3.2 ADAPTAÇÃO DA METODOLOGIA DE MAPEAENTO

Um projeto estruturado para síntese, conforme ilustrado na figura 3, é composto por um sistema descrito a nível comportamental e por um conjunto de componentes *(component)*. Assim, um sistema será composto pela junção de componentes. Um componente é um subsistema que será reutilizado e é representado em VHDL por uma entidade. Esta entidade é capaz de executar um conjunto de operações que, no ambiente de síntese de alto nível AMICAL, são ativadas por chamadas a funções ou procedimentos localizados na descrição comportamental do sistema. Um componente também atua como uma caixa preta que liga o nível comportamental com o de transferência de registros. Um componente pode corresponder a um projeto produzido por ferramentas e métodos externos ou a um subsistema sintetizável, isto é, resultante de seções anteriores de projeto.

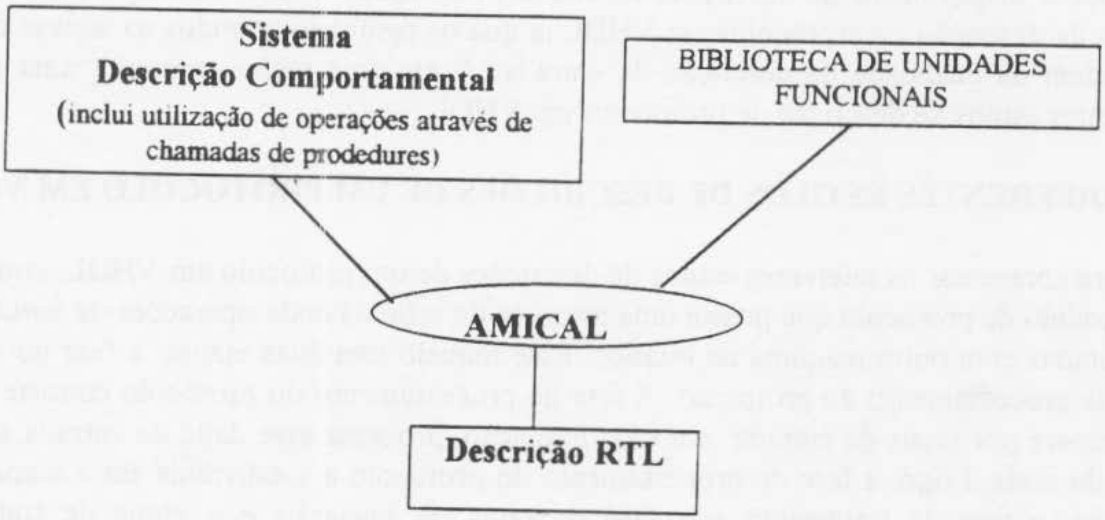


Figura 3 Projeto Estruturado para Síntese

O conceito abstrato de componente no nível comportamental foi empregado para permitir a sua reutilização. Entretanto, o componente assim definido é uma generalização do conceito de unidades funcionais. Cada unidade funcional pode ser especificada em quatro diferentes níveis de abstração:

- No nível conceitual, uma unidade funcional é um objeto que pode executar uma ou mais operações que compartilham eventualmente um ou mais dados;
- No nível de implementação, a informação de interesse é o relacionamento da unidade funcional com o meio externo, isto é, a unidade funcional é representada por suas entradas, saídas e seleção de comandos.
- No nível comportamental, uma unidade funcional é descrita através das operações padrão ou funções e procedimentos que podem ser solicitadas pela descrição comportamental do sistema.
- No nível da síntese, uma unidade funcional é descrita através de sua interface externa, do conjunto de operações que podem ser executadas pela unidade funcional e pelos parâmetros a

serem utilizado por cada operação. Seu objetivo é ligar a visão comportamental com a de implementação.

Com o intuito de permitir projetos hierárquicos e a reutilização de componentes, uma metodologia de projeto estruturado para VLSI é combinado com o AMICAL. Esta metodologia de projeto estruturado para VLSI consiste de três passos:

- Particionamento do sistema em subsistemas;
- Síntese de cada subsistema resultante;
- Abstração de cada subsistema sintetizável para ser usado como componente durante a síntese do sistema de maior nível hierárquico.

Tal metodologia permite a utilização de projetos complexos com enfoque hierárquico. A análise e o particionamento é obtido manualmente. O particionamento divide a especificação do sistema em sub-sistemas mais simples. Estas unidades serão definidas de acordo com o grau de re-utilização.

Após o mapeamento de instruções Estelle em instruções VHDL devemos analisar diferentes estilos de descrição de protocolos em VHDL já que os resultados obtidos na síntese de alto nível dependem da qualidade da descrição de entrada. A próxima seção apresenta uma proposta de diferentes estilos de descrição de protocolos em VHDL.

III. DIFERENTES ESTILOS DE DESCRIÇÕES DE UM PROTOCOLO EM VHDL

Para apresentar os diferentes estilos de descrições de um protocolo em VHDL, consideraremos um modelo de protocolo que possui uma máquina de estados onde operações de *handshaking* são executadas com outra máquina de estados. Esse modelo tem duas etapas: a fase de iniciação e a fase de processamento do protocolo. A fase de processamento do protocolo consiste de um *loop* que espera por sinais de entrada em cada interação, processa esse dado de entrada e transmite o sinal de saída. Logo, a fase de processamento do protocolo é subdividida em 3 etapas: a fase de entrada, a fase de tratamento e a fase de saída. A iniciação e a etapa de tratamento são representadas pelos nós E1 e E2, E3, respectivamente. A etapa de entrada pode ser representada pela instrução "**wait until** *condição_interação*" e a etapa de saída pela instrução "x <= *valor*". Cada um dos nós nas figuras 4, 5, 6, 7 e 8 podem conter mais de uma instrução.

Este modelo será utilizado para analisar os vários estilos de descrição e o efeito de cada estilo nos resultados da síntese de alto nível. Nesta análise nós nos restringiremos aos resultados do *scheduling* o que nos dá a complexidade do controlador. De fato, todos estes estilos produzirão *data_paths* aproximadamente equivalentes. Entretanto, para o projeto de protocolo o *data_path* é menor que o controlador.

O primeiro estilo de descrição distribui as instruções "**wait until ...**" nos ramos da instrução "**case ...**". A figura 4 apresenta a descrição VHDL e o gráfico de controle de fluxo (CFG) resultante do primeiro estilo. A tabela 1 apresenta a tabela de estados resultante do scheduling desta descrição usando o algoritmo DLS (Dynamic Loop Scheduling). Durante a geração do caminho pelo algoritmo DLS, pode ser visto que a geração do caminho corrente, por exemplo E1_Loop_Case, é interrompido e um novo caminho, por exemple o caminho Wait_E2_out_Loop_Case, começa quando a instrução "**wait** until **...**" é encontrada. É importante

observar que neste exemplo existem duas possibilidades para começar um novo caminho e isto ocorre devido a existência de uma instrução **"wait until** *condição_entrada* **"** em cada opção da instrução **"case ..."**.

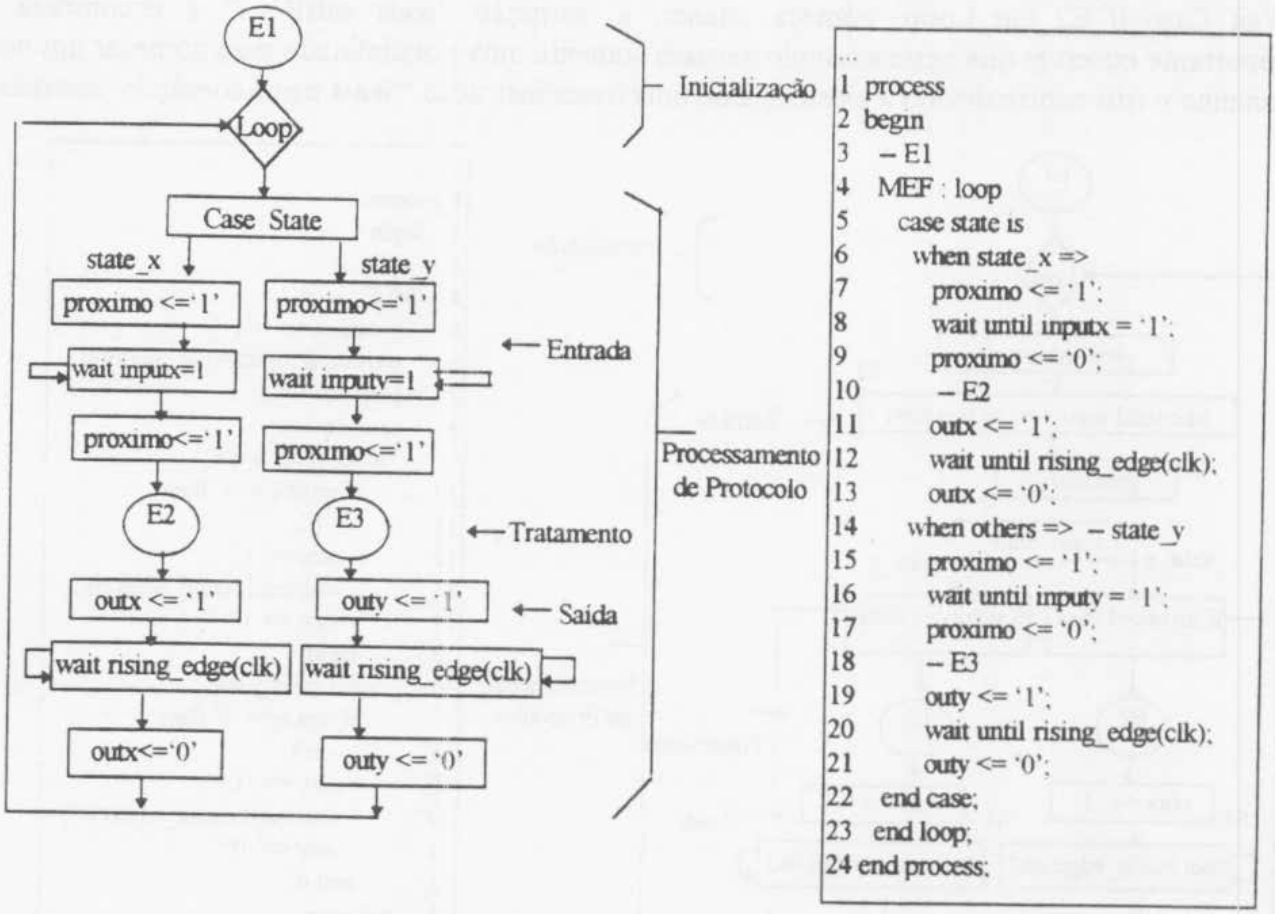


Figura 4 CFG e a descrição VHDL referente o Primeiro Estilo de Descrição

Transição	*Estado*	*Operação*	*próximo*	*Condição*
1	S1	3,7	S2	state = statex
2	S1	3,15	S4	state /= statex
3	S2	9,10,11	S3	inputx = '1'
4	S2	---	S2	inputx /= '1'
5	S3	13,7	S2	state = statex
6	S3	13,15	S4	state /= statex
7	S4	17,18,19	S5	inputy = '1'
8	S4	---	S4	inputy /= '1'
9	S5	21,7	S2	state = statex
10	S5	21,15	S4	state /= statex

Tabela 1 Tabela de Transição do Primeiro Estilo de Descrição

O segundo estilo de descrição para a implementação de uma máquina de estados de um protocolo é uma variação da primeira descrição. A única diferença é que a instrução "**wait until** *condição_entrada*" é posta fora da instrução **"case ..."**. A figura 5 apresenta a descrição VHDL e

o gráfico de controle de fluxo (CFG) resultante do segundo estilo. A tabela 2 apresenta a tabela de estados resultante do scheduling desta descrição usando o algoritmo DLS. Durante a geração de caminho pelo algoritmo DLS, pode ser observado que a geração do caminho corrente, por exemplo E1_Loop, é interrompido e um novo caminho, por exemple o caminho Wait_Case_If_E2_out_Loop, começa quando a instrução "**wait until ...**" é encontrada. É importante observar que neste exemplo existem somente uma possibilidade para começar um novo caminho e isto ocorre devido a existência de uma única instrução "**wait until** *condição_entrada* ".

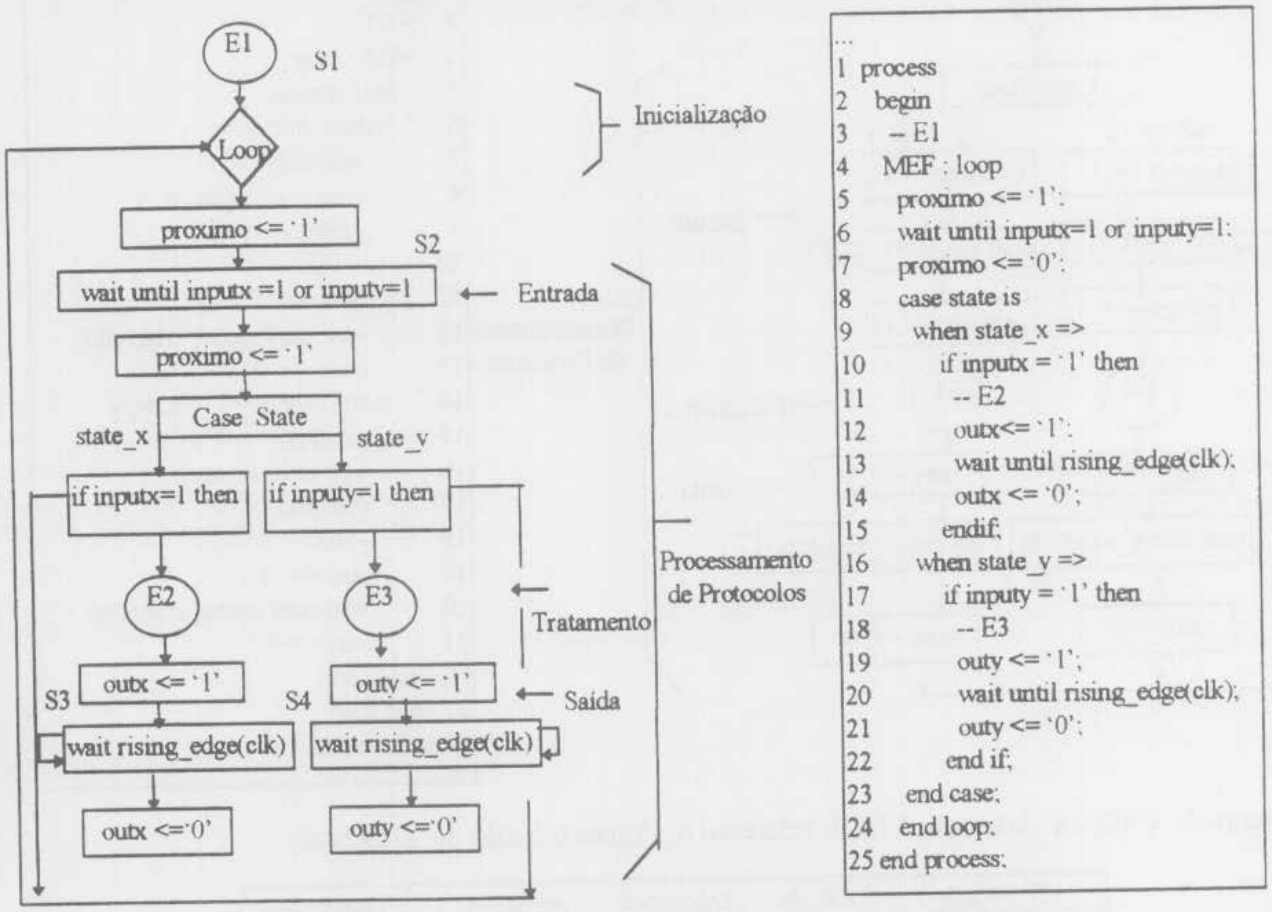
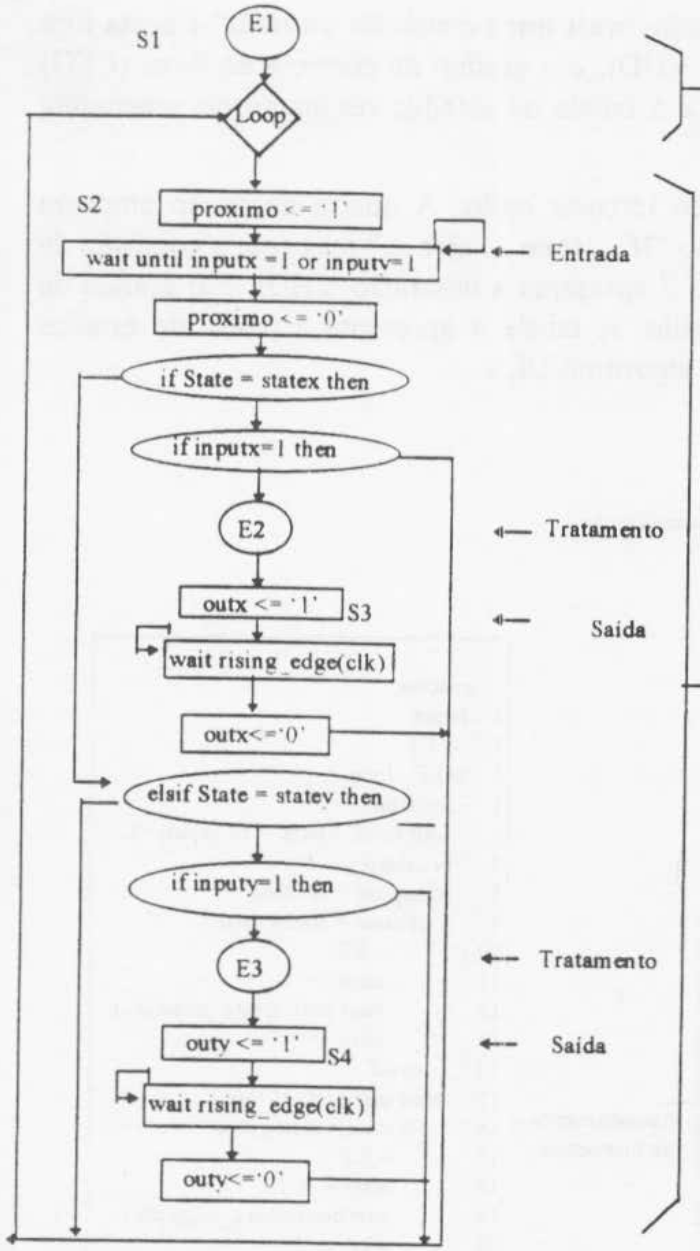


Figura 5 CFG e a descrição VHDL referente ao Segundo Estilo de Descrição

Transição	*Estado*	*Operação*	*Próximo*	*Condição*
1	S1	3,5	S2	--
2	S2	7,11,12	S3	state = statex and inputx = 1
3	S2	5	S2	state = statex and inputy = 1
4	S2	7,18,19	S4	state /= statex and inputy = 1
5	S2	5	S2	state /= statex and inputx = 1
6	S2	---	S2	(inputx /=1 or inputy /= 1)
7	S3	14,5	S2	---
8	S4	21,5	S2	---

Tabela 2 Tabela de Transição do Segundo Estilo de Descrição




```
...
1  process
2  begin
3   -- E1
4  MEF: loop
5  proximo <= '1';
6  wait until inputx=1 or inputy=1;
7  proximo <= '0';
8    if state = statex then
9      if inputx = '1' then
10       -- E2
11       outx <='1';
12       wait until rising_edge(clk);
13       outx <= '0';
14     endif
15   elsif state = statey then
16     if inputy = '1' then
17       -- E3
18       outy<= '1';
19       wait until rising_edge(clk);
20       outy <= '0';
21     end if;
22   end if;
23 end loop;
24 end process;
```

Figura 6 CFG e a descrição VHDL referente ao Terceiro Estilo de Descrição

Transição	*Estado*	*Operação*	*Próximo*	*Condição*
1	S1	3,5	S2	---
2	S2	7,10,11	S3	state = statex and inputx = 1
3	S2	5	S2	state = statex and inputy = 1
4	S2	7,17,18	S4	state = statey and inputy = 1
5	S2	5	S2	state = statey and inputx = 1
6	S2	5	S2	(inputx =1 or inputy = 1) and (state /= statex and state /= statey)
7	S2	---	S2	(inputx /=1 and inputy /= 1)
8	S3	13,5	S2	---
9	S4	20,5	S2	---

Tabela 3 Tabela de Transição do Terceiro Estilo de Descrição

O terceiro estilo de descrição emprega dentro da instrução " **process** " a instrução "**If ... then ... else ...**" em vez da instrução **"case ..."**. É utilizado na instrução "**If ... then ... else ...**" o

estado*(state)* como variável mais externa. A instrução "**wait until** *condição_entrada*" é posta fora do ninho de *ifs*. A figura 6 apresenta a descrição VHDL e o grafico de controle de fluxo (CFG) resultante do terceiro estilo. A tabela 3 apresenta a tabela de estados resultante do scheduling desta descrição usando o algoritmo DLS.

O quarto estilo de descrição é uma variação do terceiro estilo. A quarta descrição emprega também dentro da instrução " **process** " a instrução "**If ... then ... else ...**" mas com a condição de entrada *(input)* como teste mais externo. A figura 7 apresenta a descrição VHDL e o grafico de controle de fluxo (CFG) resultante do quarto estilo. A tabela 4 apresenta a tabela de estados resultante do scheduling desta descrição usando o algoritmo DLS.

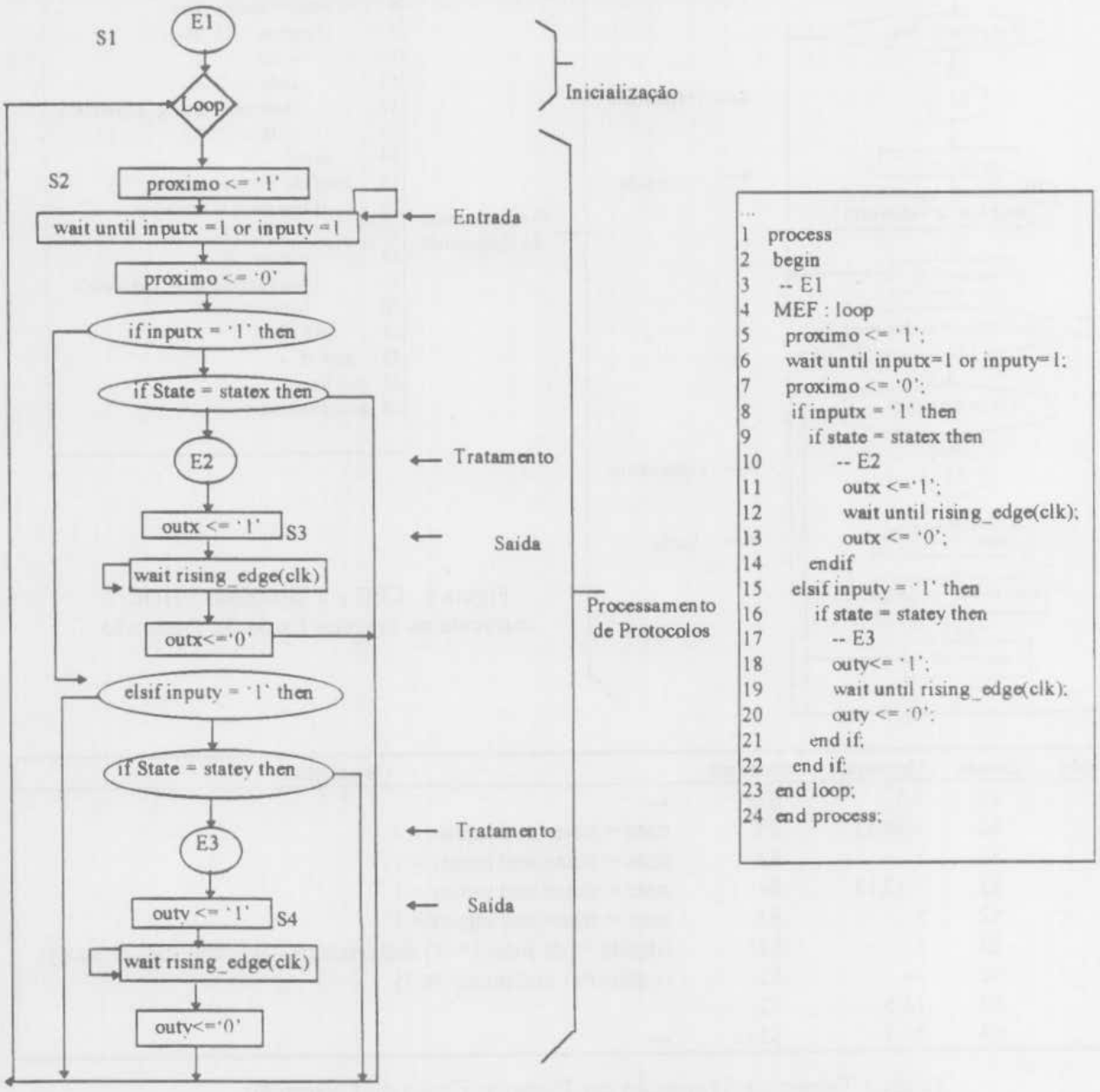


Figura 7 CFG e a descrição VHDL referente ao Quarto Estilo de Descrição

Transição	*Estado*	*Operação*	*Próximo*	*Condição*
1	S1	3.5	S2	---
2	S2	7.10.11.	S3	inputx =1 and state = statex
3	S2	5	S2	inputx =1 and state /= statex
4	S2		S4	inputy = 1 and state = statey
5	S2	7.17.18	S2	inputy = 1 and state /= statey
6	S2	5	S2	(inputx = 1 or inputy = 1) and (inputx /=1 and inputy /= 1)
7	S2	5	S2	(inputx /= 1 or inputy /= 1)
8	S3	13.5	S2	---
9	S4	20.5	S2	---

Tabela 4 Tabela de Transição do Quarto Estilo de Descrição

O quinto estilo de descrição é uma variação do segundo estilo de descrição. A única diferença é que os diferentes sinais de controle de saída recebem o valor zero imediatamente depois da instrução **"case ..."**. A figura 8 apresenta a descrição VHDL e o gráfico de controle de fluxo (CFG) resultante do quinto estilo. A tabela 5 apresenta a tabela de estados resultante do scheduling desta descrição usando o algoritmo DLS.

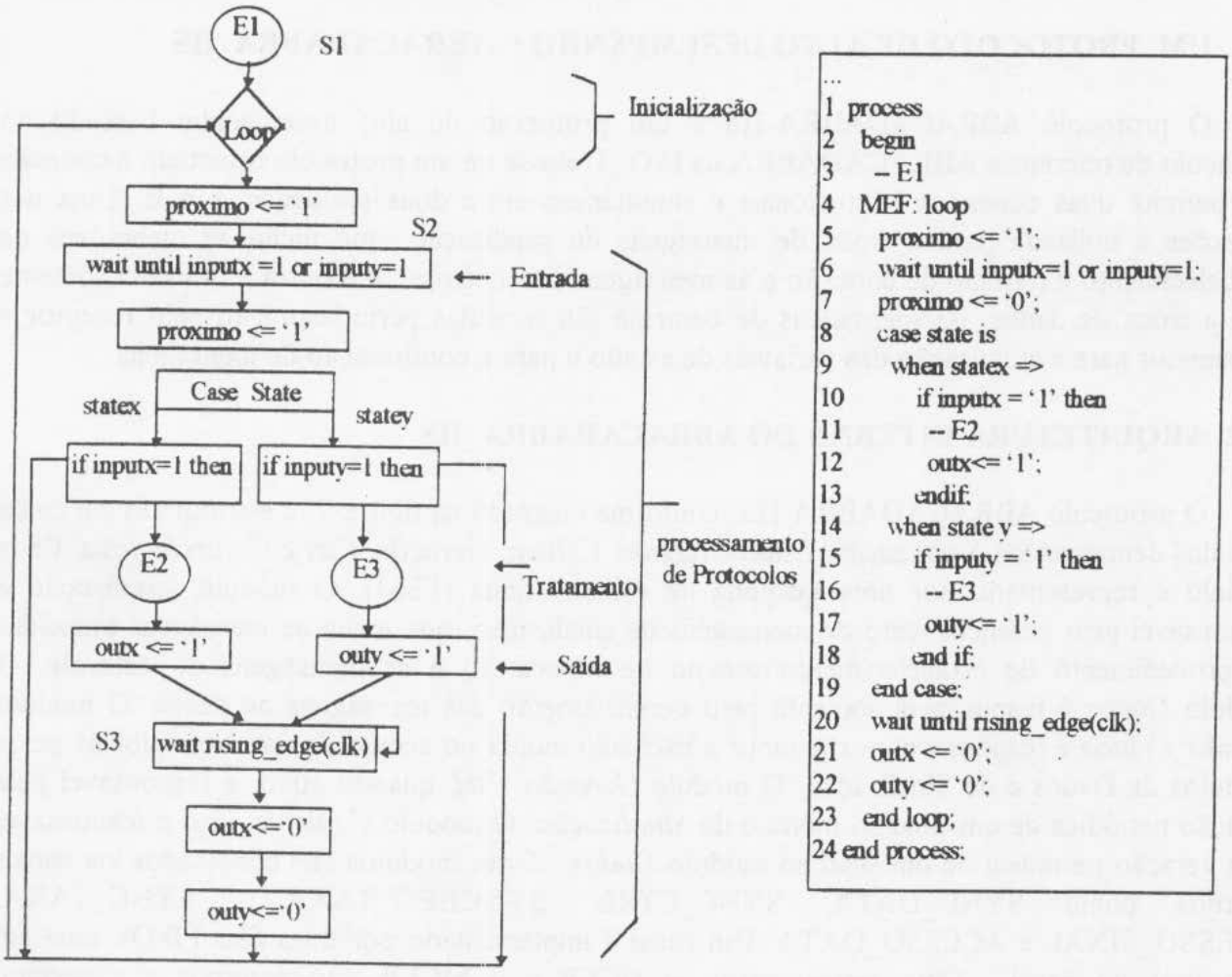


Figura 8 CFG e a descrição VHDL referente ao Quinto Estilo de Descrição

Transição	*Estado*	*Operação*	*Próximo*	*Condição*
1	S1	3.5	S1	---
2	S2	7.11.12	S2	state = statex and inputx =1
3	S2	7	S2	state = statex and inputy =1
4	S2	7.16.17	S2	(state /= statex and inputy =1
5	S2	7	S2	state /= statex and inputx =1
6	S2	---	S2	(inputx /=1) or (inputy /= 1)
7	S3	21.22.5	S2	---

Tabela 5 Tabela de Transição do Quinto Estilo de Descrição

Se estes diferentes estilos de descrição são sintetizados pelo AMICAL, diferentes resultados serão produzidos pelo DLS. Na próxima sessão é apresentado os diferentes resultados gerados pelo algoritmo DLS para diferentes estilos de descrição da máquina de estado de Sinalização.

IV. APLICAÇÃO A UMA ESPECIFICAÇÃO DE PROTOCOLOS

Os diferentes estilos que podem ser utilizados para descrever um protocolo em VHDL são discutidos com a ajuda de um exemplo, o protocolo ABRACADABRA_HS.

IV.1 UM PROTOCOLO DE ALTO DESEMPENHO : ABRACADABRA_HS

O protocolo ABRACADABRA-HS é um protocolo de alto desempenho baseado no protocolo de referência ABRACADABRA da ISO. Trata-se de um protocolo orientado a conexão que permite duas conexões bidirecionais e simultâneas entre duas entidades A e B. Uma das conexões é utilizada para a troca de mensagens de sinalização, que inclui as mensagens de estabelecimento e término de conexão e as mensagens de controle, e a outra é utilizada somente para a troca de dados. As mensagens de controle são trocadas periodicamente pelo receptor e transmissor para a atualização das variáveis de estado e para a confirmação de mensagens.

IV.2 ARQUITETURA INTERNA DO ABRACADABRA_HS

O protocolo ABRACADABRA-HS, conforme ilustrado na figura 9, é estruturado em cinco módulos denominados *Sinalização, Dados, Região_Crítica, Geração_Ctrl e Controle_taxa*. Cada módulo é representado por uma máquina de estados finita (FSM). O módulo *Sinalização* é responsável pelo gerenciamento de mensagens de sinalização, que inclui as mensagens utilizadas no procedimento de estabelecimento/término de associação e as mensagens de controle. O módulo *Dados* é responsável somente pelo gerenciamento das mensagens de dados. O módulo *Região_Crítica* é responsável por garantir a exclusão mútua no acesso de variáveis globais pelos módulos de Dados e de Sinalização. O módulo *Geração_Ctrl*, quando ativo, é responsável pela geração periódica de um sinal ao módulo de *Sinalização*. O módulo *Controle_taxa* é responsável pela geração periódica de um sinal ao módulo *Dados*. Estes módulos são conectados via canais internos como SYNC_DATA, SYNC_CTRL, SYNCBEG_TAXA , SYNC_TAXA, ACESSO_SINAL e ACESSO_DATA. Um canal é implementado por duas filas FIFOs, uma em cada lado do canal. Dois destes canais, o UCEP e o MCEP, são externos e conectam, respectivamente, o nível superior e o nível inferior.

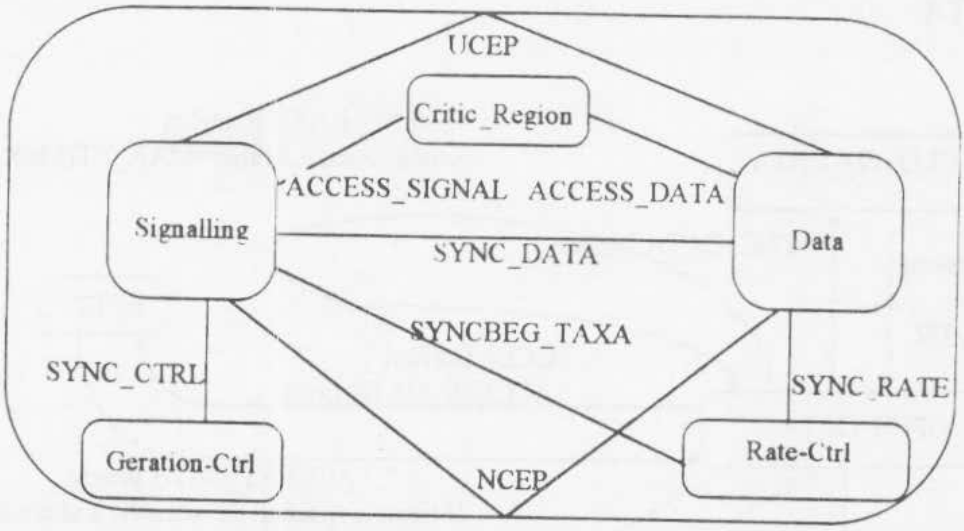


Figura 9 Arquitetura do provedor do ABRACADABRA_HS.

A seguir, a título de ilustração, apresenta-se como exemplo a máquina de estados do módulo Data. Os resultados apresentados na próxima seção são no entanto relativos ao módulo de sinalização, um dos módulos mais complexos do protocolo ABRACADABRA_HS.

A máquina de estados do módulo Data, conforme ilustrado na figura 10, encontra-se inicialmente no estado CLOSED_DATA, significando que está fechada para comunicação de dados. Esta máquina de estados mudará o estado de CLOSED_DATA para OPEN_DATA após o recebimento de um pedido de conexão (DConReq) através do canal SYNC_DATA. Neste estado, a máquina de estados está ponta para transmitir ou receber mensagens de dados. O inverso, isto é, a mudança do estado OPEN_DATA para CLOSED_DATA ocorrerá após a recepção de um pedido de desconexão (DDisReq) através do canal SYNC_DATA.

Esta máquina de estados usa algumas variáveis no estado OPEN_DATA cujo acesso é controlado por outra máquina de estados denominada Região_Crítica. Para ter acesso a estas variáveis, a máquina de estados Data emite uma primitiva Lock e muda para um dos quatro estados possíveis dependendo da primitiva (DatReq, RetxReq, UnitInd). O estado EST1 é alcançado se a primitiva RetxReq é recebida pelo canal SYNC_DATA. O estado EST2 é alcançado após o recebimento da primitiva Datreq pelo canal UCEP. O estado EST3 é alcançado após o recebimento da primitiva UnitInd pelo canal MCEP. O estado EST4 é utilizado para retransmitir mensagens de dados. Em todos estes estados (EST1, EST2, EST3, EST4), a máquina de estados está esperando a primitiva Ready pelo canal ACCESS_DATA.

Ao receber a primitiva *Data Request (DatReq)* pelo canal UCEP, a máquina de estados de Dados evolui do estado OPEN_DATA para o estado EST2, emite a primitiva *Lock* através do canal ACCESS_DATA para solicitar o acesso de algumas variáveis globais controlada pelo módulo Região_Crítica, e inicia a espera da primitiva Ready. Quando esta máquina recebe a primitiva *Ready* através do canal ACCESS_DATA no estado EST2, transmite mensagens de dados (DT) se existe espaço na janela e se há crédito disponível, emite a primitiva *VarInd* através

do canal ACCESS_DATA para liberar o uso destas variáveis globais e, finalmente, evolui para o estado OPEN_DATA.

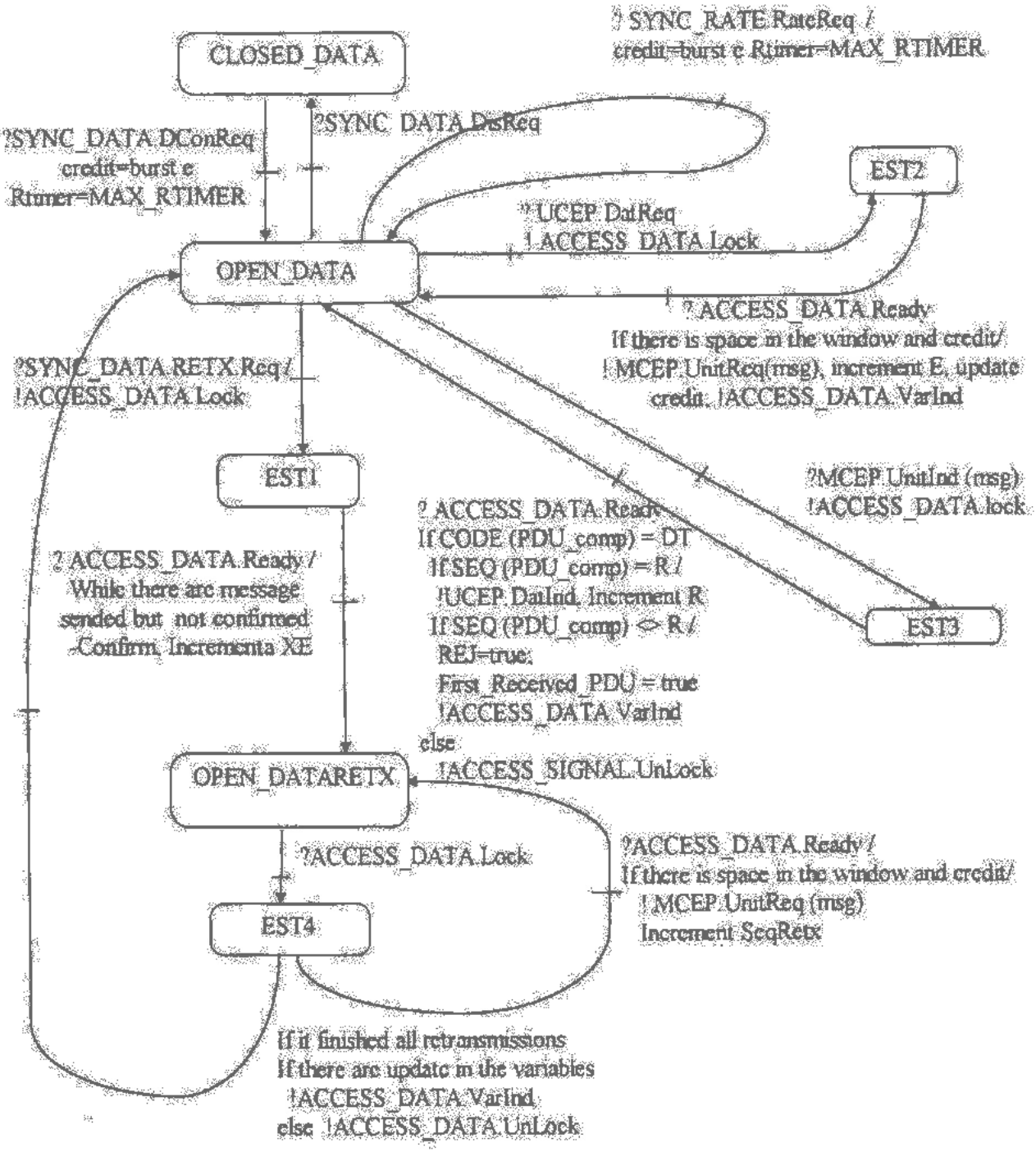


Figure 10 Máquina de Estados do Módulo Data

Uma especificação completa do protocolo ABRACADABRA_HS pode ser encontrada em [14].

IV.3 RESULTADOS

Esta sessão apresentará os resultados dos diferentes estilos de descrição do módulo de Sinalização, o mais complexo do ABRACADABRA_HS. Todos estes diferentes estilos estão descritos em VHDL. Os resultados foram obtidos usando o algoritmo DLS (Dynamic Loop Scheduling). Para cada descrição, a tabela 6 apresenta os resultados obtidos pelo algoritmo. As primeiras duas colunas apresentam o número de caminhos e o número de estados gerados por cada descrição. As próximas colunas mostram o número de operações, o número de Loops, o número de waits, o número de ifs e o número de cases. As últimas duas colunas dão o número de linhas de código da descrição de entrada em VHDL e a área do controlador dado em número de transistores.

Descrição	*Caminho*	*Estados*	*Operações*	*Loops*	*Wait*	*Ifs*	*Case*	*Linhas*	*Área*
Primeira	614	88	400	12	54	34	1	653	206.183
Segunda	138	70	406	12	45	42	1	622	46.473
Terceira	136	69	413	12	44	51	0	625	45.904
Quarta	137	69	451	12	44	52	1	616	46.237
Quinta	104	36	325	12	11	42	1	557	35.641

Tabela 6 Resultados do algoritmo DLS

As cinco descrições utilizam estruturas de controle complexas como instruções de "**wait ...**" com várias condições, chamadas de rotinas, loops, ninhos de instruções ifs e ninhos de loops. Todas estas descrições levaram menos que um segundo para serem executadas em uma estação de trabalho SPARC II. Pode ser observado da saída do algoritmo DLS que a Quinta descrição possui os melhores resultados devido a redução significativa do número de instruções de "**wait until** *condição_entrada*" e de "**wait until** *rising_edge (clk)*", reduzindo consequentemente o número de caminhos gerados, e , assim, o custo de computação. Isto é devido a construção do algoritmo DLS que gera um novo caminho cada vez que encontra uma instrução de wait. Também pode ser observado que a Segunda e a Terceira descrição possuem ótimos resultados devido a redução também significativa do número de instruções de "**wait until** *condição_entrada*".

A Segunda e a Terceira descrição produzem resultados similares já que a instrução "if ... then ... else ..." e a instrução "case ..." são construtores similares. Um outro ponto interessante de observar são os resultados da terceira e da Quarta descrição. Na Terceira descrição, a instrução "**If ... then ... elsif ... end if** " mais externa testa a variável de estado e no Quarto estilo de descrição a instrução "**If ... then ... elsif ... end if** " mais externa testa as condições de entrada. Os resultados mostram que o Terceiro é melhor do que o Quarto estilo de descrição porque neste exemplo o processamento é mais dependente do estado do que das condições de entrada reduzindo consequentemente o número de caminhos gerados e o número de operações. Entretanto, podem existir situações no qual o processamento é mais dependente das condições de entrada do que da variável de estado.

Com o objetivo de reduzir o numero de caminhos e consequentemente o numero de transições a seguinte metodologia deve ser adotada:

1. Identificar a FSM do protocolo e seus sinais. O modelo de uma FSM corresponde a um processo em VHDL. O processo é composto por dois loops: um loop externo que modela a reinicialização do processo e um loop interno que modela a FSM do protocolo.
2. A instrução "**wait until** *condição_entrada* " é posta no inicio do loop mais interno (INNER_LOOP) onde os sinais de controle são testados.
3. Para evitar duplicações no código onde as exceções foram especificadas, sinais de exceções (reset , situações de erros) são criados e acrescentados na lista sensitiva da instrução de wait. Os sinais de exceções são imediatamente verificados após a instrução de wait.
4. A máquina de estados é construída utilizando a instrução "**case ...**" ou a construção **"if ... then ... elsif ... end if "**. Em ambas as soluções, a escolha entre testar inicialmente o estado ou as condições de entrada depende se o processamento é mais dependente do estado ou das condições de entrada.
5. Após o uso da instrução "**case ...**" ou da construção "**if ... then ... elsif ... end if** " , os diferentes sinais de controle recebem o valor zero.

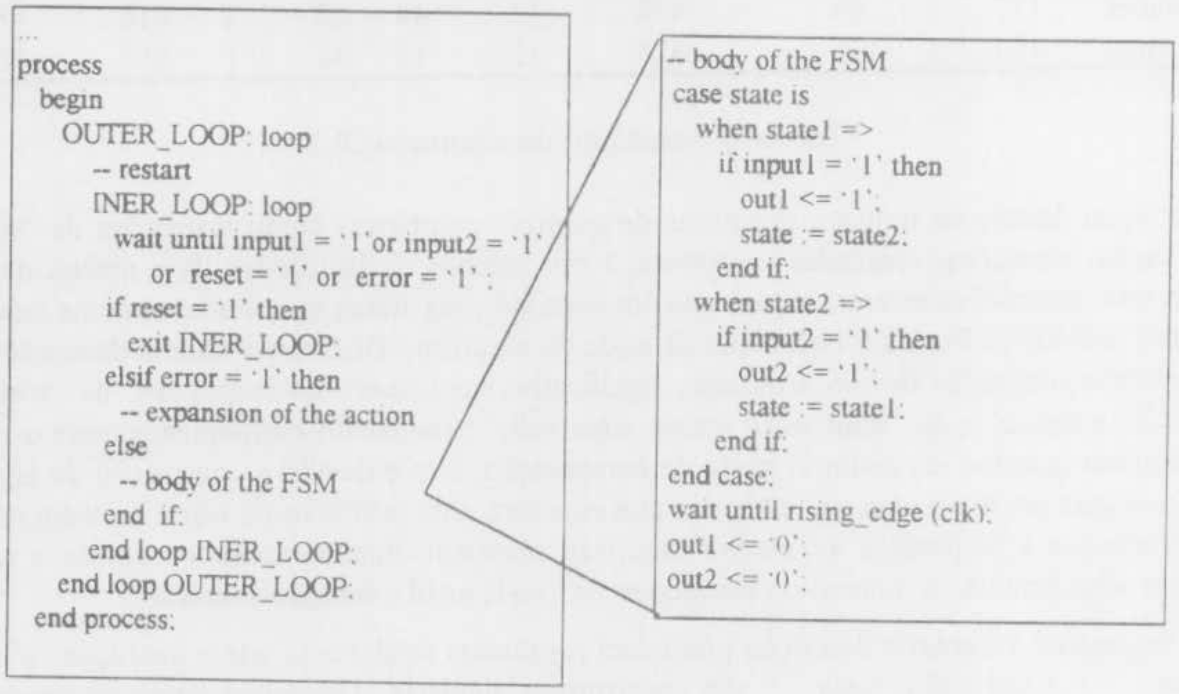


Figura 11 Modelo de uma FSM de Protocolo

V. CONSIDERAÇÕES FINAIS

A linguagem Estelle é uma técnica de descrição formal usada para a especificação, simulação e verificação de protocolos. VHDL é uma linguagem de descrição de hardware usada para a descrição, simulação e síntese de hardware. Para um projetista de protocolo, Estelle é mais fácil de ser utilizada para especificar e analisar os protocolos de comunicação do que o VHDL, já que Estelle possui o conceito de máquina de estados e de fila FIFO embutido na linguagem. Por outro

lado, VHDL é melhor do que Estelle para lidar com problemas relacionados com o tempo de execução e para relacionar com as ferramentas de síntese já existentes.

A estratégia proposta para o mapeamento de Especificações Estelle em descrições VHDL é adequada à simulação. Os procedimentos que tornam este mapeamento adequado à síntese quando se usa o ambiente AMICAL são também indicados.

Este artigo apresentou os diferentes estilos em que protocolos podem ser descritos em VHDL e as restrições que determinam o mais adequado para as ferramentas de síntese. Esta análise foi feita com auxílio de um exemplo, o protocolo ABRACADABRA_HS. As principais contribuições desta análise são:

- o uso de múltiplas condições de entrada em uma única instrução de "**wait until** *condição_entrada*" na descrição de entrada, o que permite uma redução significativa na complexidade da tabela de transição, através da redução do número de caminhos gerados;
- a escolha do estado (state) como variável mais externa, quando o processamento é mais dependente do estado do que das condições de entrada. Isto permite reduzir o número de estados gerados;
- a criação de sinais de exceção de forma que o código de exceção seja tratado imediatamente após a instrução de "wait until ...", o que permite evitar a duplicação de código de exceção.

Como perspectivas de trabalhos futuros podemos citar:

- a obtenção de novos estilos de descrição de protocolos que sejam eficientes para os processos de HLS no sentido da obtenção de estruturas de hardware otimizadas para a implementação de protocolos;
- a inclusão de soluções que permitam o co-design de hardware/software utilizando o mapeamento proposto de Estelle para VHDL e o ambiente de CAD desenvolvido pela COPPE.

VI. REFERÊNCIAS

[1] Fraser, A. Marshall, W. *Data Transport in a Byte Stream Network*. IEEE Journal on Selected Areas in Communications, Vol. 7, NO.7, September, 1989.

[2] Strayer W. et al, *XTP: The Xpress Transfer Protocol*. Addison-Wesley, 1985.

[3] Netravali, A. et al *Design and Implementation of a High-Speed Transport Protocol*. IEEE Trans. on Communications, Vol. 38, No. 11, pp. 2010-24, Nov.1990.

[4] Doshi, B. and Johri, P. *Communications Protocols for High Speed Packet Networks*. Computer Networks and ISDN Systems, No 24, pp. 243-73. 1992.

[5] Doeringer, W. et al *A Survey of Light-Weight Transport Protocols for High-Speed Networks*. IEEE Trans. Comm., vol. 388, No. 11, pp. 2025-38., Nov.1990.

[6] Budkowski, S. and Dembinski, P. *An Introduction to Estelle : A Specification Language for Distributed Systems*. Computer Network and ISDN System 14, pp. 3-23, Elsevier, 1987.

[7] Chanson et al, *On Tools Supporting the Use of Formal Description Techniques in Protocol Development*. Computer Networks and ISDN Systems 25,pp.723-39,Elsevier, 1993.

[8] Gajski, D. et al, *High Level Synthesis : Introduction to Chip and System Design*. Kluwer, 1992.

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

[10] Navabi, Z. *A High-Level Language for Design and Modeling of Hardware*. J. System Software 18, pp. 5-18, 1992.
[11] Courtiat, J. *Estelle* A Powerful Dialect of Estelle for OSI Protocol Description*, Intern. Workshop on Protocol Specification Verification and Testing, France, 1988.
[12] Park I., O'Brien K., Jerraya A.A., AMICAL: Architectural Synthesis Based on VHDL", in G. Saucier, J. Trilhe (eds), "Synthesis for Control Dominated Circuits", North-Holland, 1993.
[13] Rahmouni M., O'Brien K.and Jerraya A. A. *A Loop-based Scheduling Algorithm For Hardware Description Languages'*, Parallel Processing Letters, World Scientific Publishers, Vol. 4 No 3, pp 351-364, 1994.
[14] Pirmez L., *Uma Metodologia para a Implementação de Sistemas Distribuídos em Hardware a partir de uma Descrição Formal*, Documento apresentado para o exame de tema de tese a COPPE Elétrica, Abril de 1995.
[15] Pirmez L., Carneiro F., Pedroza A., Mesquita C. *A Methodology to Developed Integrated Circuits from Estelle Specifications.* 38th Midwest Symposium on Circuits and Systems, Brazil, 1995
[16] Pirmez L., Pedroza A., Mesquita C. *A Methodology to the Implementation of Distributed System in Hardware from a Formal Description*, IFIP WG. 6.1 Fifteenth International Symposium on Protocol Specification, Testing and Verification, Varsow, Poland, 1995.
[17] Wytrebowicz J., *Hardware Specification Generated from Estelle*, IFIP WG. 6.1 Fifteenth International Symposium on Protocol Specification, Testing and Verification, Varsow, Poland, 1995.
[18] Kission P., Ding Hong and Jerraya A. , *Structured Design Methodology for High-Level Design*, 31st ACM/IEEE Design Automation Conference, 1994.
[19] Camposano R., *Path-Based Scheduling for Synthesis*, IEEE T. CAD, Vol 10(1), pp 85-93, January 1991.
[20] Rahmouni M., Jerraya A.A., *Formulation and Evaluation of Scheduling Techniques for Control Flow Graphs*, Proc. EURODAC'95, Brighton, UK, September 1995
[21] C. D. Kloos et all, "VHDL generation from a timed extension of the formal description technique LOTOS within the FORMAT project", Microprocessing and Microprogramming 38 (1993) 589-596, North-Holland.
[22] Jerraya, A. and O'Brien, K. *SOLAR: an intermediate format for system-level medeling and synthesis.* Chapter 7 in Codesign computer aided software/hardware engineering , IEEE press, (1994),145-175.
[23] Jerraya, A. and O'Brien, K. *Linking System Design Tools and Hardware design Tools.* Proc. VHDL'93, (1993) 331-338.
[24] Vahid, F. et al. *SpecCharts: A language for System Level Design.* Proc. VHDL'91, 145-155.
[25] Kission P., Ding H., Jerraya A. *VHDL Based Design Methodology for Hierarchy and Component Re-Use at Behaviora Level*, EURODAC'95, Brighton, UK, September 1995.
[26] Kission P., Closse E., Bergher L., Jerraya A. Industrial Experimentation of High-Level Synthesis, EuroDAC-EuroVHDL, 1993.