UNIVERSIDADE ESTADUAL DE CAMPINAS
Faculdade de Engenharia Elétrica
Departamento de Semicondutores, Instrumentos e Fotônica

MODELAMENTO EM VHDL DE UM SISTEMA DE SIMULAÇÃO DE CIRCUITOS

Por: Wilfredo Machaca Luque
Orientador: Prof. Dr. Furio Damiani

Dissertação submetida como requisito parcial para a obtenção do título de Mestre em Engenharia Elétrica.

Campinas, Novembro de 1994.

Dedico este trabalho

Aos meus pais
Benito e Elvira

À minha companheira
Lucy Doris

Ao meu filho, pela felicidade
de sua chegada

Agradecimentos

Agradeço ao Prof. Furio Damiani pela orientação e confiança no transcorrer do desenvolvimento deste trabalho.

Ao Prof. Norian, do DCCE/IBILCE/UNESP, pelas frutíferas discussões sobre o sistema ABACUS.

À CAPES pelo apoio financiero.

Aos colegas do DSIF: Eliane, Margareth, Carla, Ilka, Dahge, Walter, Marcelo, Nivaldo, Augusto, Alfredo, Juan Carlos, Thomas e Yasushi pelo agradável ambiente de trabalho proporcionado.

À minha companheira Lucy pela confiança, compreensão e apoio prestado em todo momento.

Aos meus pais pela sua incansável dedicação e pelo apoio em todas as etapas da minha vida.

Aos meus amigos: Yemiko, Delson, José e Ana pelos laços de amizade criados e por tornarem a estada no Brasil muito agradável.

Resumo

O ABACUS é um sistema voltado à simulação de circuitos de alta escala de integração que utiliza um arranjo de processadores dedicados e que diferente das metodologias convencionais de simulação evita a resolução do sistema de equações algébrico-diferenciais não lineares que descrevem os circuitos. Neste trabalho apresenta-se o modelamento na linguagem de descrição de *hardware* VHDL desse sistema em dois níveis de abstração, comportamental e de transferência entre registradores, com a finalidade de simulá-lo detalhadamente, objetivando a sua implementação física.

Como produto da simulação apresentam-se resultados sobre o dimensionamento dos canais de comunicação entre os processadores do arranjo e sobre os tempos de execução dos modelos dos dispositivos, os quais são de grande importância para o modelamento do sistema ABACUS nos níveis seguintes de abstração.

Índice

Capítulo 1

Introdução

A simulação de circuitos cumpre um papel muito importante nas fases de projeto e verificação de Circuitos Integrados (CI's). De fato, ela é uma das ferramentas de Projeto Assistido por Computador (PAC) mais utilizadas, consumindo uma grande parcela do tempo de processamento necessário ao projeto de CI's [Sal89, Cox91].

O processo de fabricação de um circuito integrado representa um alto custo do ponto de vista econômico e de tempo, por isso o circuito é simulado antes para verificar a sua funcionalidade e obtenção de uma informação detalhada de sua temporização. A importância da simulação deve-se à sua confiabilidade e capacidade de prever resultados, mesmo em circuitos extremamente complexos. Não obstante, o sempre crescente número de transistores por chip, tem exigido o desenvolvimento de novas ferramentas e metodologias para a simulação de circuitos.

Uma nova metodologia para a simulação a nível de circuitos, motivo de nosso trabalho, foi proposta em uma tese de doutorado no DSIF/FEE/UNICAMP. Essa metodologia visa simular circuitos de alta escala de integração, utilizando para tanto um arranjo de processadores dedicados onde os cálculos são assíncronos e controlados apenas pelo fluxo de informações gerado a partir das condições de contorno [Mar90a, Mar92].

Por outro lado, os circuitos de grande escala de integração tais como VLSI[1] e ULSI[2], especialmente os microprocessadores recentes ou mesmo nosso sistema em estudo, seriam muito difíceis de se desenvolver sem o auxílio de alguma ferramenta PAC. A VHDL[3] vem-se tornando uma ferramenta, no processo de projeto, muito importante devido à sua capacidade de suporte de uma ampla faixa de abstrações, desde descrições em nível de portas até descrições em nível de algoritmo, razão pela qual foi escolhida para descrever o nosso sistema.

---

[1] *Very Large Scale Integration*

[2] *Ultra Large Scale Integration*

[3] VHSIC *Hardware Description Language* (VHSIC - *Very High Speed Integrated Circuits*)

1.1 Objetivos

O presente trabalho visa modelar, nos níveis comportamental e de transferência entre registradores, o Simulador de Circuitos Baseado em *Hardware* (*Hardware-Based Circuit Simulator* - ABACUS)[Mar90b] utilizando a linguagem de descrição de *hardware* VHDL :
- Para realizar a sua simulação detalhada.
- Para elaborar o seu modelamento objetivando a sua implementação física.

1.2 Ferramentas Clássicas para Simulação de Circuitos

As ferramentas de simulação de circuitos tais como o SPICE [Mea88] e o ASTAP [Wee73], oferecem uma ampla variedade de tipos de análises, que incluem análise DC, análise transiente no domínio do tempo, análise AC no domínio da freqüencia e análise de ruído e distorsão. Desses tipos de análise, a análise transiente no domínio do tempo, é a que tem maior custo computacional.

Os programas como o SPICE originalmente foram projetados para simular circuitos contendo no máximo 100 transistores. Porém, devido à necessidade o SPICE tem sido utilizado para simular circuitos com mais de 10.000 transistores; obviamente, com um custo computacional altíssimo.

Para melhorar o desempenho dos simuladores sem afetar a exatidão dos cálculos, foram propostos uma variedade de novos algoritmos para a solução da matriz de equações algébrico-diferenciais não lineares que descrevem os circuitos. Particularmente tem-se as técnicas baseadas na relaxação, tais como relaxação de formas de onda [Lel82, Whi86] e análise iterativa de temporização [Sal83].

Recentemente com a disponibilidade de um grande número de computadores com multiprocessadores, o enfoque das pesquisas tem mudado para a aplicação de técnicas de processamento paralelo, obtendo assim, reduções no tempo de processamento. Tanto a técnica padrão como as técnicas baseadas na relaxação foram consideradas para a paralelização [Deu84].

O desempenho dos simuladores também pôde ser melhorado com o uso de processadores vetoriais[Vla82]. O SPICE tem sido "vetorizado" por várias empresas de *software*[Gre86], não obstante, isto significa uma reformulação do algoritmo e reorganização das estruturas de dados. O processamento vetorial tem sido limitado por vários fatores como, por exemplo a depêndencia com relação à disponibilidade dos dados na computação dos resultados dos modelos de circuitos. Com o processamento vetorial tem-se conseguido velocidade de processamento da ordem de 2 a 4 vezes maior do que a das técnicas padrão. Ainda assim, estas melhoras têm sido insuficientes para a simulação de circuitos de alta escala de integração.

Adicionalmente têm surgido sistemas com um *hardware* específico para a simulação de circuitos.

1.3 Descrição da Metodologia em Estudo

Nas metodologias clássicas de simulação de circuitos, o custo computacional maior está no cálculo da matriz de equações que representa o circuito a ser simulado.

Na metodologia em estudo, foi proposta a utilização de um *hardware* dedicado e uma nova metodologia de simulação, que consiste em evitar a solução do sistema de equações algébrico-diferencias não lineares, que em geral são muito grandes. Esta metodologia utiliza um arranjo especial de processadores sobre o qual é mapeado o circuito, onde cada processador ou um conjunto de processadores simulam um elemento de circuito. Cada processador do arranjo é um elemento de processamento de modelos. Existe um processador específico que se encarrega da interação com o usuário, do pré-processamento do circuito e do gerenciamento do arranjo. Um sistema programável de barramentos e chaves reflete a topologia do circuito[Mar90c].

1.4 Organização da Tese

No Capítulo 2 faz-se uma descrição do sistema ABACUS, estudando-se a sua arquitetura. São descritas também as funções do gerenciador e do arranjo de processadores, assim como das unidades de processamento de modelos e dos elementos de interconexão.

O Capítulo 3 dedica-se à ferramenta de modelamento VHDL utilizada para a simulação do sistema em estudo, dando-se uma abordagem visando síntese. Descreve-se também o ambiente de trabalho *Falcon Framework* da *Mentor Graphics* empregado na presente tese.

No Capítulo 4 trata-se o modelamento em VHDL do sistema ABACUS. Em um primeiro nível de abstração descreve-se o sistema de forma geral, preocupando-se mais com o funcionamento. Faz-se também a sua respectiva validação por meio da simulação. Em um segundo nível de abstração descreve-se o sistema com mais detalhamento, conferindo logo os resultados da simulação deste nível com os do primeiro nível.

Finalmente, são apresentadas considerações sobre os principais resultados do trabalho.

O Apêndice se constitui das listagens das descrições em VHDL do sistema ABACUS dos dois níveis de abstração realizados.

Capítulo 2

Descrição do Sistema ABACUS

Neste capítulo elabora-se uma descrição pormenorizada do sistema Simulador de Circuitos Baseado em *Hardware* (ABACUS), a qual será ponto de partida para seu modelamento em VHDL[1]. Descreve-se a sua arquitetura, assim como seu princípio de funcionamento. Descreve-se também a arquitetura de cada um dos blocos que o conformam, os processos que governam o processador gerenciador, as unidades de processamento de modelos e as chaves utilizadas na rede de interconexão. Com isto, mostra-se a conformação do arranjo de processadores em conjunto com a rede de interconexão. Aborda-se também assuntos sobre os modelos de alguns dispositivos e de seus algoritmos de processamento utilizados nos elementos de processamento de modelos (MPHs[2]). No final do capítulo é realizada uma especificação da arquitetura ABACUS.

2.1 Arquitetura do Sistema ABACUS

Como foi mencionado no capítulo anterior, a arquitetura do ABACUS é formada por um processador gerenciador, um arranjo de processadores de modelos e um conjunto de barramentos e chaves programáveis (rede de interconexão), vide Fig. 2.1.

O funcionamento do sistema pode ser explicado da seguinte maneira: o processador gerenciador recebe a descrição do circuito a ser simulado na forma de *netlist* além dos parâmetros iniciais da simulação. Em seguida, faz uma verificação topológica do circuito, identificando os elementos e a análise a ser feita. Mapeia o circuito no arranjo de processadores e faz o roteamento na rede de interconexão programável de forma a refletir a topologia do circuito. A seguir, transfere os parâmetros iniciais da simulação de cada elemento do circuito aos processadores do arranjo correspondentes. Então, inicia-se o processo de simulação começando com o primeiro instante. Enquanto isso, o processador gerenciador aguarda pelos resultados, controlando ao mesmo tempo a convergência do arranjo. Quando o arranjo converge em uma solução, o gerenciador lê os resultados,

---

[1] *VHSIC Hardware Description Language*

[2] *Model Processing Hardware-element*

acrescenta em um passo o tempo de simulação e inicia o seguinte instante de simulação. Este processo é repetido até processar todos os instantes solicitados pelo usuário[Mar92].

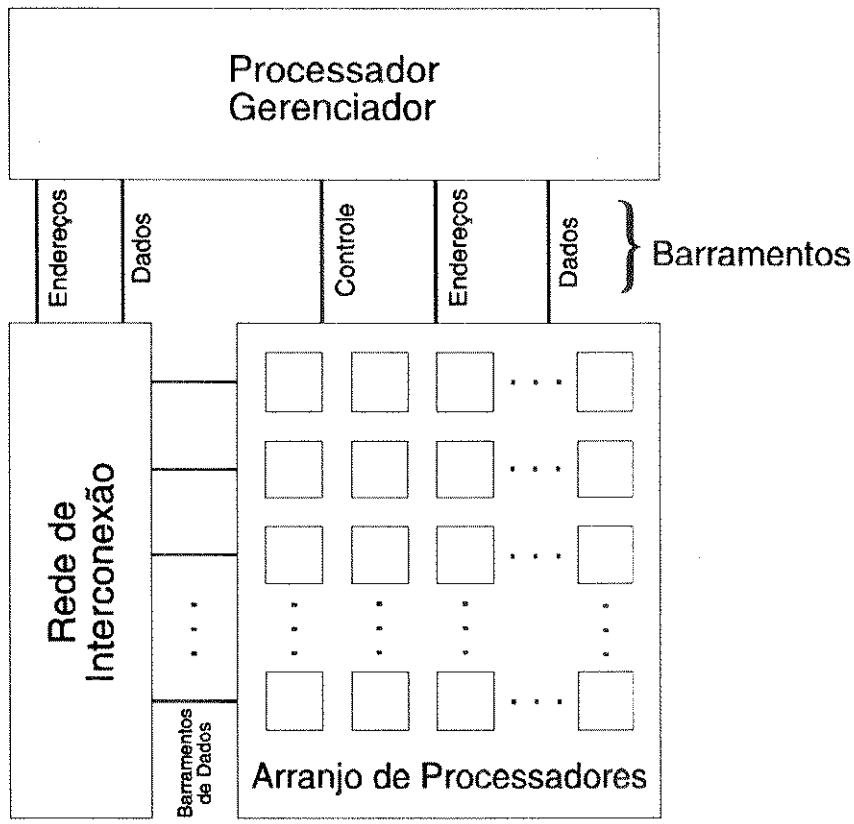


Fig. 2.1 Arquitetura básica do sistema ABACUS.

Paralelamente à configuração dos processadores no arranjo, para cada circuito a ser simulado, o processador gerenciador transfere no instante inicial a configuração correspondente à interconexão dos elementos do circuito para o sistema de barramentos e chaves programáveis. Com isto, é possível estabelecer uma configuração diferente para cada novo circuito mapeado no arranjo.

Cada processador do arranjo é um Elemento de Processamento de Modelos ou MPH (***Model Processing Hardware-element***) e básicamente emula um dispositivo por análise, trabalhando assíncronamente e governado apenas pelo fluxo de dados que a ele chega, permitindo assim que o arranjo de MPHs simule o comportamento do circuito.

O estado inicial de todos os MPHs é desconhecido. No momento que o processo de simulação começa, apenas os processadores que estiverem simulando elementos que estão ligados à fontes de alimentação ou estímulo terão eventos nas suas entradas. Portanto serão computadas as suas respostas e colocadas nas suas saídas, de tal forma que os MPHs que estiverem conetados a eles possam lê-los e calcular seus resultados. Enquanto isso, os

primeiros aguardarão por um novo evento nas suas entradas, e assim por diante.

2.2 O Processador Gerenciador

O algoritmo de simulação usado no simulador em estudo, executado pelo processador gerenciador, consta de quatro processos: *Host Control Process* (HCP), *Host Input Process* (HIP), *Host Output Process* (HOP) e *Array Management Process* (AMP). Além destes 4 processos existe outro chamado de MPH, que é executado em cada um dos elementos de processamento de modelos. A seguir descreve-se cada um deles.

2.2.1 O Processo HCP (*Host Control Process*)

Este processo controla o simulador ABACUS operando em nível de sistema. Encarrega-se do gerenciamento assíncrono dos demais processos, iniciando e encerrando a operação do ambiente de simulação, ou seja, ele ativa e desativa cada um dos processos mencionados quando for necessário. O algoritmo mostra-se na Fig. 2.2.

```
1. Iniciar o processo de simulação.
2. Iniciar o processo HIP.
3. Iniciar o processo AMP.
4. Iniciar o processo HOP.
5. Terminar quando todos os processos estiverem concluídos.
```

Fig. 2.2 Algoritmo do processo HCP.

2.2.2 O Processo HIP (*Host Input Process*)

Este Processo tem como função receber os dados correspondentes à descrição do circuito a ser simulado, expresso na forma de *netlist*, assim como os parâmetros iniciais da simulação. Tem também a tarefa de organizar esses dados para o posterior tratamento pelos demais processos do simulador. Os passos do algoritmo são mostrados na Fig. 2.3.

```
1. Lêr o arquivo que contém os dados de entrada.
2. Identificar os elementos do circuito a simular.
3. Identificar a análise a ser feita.
4. Montar a base de dados do circuito.
5. Montar o grafo de interconexão do circuito.
6. Verificar o número de MPHs disponíveis no arranjo.
7. Se necessário particionar o circuito em sub-circuitos, de acordo com
   o número de MPHs detectado (passo 6) e mantendo os nós fortemente
   acoplados no mesmo sub-circuito.
8. Preparar o circuito (ou cada sub-circuito) para a simulação e
   transferi-lo ao processo AMP. Avisar ao processo AMP de tal
   transferência.
9. Terminar o processo HIP e avisar ao processo HCP.
```

Fig. 2.3 Algoritmo do processo HIP.

A tarefa de particionamento do circuito (passo 7 do algoritmo HIP) em sub-circuitos, deve ser realizada tomando-se o cuidado de não seccionar em nós fortemente acoplados, mantendo-os desta maneira dentro de um mesmo módulo ou sub-circuito. Este procedimento com o uso do processamento paralelo na simulação de circuitos de alta escala de integração, tem sido estudado amplamente e se tornado uma operação muito utilizada[Sav93].

2.2.3 O Processo AMP (*Array Management Process*)

Este processo tem como finalidade controlar assíncronamente o funcionamento do arranjo de processadores, ou seja, controla a operação dos processadores no arranjo principalmente no que se refere à transferência de dados de cada processador ao processo AMP. Controla também a convergência global do sistema, verificando constantemente os indicadores de convergência dos processadores no arranjo. O algoritmo tem os passos mostrados na Fig. 2.4.

```
1.  Verificar o número de MPHs disponíveis no arranjo, armazena-lo e
    informar seu valor quando for requisitado.
2.  Receber o circuito (ou sub-circuito) a ser simulado do processo HIP.
3.  Mapear o circuito (ou sub-circuto) nos processadores do arranjo.
4.  Estabelecer as ligações do circuito (ou sub-circuito) na rede de
    interconexão.
5.  Transferir os parâmetros iniciais de simulação para cada MPH, assim
    como dados auxiliares referente a os processadores "vizinhos".
6.  Estabelecer o instante inicial de simulação e transferi-lo a todos
    os MPHs do arranjo.
7.  Dar a sinalização a todos os MPHs, para o inicio da simulação
    propriamente dita.
8.  Aguardar pela convergência global do arranjo de processadores.
9.  Buscar as soluções no arranjo.
10. Salvar as soluções do instante em curso na base de dados.
11. Estabelecer o seguinte instante de simulação.
12. Repetir os passos 7 a 11 até que todos os instantes tenham sido
    simulados.
13. Repetir os passos 2 a 12 caso o circuito tenha sido particionado,
    até que todo o circuito seja simulado.
14. Encerra o processo AMP, avisando aos processos HCP e HOP.
```

Fig. 2.4 Algoritmo do processo AMP.

A operação de mapeamento, mencionada no passo 3 (Fig. 2.4), consiste na distribuição dos elementos do circuito a simular no arranjo de processadores. Essa operação deve interagir fortemente com a operação de roteamento mencionada no passo 4. Critérios como o de manter interconexões entre elementos ligados a um mesmo nó no circuito, em processadores vizinhos no arranjo, devem ser seguidos.

O estabelecimento das ligações do circuito na rede de interconexão, mencionada no passo 4 (Fig. 2.4), consiste em configurar as chaves do sistema de interconexão programável de acordo com o roteamento realizado previamente. Esta operação interage fortemente com a operação de mapeamento referida no parágrafo anterior. Os algoritmos de roteamento têm

sido amplamente estudados, porém a aplicação de algum deles precisará do seu adequamento a nosso problema específico[Kho92, Lie92].

2.2.4 O Processo HOP (*Host Output Process*)

Este processo tem a função contrária ao do processo HIP. Ele fornece, num formato previamente determinado, os resultados da simulação que o usuário solicitou a partir da base de dados gerada no processo HIP e das soluções obtidas nos processadores do arranjo. Os passos seguidos no algoritmo são mostrados na Fig. 2.5.

```
1. Requisitar a base de dados do circuito gerada no processo HIP.
2. Ler da base de dados os resultados da simulação do processo AMP e
   formatá-los para a saída.
3. Quando os processos HIP e AMP sinalizarem o término de funcionamento,
   transferir os resultados finais, já formatados, para a saída
   especificada pelo usuário.
4. Terminar o processo HOP e avisar ao processo HCP de tal fato.
```

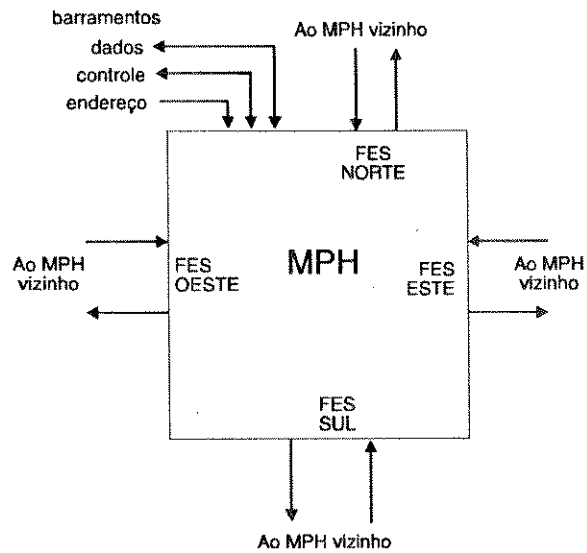
Fig. 2.5 Algoritmo do processo HOP.

2.3 Os Elementos de Processamento de Modelos (MPHs)

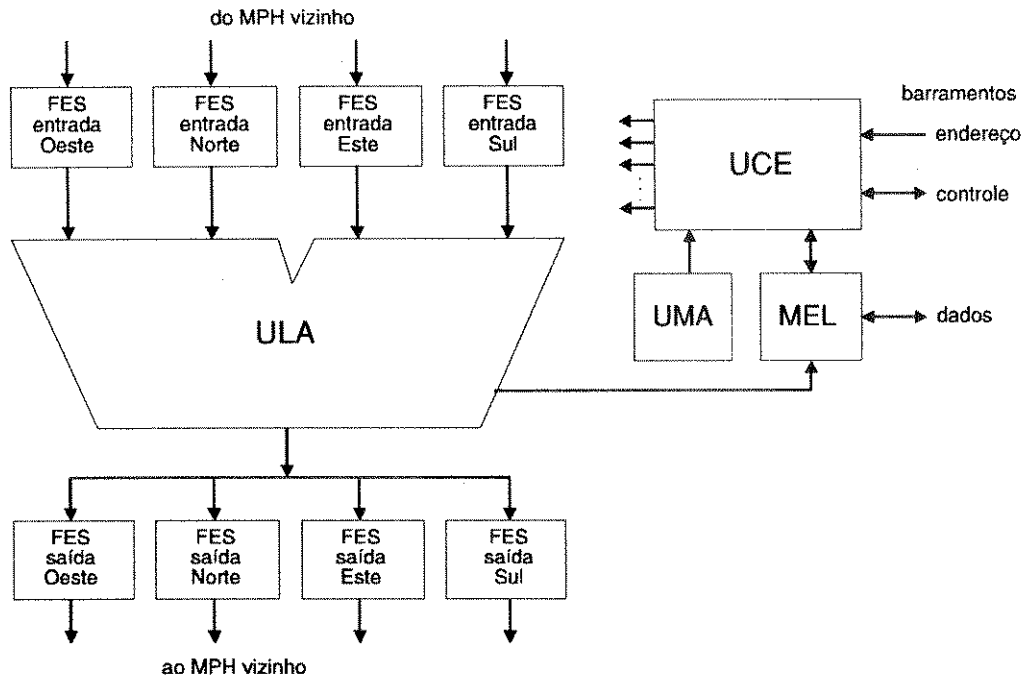
O arranjo de processadores está conformado por processadores dedicados à execução de modelos dos componentes do circuito a ser simulado. Mostra-se a seguir: a arquitetura destes elementos de processamento, o processo que é executado em cada um deles, os modelos dos componentes de um circuito e o algoritmo de processamento desses modelos.

2.3.1 Arquitetura dos MPHs

A arquitetura dos elementos de processamento de modelos está conformada por uma Unidade Lógica Aritmética (ULA), uma Memória de Escrita e Leitura (MEL), uma Unidade de Controle Elementar (UCE), uma Unidade de Modelos Armazenados (UMA) e Filas de Entrada e Saída (FES). A Fig. 2.6 mostra essa arquitetura.



a) diagrama de bloco



b) arquitetura

Fig. 2.6 Arquitetura básica de um Elemento de Processamento de Modelos (MPH).

A **ULA** está composta por blocos que realizam operações aritméticas e lógicas, como por exemplo somadores, multiplicadores, deslocadores, comparadores, etc. Todos esses blocos devem ser de alta velocidade e possivelmente as suas arquiteturas tenham que ser de *pipelining*[1]. Eles também, devem ser capazes de trabalhar com representações de números em ponto flutuante. Como será visto mais adiante, a ULA é o gargalo no processamento dos modelos dos componentes, consumindo uma parcela muito significativa do tempo de processamento. Portanto, pensou-se em uma ULA composta por vários blocos somadores, multiplicadores, comparadores, etc., de forma tal que se possa montar o modelo do componente, armazenado na UMA, em um *hardware* dedicado. Isto significa que a ULA é configurada apenas uma vez por circuito (ou sub-circuito) simulado.

A **MEL** por sua vez é uma memória de acesso aleatório. Nela, o processador gerenciador, durante a execução do processo AMP, armazena os parâmetros iniciais do componente a ser simulado e alguns dados dos processadores interligados, referentes à topologia e tipo de dispositivos que eles simulam. As MELs servem para armazenar os resultados da simulação e durante cada fase de simulação, ficam dedicadas aos respectivos MPHs. No fim de cada instante de simulação o processador gerenciador lê os resultados da mesma na MEL de cada MPH.

A **UCE** é a unidade encarregada de controlar o processamento nos MPHs e seu interfaceamento com o processador gerenciador. Ela executa o processo MPH. A UCE controla três fluxos de informação principais que acontecem nos MPHs: sinalização da convergência local para o processador gerenciador; a transferência de dados do processador gerenciador para o MPH e viceversa; e intercâmbio dos resultados parciais, por meio das FESs, com os processadores interligados. O controle interno das MPHs é síncrono e para tanto a UCE dispõe de um bloco de temporização.

A **UMA** é uma memória apenas de leitura, onde os modelos de dispositivos conhecidos pelo sistema ABACUS são armazenados. Estes modelos consistem de microprogramas e apenas aquele correspondente ao dispositivo que está sendo simulado é executado pela UCE. Os modelos mencionados não podem ser alterados durante o processo de simulação, porém, a modificação deles é possível em modo *offline*[2]. Isto permite a inclusão de modelos de novos dispositivos e também a troca dos modelos existentes por outros mais simples, ou mais complexos. Como já foi mencionado, os modelos dos dispositivos são montados em um *hardware* dedicado, portanto a UMA contém a configuração respectiva da ULA.

As **FES**s são os elementos encarregados de receber os resultados parciais, da iteração correspondente, vindos dos processadores ligados ao processador local e enviar os resultados deste último às FESs dos primeiros. Estes elementos, no modo de recepção, operam sob o princípio de fluxo de dados, ou seja, eles aguardam por eventos nas suas entradas. Quando algum deles detetar um evento, comunica à UCE do fato e esta por sua vez dá início à iteração seguinte.

---

[1] termo utilizado para a realização de operações em paralelo.

[2] termo utilizado para indicar a programação fora do modo de operação.

2.3.2 O Processo MPH

Os processos MPH (*Model Processing Hardware-element*), executados pelas UCEs, são os responsáveis pela simulação propriamente dita dos componentes do circuito. Estes processos trabalham com base nos dados fornecidos pelo processo AMP. O algoritmo dos processos MPH compreende os passos mostrados na Fig. 2.7.

```
1.  Receber os dados do processo AMP.
2.  Identificar o elemento a ser simulado.
3.  Configurar a ULA de acordo com o elemento.
4.  Identificar as FESs de entrada e saída ativas.
5.  Aguardar pelo sinal procedente do processo AMP para poder iniciar a
    simulação do instante em curso.
6.  Aguardar por novos dados nas FESs de entrada ativas.
7.  Calcular os resultados correspondentes.
8.  Verificar a condição de convergência local.
9.  Comunicar o estado de convergência ao processo AMP.
10. Salvar o resultado parcial, da iteração em curso, na MEL.
11. Transferir os resultados parciais para as FES de saída ativas.
12. Se existir convêrgencia global, aguardar pelo sinal do AMP para a
    transferência dos resultados do instante simulado. Logo após,
    retornar ao passo 5.
13. Voltar ao passo 6.
```

Fig. 2.7 Algoritmo do processo MPH

A convergência local e global, mencionadas no algoritmo da Fig. 2.7, referem-se à convergência no MPH local e no arranjo todo, respectivamente. Da mesma forma, as iterações referem-se àquelas processadas dentro de um mesmo instante de simulação.

Dependendo da topologia do circuito a ser simulado, ter-se-ão no arranjo, MPHs que não utilizem todas as FESs. Diz-se que uma FES está ativa, seja de entrada ou saída, quando ela está sendo utilizada para uma interconexão efetiva com outro MPH.

2.4 O Arranjo de Processadores

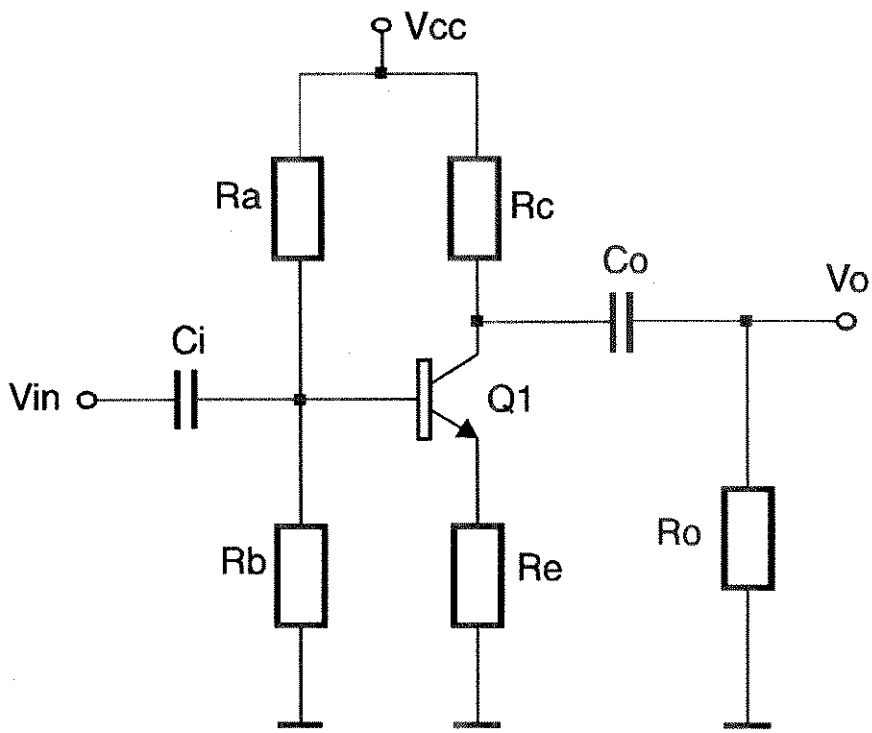
Os elementos de processamento de modelos (MPHs) descritos anteriormente, são os componentes básicos para a conformação do arranjo de processadores. A sua configuração é realizada efetivamente pelo processador gerenciador, na execução do processo AMP.

A metodologia ABACUS visa simular circuitos de alta escala de integração tais como VLSI e ULSI[Mar92]. Designam-se como circuitos VLSI aqueles que contém acima de 50.000 transistores e os circuitos ULSI aqueles que contém acima de 500.000 transistores[Hur92]. Circuitos comerciais tais como o processador 486 da INTEL, possuem aproximadamente um milhão e meio de transistores. O ideal na metodologia estudada, seria ter um número de processadores no arranjo, igual ao número de elementos no circuito. Inicialmente foi

considerado um arranjo com 32728 (2^{15}) processadores de modelos, os quais estariam distribuídos em 32 camadas e cada camada possuiria 32x32 processadores[Mar92]. Com isto os circuitos poderiam ser particionados em módulos de 30.000 elementos, tendo-se um aproveitamento do 90% do arranjo.

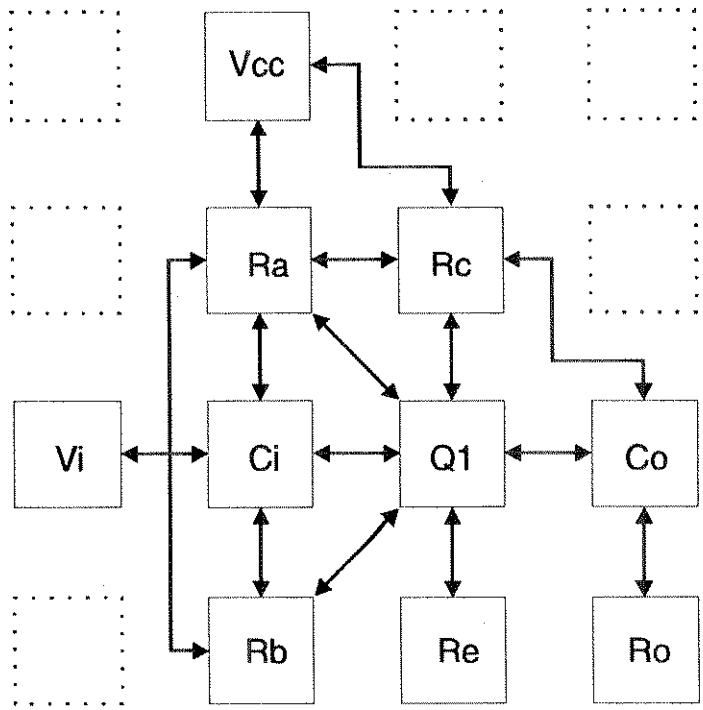
Para ter uma melhor visão acerca do mapeamento de um circuito no arranjo de processadores, considere-se como exemplo o circuito da Fig. 2.8a e sua distribuição respectiva no arranjo de processadores na Fig. 2.8b. Note-se no exemplo que o circuito foi mapeado em um arranjo de 4x4 processadores. Nele são apenas mostradas as ligações que suportam o fluxo de dados entre os elementos do circuito, o qual é bidirecional. Não é ilustrada a rede de interconexão, nem os barramentos que existem entre o processador gerenciador e o arranjo, para uma melhor compreensão.

Observe-se algumas particularidades, como por exemplo que o processador que simula o transistor Q1 tem 6 ligações, ultrapassando o número previsto de FESs como foi descrito anteriormente. Esse problema e uma possível solução serão discutidos mais adiante. Note-se também que as ligações ao terra não foram consideradas no mapeamento do circuito (Fig. 2.8a) no arranjo (Fig. 2.8b).



a) Circuito exemplo

Fig. 2.8 Exemplo de mapeamento de um circuito num arranjo de 4x4 processadores.
(continua na Pág. 13)



b) Mapeamento no arranjo

Fig. 2.8(cont.) Exemplo de mapeamento de um circuito num arranjo de 4x4 processadores.

2.4.1 Modelos de Elementos de Circuitos

Como já foi mencionado, a UMA armazena os modelos dos elementos de circuito que um MPH pode simular. Inicialmente pensou-se na simulação dos diversos elementos que um circuito pode ter, em função de alguns elementos básicos como resistores, capacitores, diodos e fontes dependentes e independentes. Isto permitiria a inclusão gradual de modelos mais complexos que contemplem por exemplo, o efeito das capacitâncias parasitas em um transistor MOS ou outro tipo de efeitos. Não é nosso objetivo aprofundar no tema, o qual pode ser questão de outros estudos. Para uma primeira fase, podemos considerar os modelos mostrados a seguir:

Resistor:

$$V=R.I$$

Capacitor:

$$I=Io.e^{\frac{t-\Delta t}{\tau}}$$

$$V=Vf(1-e^{-\frac{t-\Delta t}{\tau}})+Vcarga.e^{-\frac{t-\Delta t}{\tau}} \qquad \tau=\frac{Vf.C}{Io}$$

Diodo:

$V<0 \Rightarrow I=-Isat$

$V\geq 0 \Rightarrow I=\Sigma Ientrada$

$$V=Vt.Ln(\frac{I}{Isat}+1)$$

Fontes de tensão e corrente senoidais:

$$I=Imax.Cos(2\pi ft)$$

$$V=Vmax.Cos(2\pi ft)$$

Fontes de tensão e corrente quadradas:

$Round(tf)\leq tf \Rightarrow V=Vmax; \quad I=Imax$

$Round(tf)>tf \Rightarrow V=-Vmax; \quad I=-Imax$

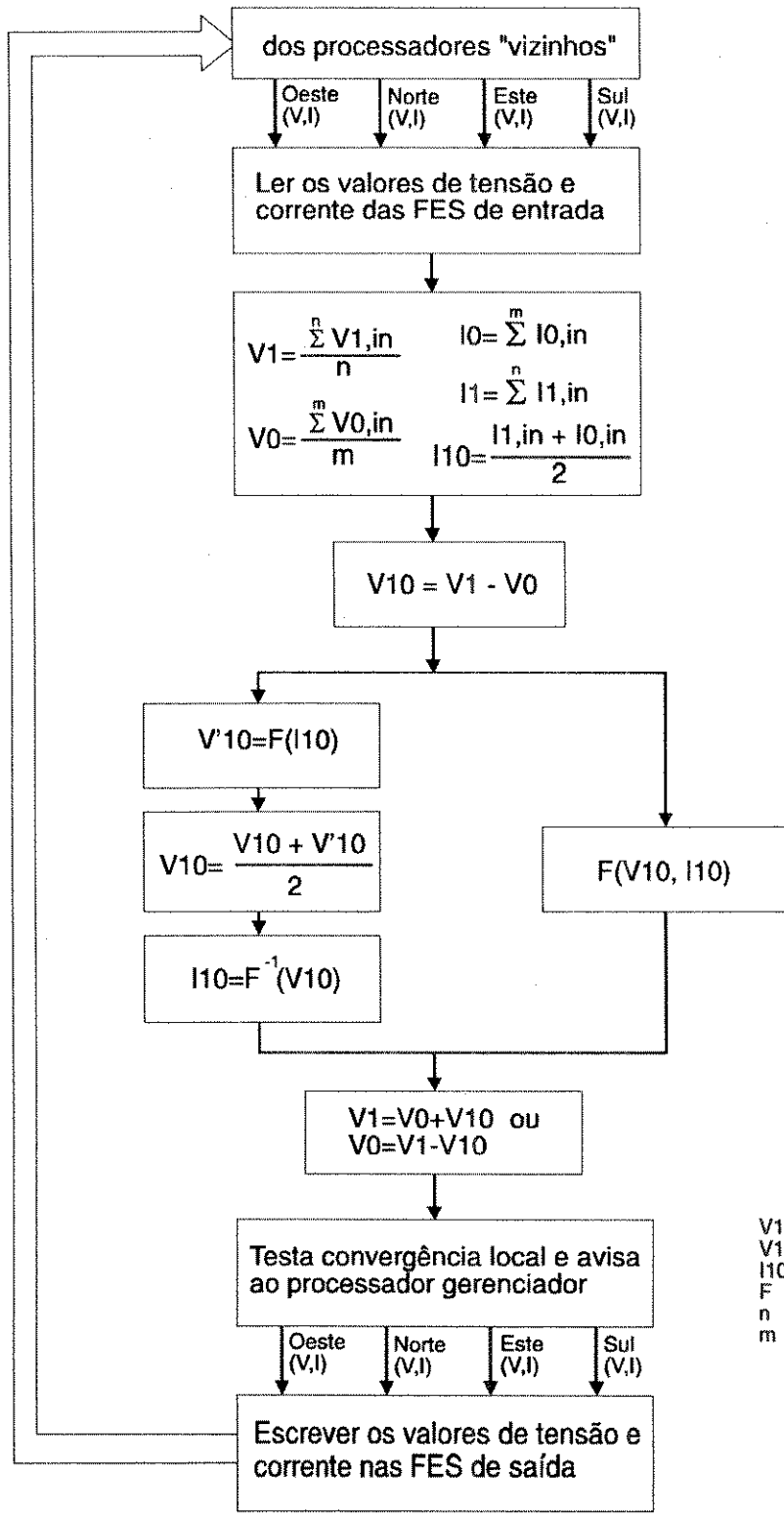
Fontes controladas:

$I=R.Vcontrole; \quad I=Ki.Vcontrole$

$V=g.Icontrole; \quad V=Kv.Vcontrole;$

2.4.2 Algoritmo de Processamento dos Modelos

O processo MPH executado pela UCE, é o responsável pela simulação propriamente dita, como já foi citado anteriormente. A parte mais importante desse processo compreende os passos 6 a 13 de seu algoritmo. Na Fig. 2.9 ilustra-se em um diagrama de fluxo o algoritmo de processamento dos modelos de elementos de um circuito, que expressa com maior detalhe tais passos.



V1,in tensões de entrada polos 1 e 0
V10 tensão entre os terminais do elemento
I10 corrente que circula pelo elemento
F modelo do elemento
n número de elementos ligados ao polo 1
m número de elementos ligados ao polo 0

Fig. 2.9 Diagrama de fluxo do algoritmo de processamento dos modelos.

Cada processador do arranjo recebe os dados, correspondentes à tensão e corrente dos polos (terminais do dispositivo) 0 e 1 do elemento de circuito, vindos dos processadores vizinhos[1] pelas FESs oeste, norte, este e sul. Com estes valores, o MPH calcula a média aritmética das tensões V_0 e V_1 e a diferença V_{10}, assim como a media aritmética da corrente I_{10} que atravessa o elemento calculado a partir da soma algébrica das correntes que chegam aos polos 0 e 1. Neste ponto tem-se duas alternativas: a primeira corresponde à utilização do algoritmo da biseção[Dem78], que consiste no cálculo de uma segunda tensão V'_{10} aplicando o modelo F à corrente I_{10}; cálculo da média aritmética das tensões V_{10} e V'_{10}; cálculo de uma nova corrente I_{10}, aplicando-se o modelo F à média de V_{10} recém mencionada. A segunda alternativa consiste na aplicação direta do modelo F, quer para o cálculo da tensão V_{10}, quer para o cálculo da corrente I_{10}. Após esta bifurcação, calcula-se novamente os valores das tensões V_1 e V_0 em função de V_{10} calculado anteriormente. Com os valores de V_0, V_1 e I_{10} da iteração anterior e os calculados recentemente, calcula-se o erro e com base nele determina-se a convergência local. Finalmente, os valores calculados de V_0, V_1 e I_{10} são colocados nas FES de saída e delas estes valores são transferidos aos processadores interligados.

2.4.3 O MPH de Extensão

A utilização da estrutura de quatro FES, tanto para a entrada quanto para a saída, para cada processador no arranjo, foi escolhida pela simetria quadrangular que ela apresenta, facilitando a organização do arranjo.

No circuito exemplo, apresentado na Fig. 2.8, observa-se o problema da existência de mais de quatro ligações a um mesmo processador, como no caso do transistor Q1. Portanto, o mapeamento mostrado na Fig. 2.8b, não seria possível na estrutura de 4 FES mencionada anteriormente. Uma solução possível, é a de utilização de um processador adicional ao utilizado para simular o transistor Q1. Esse processador será chamado de MPH extensor. O novo mapeamento mostra-se na Fig. 2.10, onde novamente foi utilizado um arranjo de 4x4 processadores, mas com uma distribuição diferente. Note-se que o MPH extensor de Q1 (EXT Q1) liga ao processador de Q1 somente os processadores que simulam elementos que estão ligados ao mesmo nó (Ra, Rb e Ci), ou seja, ao nó da base de Q1. Isto facilita o processamento, como será visto no capítulo seguinte.

No exemplo anterior pode-se observar que um MPH extensor acrescenta em 2 o número de FES de entrada e saída. Se fosse ligado um MPH extensor a cada FES, teríamos em total um número de 12 FES disponíveis para cada processador. Esta estrutura poderia ser útil para aqueles processadores que simulam fontes de alimentação, as quais comumente estão ligadas a um grande número de elementos. Seguindo com a lógica anterior, poderia criar-se um número grande FES por extensão para cada MPH. Não obstante, isso tem um custo no tempo de processamento, provocando atrasos, além da diminuição do número de processadores disponíveis no arranjo.

[1] Refere-se aos processadores interligados por meio da rede de interconexão.

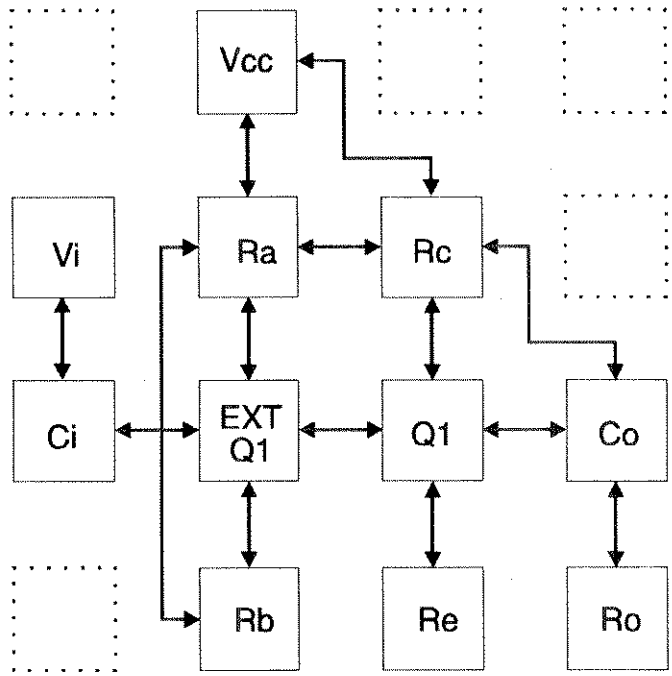


Fig. 2.10 Mapeamento do circuito da Fig. 2.8a, com a utilização de um MPH extensor para Q1.

O funcionamento de um MPH de extensão pode ser explicado da seguinte maneira: a UCE lê os dados de tensão e corrente, vindos dos processadores "conectados por extensão" (V_{in}. I_{in}), assim como os dados do "processador extendido" (V_R, I_R) nas FES de entrada. Com os valores de tensão e corrente dos processadores conetados por extensão, calcula-se a média aritmética da tensão (V_p) e a soma algébrica das correntes (I_p). Como já foi mencionado, V_p e I_p representam exclusivamente os dados de um determinado polo (0 ou 1) do elemento. Logo, a UCE escreve a tensão e corrente (I_R, V_R) para as FESs de saída ligadas aos processadores conetados por extensão. Escreve também a tensão V_p e corrente I_p para as FES da saída conetada ao processador extendido. O diagrama de fluxo deste algoritmo mostra-se na Fig. 2.11.

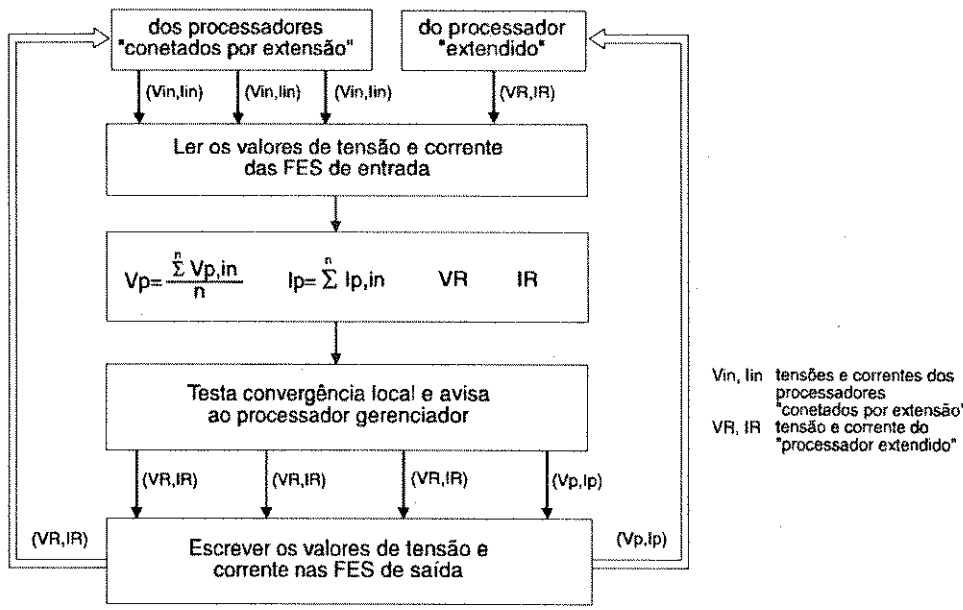


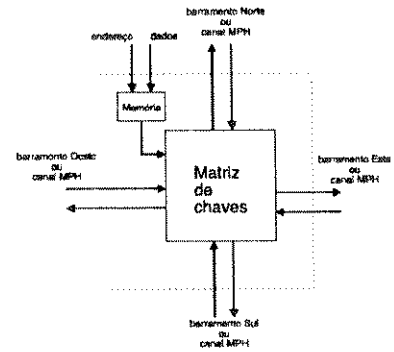
Fig. 2.11 Diagrama de fluxo do algoritmo de processamento em um MPH de extensão.

2.5 O Elemento de Interconexão

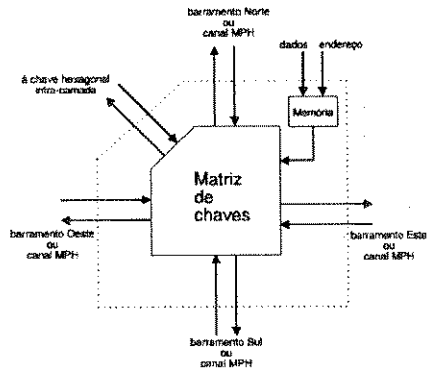
A rede de interconexão, mostrada na Fig. 2.1, formada basicamente por um conjunto de barramentos e outro de chaves, tem por finalidade a interligação de um elemento de processamento de modelos (MPH) com outros, de forma que as interligações existentes no circuito a simular possam ser refletidas no arranjo de processadores. A configuração desta rede, por parte do processador gerenciador, é feita durante a execução do processo AMP e ela é modificada apenas quando for simular outro circuito (ou subcircuito).

Especificamente a configuração da rede de interconexão consiste na programação das chaves. Para este fim utilizam-se os barramentos de dados e endereços (vide Fig. 2.1), sendo que a identificação de cada chave é feita por meio do barramento de endereços e a sua programação propriamente dita por meio do barramento de dados. As chaves têm um elemento de memória para armazenar dita configuração.

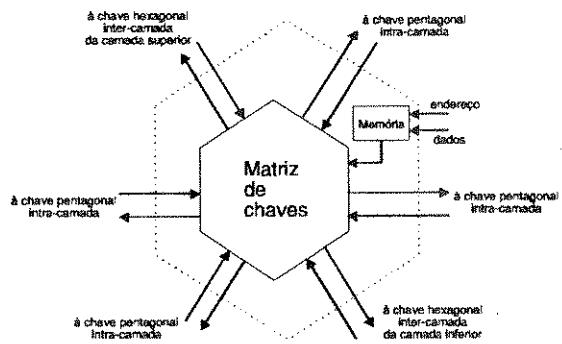
Dependendo da localização, no arranjo de processadores, as chaves podem ser dos seguintes tipos: chaves quadradas intra-camada, que são utilizadas para providenciar pontos de acesso aos MPHs para os barramentos e outros MPHs da mesma camada (vide Fig. 2.12a); chaves pentagonais intra-camada, que dão acesso às chaves quadradas intra-camada para as chaves hexagonais inter-camada e outras chaves intra-camada (vide Fig. 2.12b); chaves hexagonais inter-camada, que principalmente cumprem a função de dar pontos de acesso aos MPHs de outras camadas (vide Fig. 2.12c).



a) chave quadrada intra-camada



b) chave pentagonal intra-camada



c) chave hexagonal inter-camada

Fig. 2.12 Arquitetura das chaves utilizadas na rede de interconexão.

A interconexão dos processadores no arranjo, por meio da rede de interconexão, é tal que para cada par de linhas de processadores existem 2 barramentos em paralelo. Da mesma forma acontece para cada par de colunas de processadores. As chaves quadradas intra-camada estão localizadas de tal forma que as FES dos MPHs tenham acesso aos barramentos vertical e horizontal. Nos pontos de interseção destes barramentos têm-se 4 chaves pentagonais intra-camada. O quinto canal destas chaves está ligado a uma chave

hexagonal inter-camada, tendo-se assim acesso às outras camadas. Para melhor ilustração, na Fig. 2.13, mostra-se uma região do arranjo.

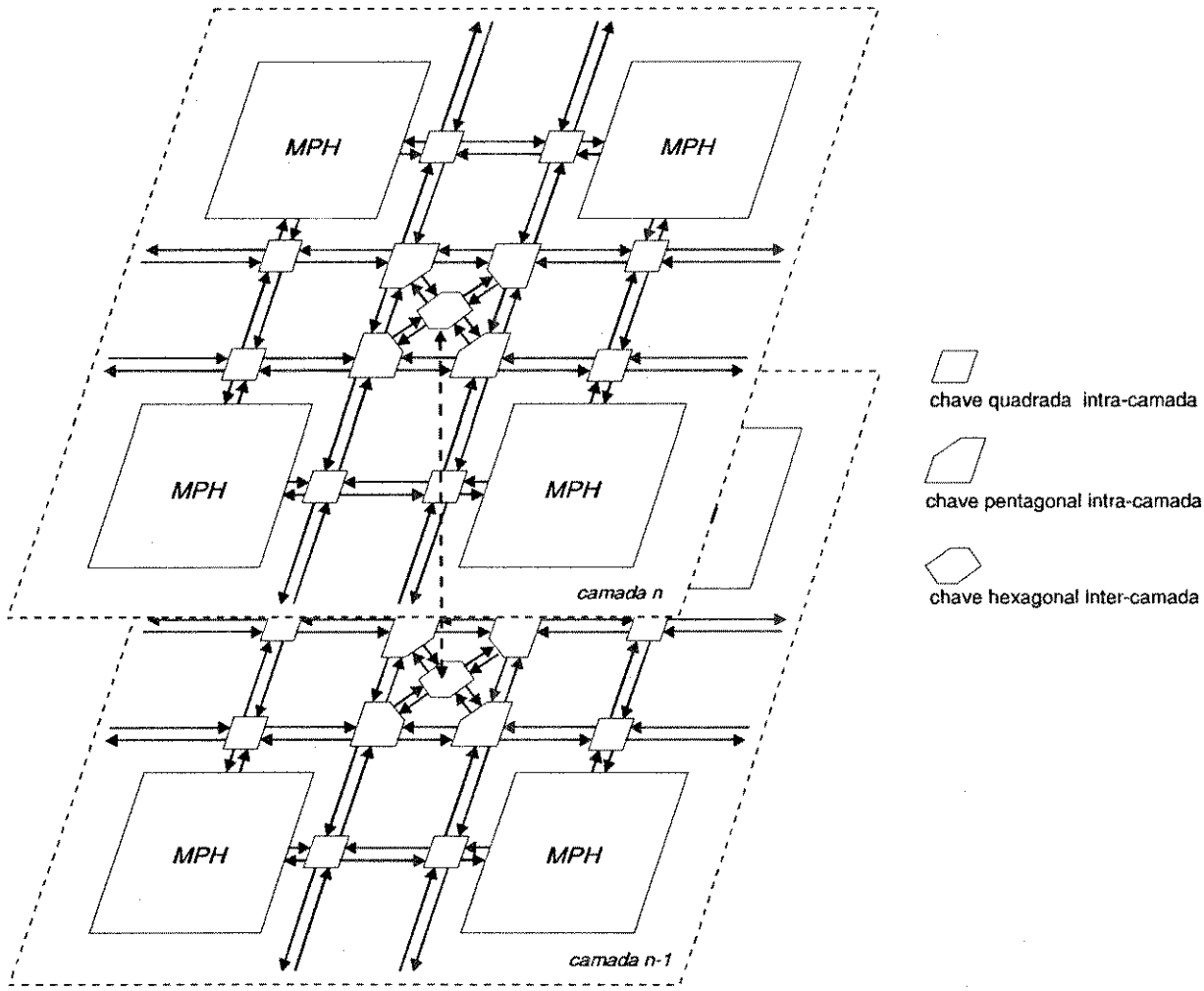


Fig. 2.13 Interconexão entre os processadores do arranjo.

2.6 Especificação da Arquitetura ABACUS.

As características gerais da arquitetura ABACUS, base para o modelamento em VHDL, são apresentadas a seguir. Algumas delas só poderão ser dimensionadas uma vez realizadas as simulações, no entanto, outras poderão ser já estabelecidas com base nas descrições dadas anteriormente.

As tarefas do processador gerenciador podem ser elaboradas num computador hospedeiro de grande capacidade de processamento, de tal forma que os processos HCP, HIP, HOP e AMP possam ser executados em paralelo. São necessárias capacidades de memória suficientemente grandes para poder manusear bases de dados de circuitos de grande escala de integração. É preciso também ter barramentos de alta velocidade e possibilidades de interligação aos barramentos de endereços, de controle e de dados.

Como foi mencionado anteriormente o arranjo consiste de 2^{15} processadores. A sua interconexão, proposta em [Mar92], assim como a distribuião das chaves da rede de interconexão é como se observa na Fig. 2.14.

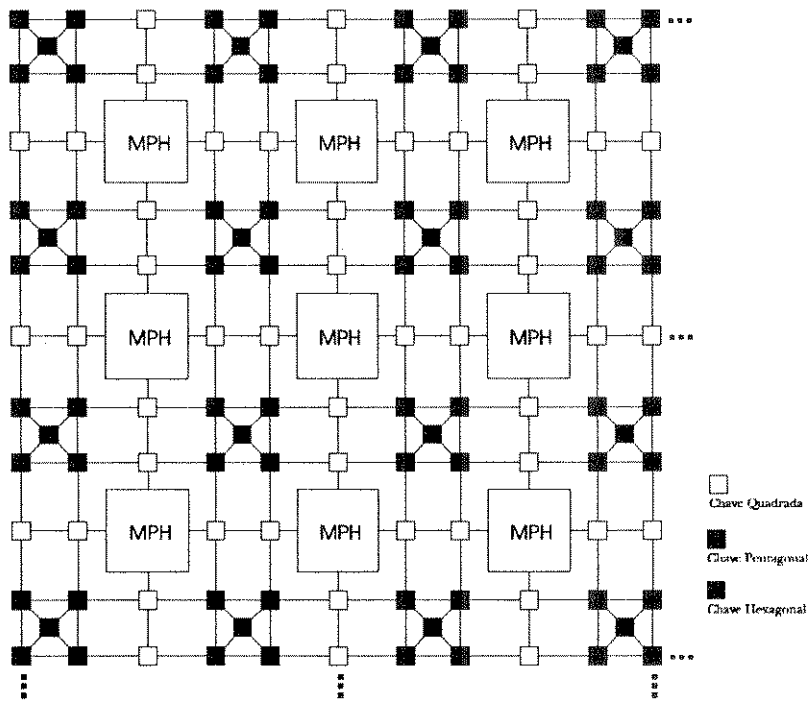


Fig. 2.14 Segmento do arranjo mostrando a distribuição das chaves e das MPHs

A estrutura dos barramentos para a comunicação entre o hospedeiro e o arranjo de processadores consiste de: um barramento de 15 bits para o endereçamento dos MPHs mais 6 bits para o endereçamento das posições da MEL em cada MPH, considerando que estas memórias armazenam 64 palavras; um barramento de dados, considerando o tamanho de cada palavra na MEL é de 32 bits; 3 linhas de controle para a leitura e escrita nas respectivas memórias em cada MPH, realizada por meio dos sinais READ e WRITE respectivamente, além do sinal READY que indica a disponibilidade da MEL; 2 linhas de controle, uma para o sinal RESET, que inicializa a simulação e outra para o sinal START, que começa um novo instante da simulação; uma linha de controle para que a convergência global seja comunicada ao hospedeiro por meio do sinal CONV. Totalizando, obtemos un número de 59 bits. No diagrama da Fig. 2.15 ilustra-se isto com mais detalhe.

A configuração das chaves da rede de interconexão é realizada por meio de um barramento de 24 bits, o qual é formado de um barramento de endereços de 19 bits, um barramento de dados de 4 bits e um sinal de controle. A dimensão do barramento de endereços, correspondente ao número total de chaves da rede de interconexão, pode ser calculado assim:

$$[(PE \times 3 + 2]^2 - PE^2 + (PE+1]^2] \times L = \text{Número total de chaves}$$

onde PE é o número de MPHs de uma linha e L o número de camadas. Como $PE=2^5=32$ e

$L=2^5$, então o número total de chaves é de 309.408. A configuração efetiva das chaves é realizada por meio do sinal CONFIG e do barramento de dados. A dimensão do barramento de dados foi calculada considerando todas as ligações possíveis diferentes que se dão entre os canais da chave hexagonal (a que tem maior número de canais), obtendo-se o número de 15, codificáveis com 4 bits.

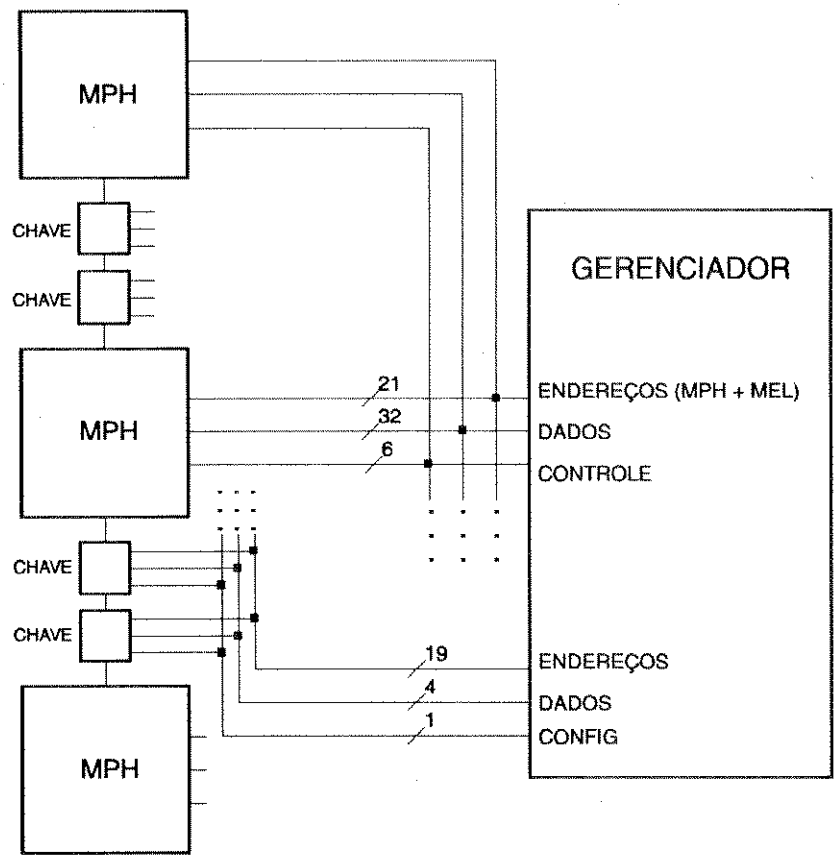


Fig. 2.15 Esquema de interconexão do processador gerenciador aos MPHs e às chaves.

O dimensionamento dos barramentos para a comunicação entre os MPHs será analisado com mais detalhe no capítulo seguinte com base nos resultados da simulação.

Finalmente, o tempo de processamento dos modelos dos dispositivos são considerados de acordo com os tempos dados em [Mar92], valores que foram calculados com base no número de ciclos utilizados para as operações em ponto flututante do Transputer T800-20 da INMOS[INM88] (vide tabelas 2.1 e 2.2).

Operação	Nro. de ciclos
Atribuição	3
Soma	3
Substração	6
Comparação	6
Divisão	31
Multiplicação	18
Exponenciação	796
Logaritmo	690

Tabela 2.1 Tempo de execução das operações em ponto flutuante do Transputer T800-20. (1 ciclo = 50 nS).

Dispositivo	Nro. de ciclos
Resistor	919
Capacitor	1020
Diodo	2555
Fonte de tensão independente	707
Fonte de tensão quadrada	879
MPH - extensor	49

1 ciclo = 50 nS.

Tabela 2.2 Tempos de execução dos modelos dos dispositivos calculados em base dos tempos da tabela 2.1.

Capítulo 3

Modelamento e Simulação em VHDL

O crescimento da complexidade dos sistemas digitais, faz com que o projeto em nível lógico se torne cada vez mais difícil. Para resolver esse problema, os projetistas de *hardware* têm migrado, da tradicional metodologia de projeto *bottom-up* para métodos mais hierárquicos. As Linguagens de Descrição de *Hardware*, tal como a VHDL[1] tornaram possível especificar, simular e projetar sistemas digitais complexos a partir de dados gerais simples[Nag92].

Vamos no presente capítulo estudar brevemente os conceitos sobre representação e níveis de abstração que um projeto pode ter, a aplicação da metodologia *top-down* e o modelamento na linguagem de descrição de *hardware* VHDL, assim como algumas características da própria linguagem. A simulação de sistemas digitais em VHDL, como meio de verificação das descrições através dos recursos de verificação do comportamento e do *back-annotation*, também sera visto.

Finalmente, descreveremos uma parte do ambiente de projeto *Falcon Framework* da empresa *Mentor Graphics*, que foi utilizado na realização de nosso trabalho e que se encontra disponível na FEE/UNICAMP. Neste sistema, enfatizaremos as ferramentas de compilação e de simulação de descrições em VHDL.

3.1 Conceitos sobre Representação de um Projeto.

A representação do projeto de um sistema digital, freqüentemente é classificada por seu domínio e em cada dominio por seu nível de abstração[Cam92, Nag92, Lip89].

Na primeira classificação, usualmente tem-se três domínios: o comportamental (funcional), o estrutural (topológico) e o físico (geométrico). Tipicamente a representação das descrições de um projeto, no domínio comportamental realizam-se em um programa de uma linguagem de descrição de *hardware*, no domínio estrutural em *netlists* e no domínio físico em *layouts*.

---

[1] VHSIC *Hardware Description Language* (VHSIC - *Very High Speed Integrated Circuits*)

Na segunda classificação tem-se vários níveis de abstração em cada domínio. Como exemplo, o domínio estrutural pode ser dividido da seguinte forma: nível de arquitetura, nível RT (*Register-Transfer Level*), nível lógico e nível de dispositivo. Cada nível do domínio estrutural pode ser caracterizado pelos elementos utilizados na representação, assim como no domínio físico pelas células geométricas. Desta forma, no domínio estrutural, o nível de arquitetura está conformado por processadores, memórias e barramentos; o nível RT, por unidades funcionais (ULA, somadores, etc.) e registradores; o nível lógico, por portas e flip-flops; e, o nível de dispositivo, por transistores, capacitores e resistores.

Os domínios e níveis de abstração mencionados podem ser representados no Diagrama Y[Cam92], como é mostrado na Fig. 3.1.

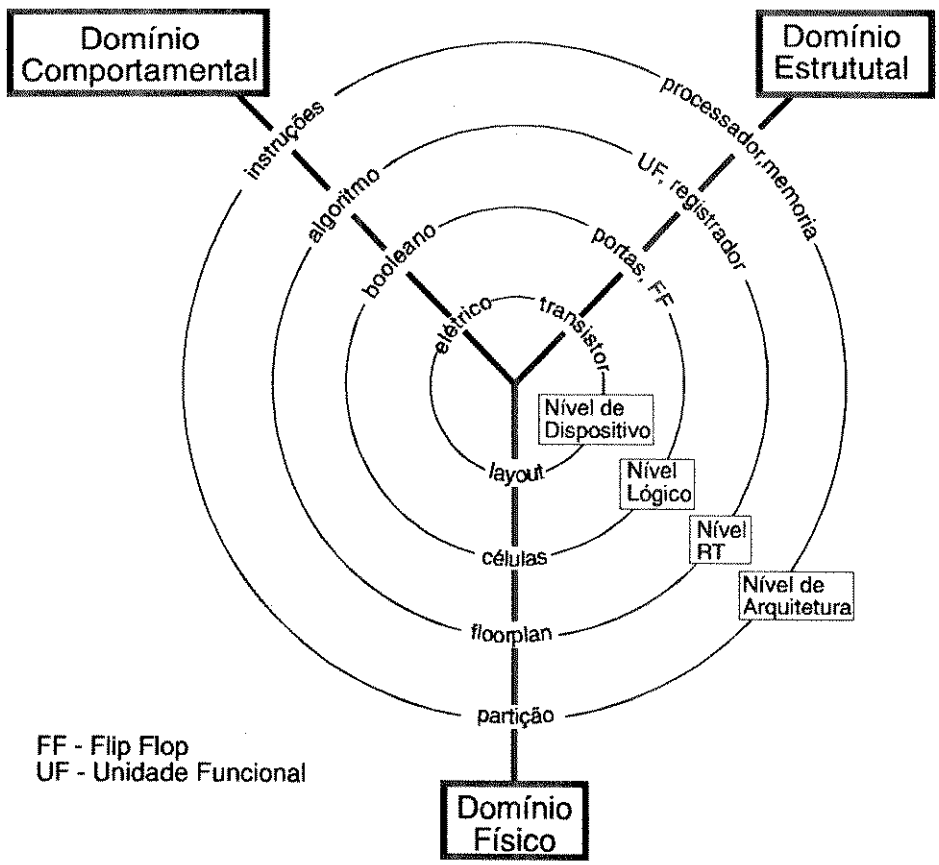


Fig. 3.1 Representação dos domínios e níveis de abstração de um projeto no diagrama Y.

As descrições arquiteturais estão no nível de abstração mais elevado. Estas descrições mostram o que os módulos fazem e não como o fazem, portanto, detalhes sobre quantos ciclos de relógio um modelo precisa para realizar sua tarefa não são dados. Por exemplo, na descrição arquitetural de um microprocessador não se tem idéia do funcionamento da ULA, no máximo ter-se-ão especificações que poderiam ser atribuídas a ela, em um ou vários ciclos de relógio. Este nível de descrição é independente da tecnologia.

As descrições RTL, estão no nível seguinte de abstração, logo abaixo do nível arquitetural. Elas definem o sistema em termos de unidades funcionales, registradores, multiplexadores. Diferente do arquitetural, neste nível tem-se noção de uma arquitetura e do esquema de temporização. Similarmente ao nível arquitetural, este nível é também independente da tecnologia.

No último nível de abstração, estão as descrições do nível lógico. Neste nível de abstração a implementação de um projeto é descrito por meio de uma Rede Booleana. Estas descrições, além de manterem a arquitetura derivada do nível RT, também mostram as implementações das funções booleanas. Apesar de representar tais descrições numa tecnologia genérica, normalmente elas dependem de alguma tecnologia em particular.

3.2 A Metodologia de Projeto *Top-Down* e a Síntese

Na metodologia de projeto *top-down* o projetista começa modelando o sistema no nível mais elevado de abstração, ou seja, no nível arquitetural[Per91]. A partir deste, por meio da síntese, vai descendo nos níveis de abstração até chegar no nível de abstração mais baixo, o nível de dispositivo. Costumeiramente, este processo começa com representações de projeto no domínio comportamental, passando pelo domínio estrutural até chegar no domínio físico. O diagrama da Fig. 3.2 nos mostra o fluxo de projeto nessa metodologia.

A síntese é uma tarefa de refinamento no processo de projeto de um sistema, que vai de um nível de abstração mais alto para outro mais baixo[Car92]. Esta transformação é o processo de mapeamento de uma representação abstrata de projeto em outra.

Existem atualmente ferramentas de síntese automática para diferentes níveis de abstração. Particularmente, tem-se o HLS (*High-Level Synthesis*) ou síntese arquitetural, que faz a transformação de representações do nível de arquitetura em representações do nível RT[Cam92]. Da mesma forma, a síntese RTL mapeia representações do nível RT no nível lógico ou de portas[Car92]. Na Fig. 3.2 ilustram-se estes conceitos. Potencialmente, a síntese automática pode resultar em representações corretas por construção, embora não otimizadas, razão pela qual vem crescendo a sua utilização na indústria, apesar de suas limitações.

A HLS e a síntese RTL compreendem várias operações, assim temos: particionamento, *pipelining*, *scheduling*, alocação de registradores, alocação de recursos, interferência entre registradores, síntese da máquina de estados, otimização da lógica multinível e otimização da lógica de dois níveis[Car92].

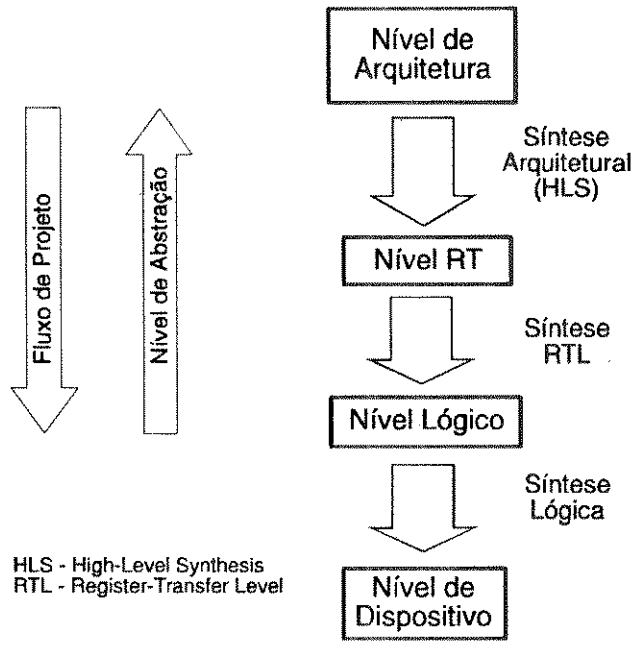


Fig. 3.2 Fluxo de projeto na metodologia Top-Down.

3.3 As Linguagens de Descrição de *Hardware*

Com o aparecimento das tecnologias de alta escala de integração como VLSI e ULSI, a automatizacão do processo de projeto dos circuitos integrados foi forçando o surgimento das linguagens que descrevessem estes sistemas digitais[Mis92]. Estas linguagens chamadas de descrição de *hardware* (HDL - *Hardware Description Language*) têm as seguintes finalidades:

1. Descrever os sistemas digitais de forma precisa e concisa.
2. Documentar convenientemente o sistema.
3. Ser entrada de descrição de sistemas, para verificação e implementação de projetos.
4. Servir como interface entre o projetista e o usuário.
5. Permitir a incorporação de mudanças de projeto e as correspondentes mudanças de documentação.

Na escolha de uma HDL deve verificar-se se ela é orientada tanto à simulação, quanto à síntese.

3.4 A Linguagem VHDL

A VHDL (VHSIC *Hardware Description Language*) foi desenvolvida sob o auspício do Departamento de Defesa dos Estados Unidos, no Programa *Very High Speed Integrated Circuits* (VHSIC) e tornou-se um padrão IEEE para descrição, documentação e simulação de sistemas digitais[IEE87]. Embora tenha sido projetada inicialmente como uma ferramenta de

modelamento e simulação, ela pode ser utilizada como ferramenta de síntese.

Na sua evolução podemos distinguir três fases: definição, onde foram delineadas as características fundamentais e suas limitações (1980 a 1986); desenvolvimento, onde foi feito o refinamento e a ampliação, para facilitar o seu uso comercial amplo (1986 até o presente); e, exploração, fase final de amadurecimento onde evoluiu ao ponto de tornar-se uma ferramenta muito utilizada (1990 em diante)[Dew92].

Em VHDL um sistema digital pode ser descrito por meio de três modos de representação: descrições comportamentais, *data-flow* e estruturais[Nag92, Dar90]. As descrições comportamentais ou funcionais, são as mais apropiadas para uma simulação rápida de sistemas digitais complexos; verificação e simulação funcional de idéias de projeto; modelamento de componentes padrão; e, documentação. As transformações entre as entradas e saídas são especificadas por meio de processos, os quais contêm instruções seqüenciais de forma similar às linguagens de programação de alto nível. As descrições neste modo podem ser escritas de forma que pessoas que não sejam da área possam entendê-los. Podem ser chamadas também de operacionais.

Diferente das descrições comportamentais, as descrições *data-flow* incluem a arquitetura do sistema digital. Neles são representados os fluxos de controle e de dados. Os relacionamentos entre as entradas e saídas são especificadas por meio de instruções e atribuições concorrentes a sinais. As simulações neste modo tornam-se mais lentas por estarem no nível de transferência de dados entre registradores e barramentos, porém, a função do *hardware* torna-se evidente.

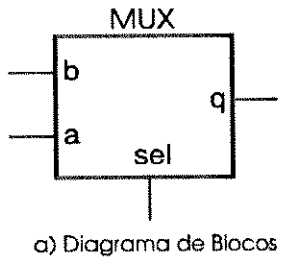
As descrições estruturais são as de maior detalhamento. Elas são especificadas por meio das interconexões dos componentes. As funções correspondentes das arquiteturas desses componentes não são visíveis neste tipo de descrições, não obstante, o *hardware* envolvido é claro e pode ser realizado até por um roteador simples.

Em simulações onde é necessário testar a funcionalidade, é conveniente usar descrições do tipo comportamental. Da mesma forma, em simulações onde é necessário verificar as resposta exatas do *timing*[1], é conveniente utilizar descrições do tipo estrutural[Nav91].

3.4.1 As Entidades de Projeto

Em VHDL um sistema digital é representado como sendo uma entidade de projeto. As entidades de projeto consistem de uma especificação de interface e uma especificação de arquitetura. A especificação de interface, que começa com a palavra `ENTITY`, além de dar um nome à entidade contém as portas de entrada e saída do componente[IEE87]. Considere-se como exemplo a descrição de interface para um bloco multiplexador 2 a 1, mostrado na Fig. 3.3.

[1] Termo utilizado para indicar a temporização de um sistema digital.



a) Diagrama de Blocos

```
ENTITY mux IS
   PORT ( a,b    : IN BIT;
          sel    : IN BIT;
          q      : OUT BIT);
END mux;
```

b) Interface em VHDL.

Fig. 3.3 Especificação da interface para um bloco multiplexor 2 a 1.

No exemplo, a descrição dos sinais de interface incluem o modo (`IN` ou `OUT`) e o tipo (`BIT`, no exemplo). `a` e `b` são as entradas, `sel` a entrada de seleção e `q` a saída.

3.4.2 As Arquiteturas e Os Níveis de Modelamento

A especificação da arquitetura, que começa com a palavra `ARCHITECTURE`, contém a descrição do funcionamento da entidade de projeto[IEE87].

Como será visto, para uma especificação de interface, podem existir várias especificações de arquitetura, cada uma com um nome diferente. Por exemplo, um componente pode ter uma especificação de arquitetura comportamental e outra de tipo *data-flow* ou estrutural (vide Fig. 3.4).

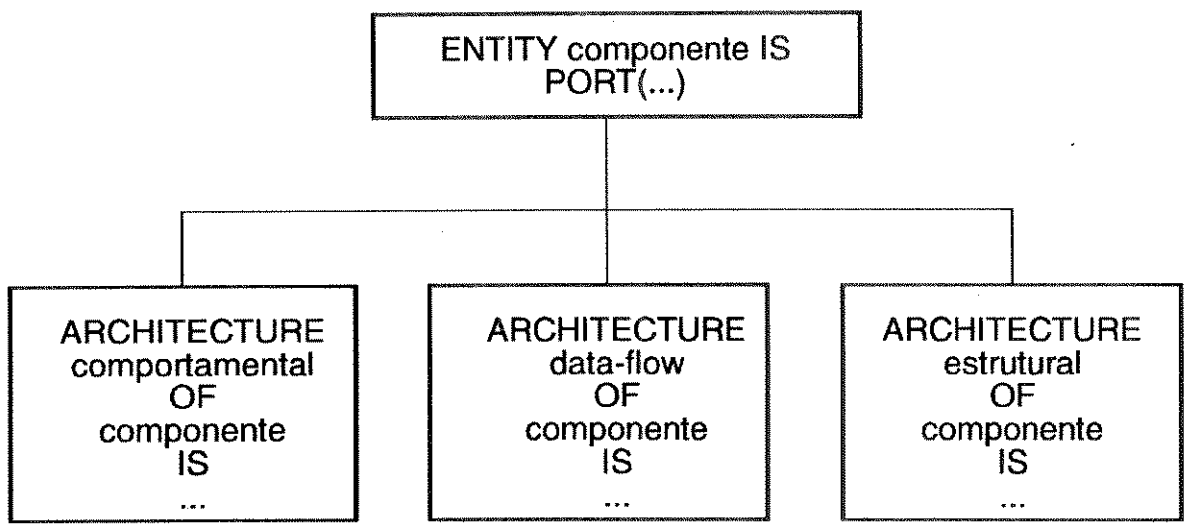


Fig. 3.4 Três especificações de arquitetura para uma especificação de interface.

No início do processo de projeto, o projetista usualmente tem uma idéia (algoritmo) sobre o funcionamento (comportamento) do sistema, coincidindo isto com o nível de maior abstração da metodologia de projeto *top-down*. Portanto, a necessidade inicial é de uma verificação da exatidão do algoritmo sem especificar uma implementação detalhada. Na Fig. 3.5, mostra-se a arquitetura correspondente à interface mostrada na Fig. 3.3, com uma descrição comportamental.

```
ARCHITECTURE comportamental OF mux IS

BEGIN

        mux2_1 : PROCESS (sel)
                 VAR temp : BIT;
        BEGIN

                 IF sel='0'THEN
                        temp := a;
                 ELSE
                        temp := b;
                 END IF;

                 q <= temp;

        END mux2_1;

END comportamental;
```

Fig. 3.5 Especificação da arquitetura correspondente à especificação de interface da Fig. 3.2b.

Note, na arquitetura da Fig. 3.5 que o condicional `IF` permite decidir qual das entradas (`a` ou `b`) deve ser transferida para a saída (`q`), segundo a entrada de seleção (`sel`). A ocorrência de algum evento em `sel` é supervisionada pelo processo `mux2_1`. A variável `temp` é usada para armazenamento temporário. Note-se também que a descrição comportamental mostrada, descreve perfeitamente o algoritmo, porém, a sua correspondência com o *hardware* real é fraca.

Continuando com o processo de projeto, o estágio seguinte da metodologia *top-down*, é a descrição *data-flow*. O exemplo a seguir trata da determinação da equação booleana a partir dos mapas de Karnaugh, que relacionam a saída em função das entradas. Note-se que o processo de síntese pode ser um processo intuitivo por se tratar de um sistema simples. A equação é:

```
q = a.sel + b.sel
```

Neste ponto o projetista pode substituir a arquitetura comportamental pela arquitetura *data-flow*, mostrada na Fig. 3.6. Esta arquitetura mostra que há uma atribuição concorrente ao sinal `q`.

No último estágio tem-se a descrição estrutural. Outra vez a síntese da descrição *data-flow* para a estrutural é intuitiva. Podemos ver que a descrição da Fig. 3.6 implica em

uma estrutura de portas lógicas AND, OR e inversoras, cujo diagrama lógico podemos ver na Fig. 3.7a.

```
ARCHITECTURE data_flow OF mux IS
BEGIN
  q <= (a AND NOT sel) OR (b AND sel);
END data_flow;
```

Fig. 3.6 Arquitetura *data-flow* para a interface da Fig. 3.3b.

A arquitetura estrutural da Fig. 3.7b mostra uma correspondência clara com o diagrama lógico da Fig. 3.7a. Note-se a utilização das entidades `and2`, `or2` e `inversor`, declaradas na arquitetura começando com a palavra `COMPONENT`. É claro que estas entidades têm que ser definidas previamente (vide Fig. 3.8) e na falta dessas definições devem ser realizadas referências a portas padrão de alguma tecnologia específica. No corpo da descrição pode-se observar as instâncias destas entidades (`I1`, `A1`, `A2` e `O1`), assim como as ligações entre elas, representadas pelos sinais `S0`, `S1`, `S2`.

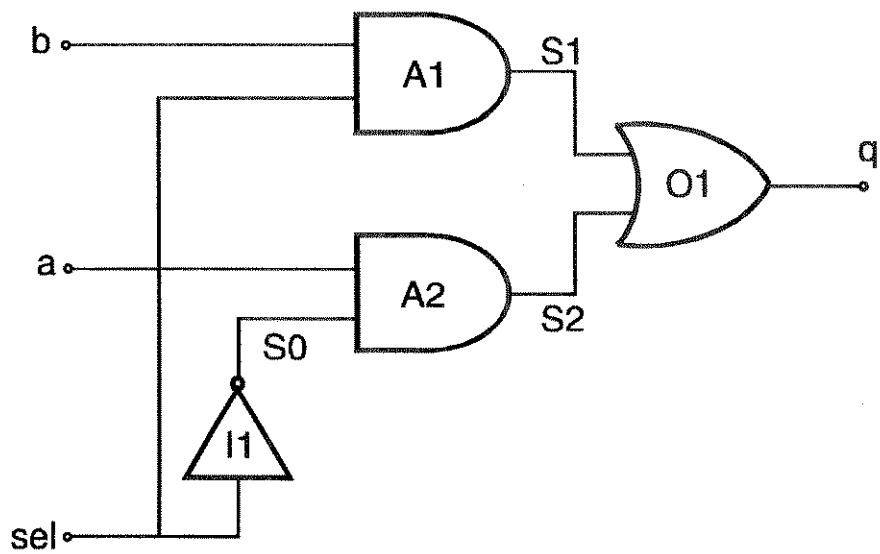


Fig. 3.7a. Diagrama lógico correspondente à arquitetura estrutural da Fig. 3.7b.

```
ENTITY mux Is
  PORT (   a, b   : IN BIT;
           sel    : IN BIT;
           q      : OUT BIT );
END mux;

ARCHITECTURE estrutural OF mux IS

  COMPONENT and2
    PORT (a, b : IN BIT; q : OUT BIT);
  END COMPONENT;

  COMPONENT or2
    PORT (a, b : IN BIT; q : OUT BIT);
  END COMPONENT;

  COMPONENT inversor
    PORT (a : IN BIT; q : OUT BIT);
  END COMPONENT;

  FOR I1 : inversor USE ENTITY WORK.inversor(data_flow);
  FOR A1, A2 : and2 USE ENTITY WORK.and2(data_flow);
  FOR O1 : or2 USE ENTITY WORK.or2(data_flow);

  SIGNAL S0, S1, S2 : BIT;

BEGIN

  I1 : inversor PORT MAP (sel, S0);

  A1 : and2 PORT MAP (b, sel, S1);

  A2 : and2 PORT MAP (a, S0, S2);

  O1 : or2 PORT MAP (S1, S2, q);

END estrutural;
```

Fig. 3.7b Arquitetura estrutural para a interface da Fig. 3.3b.

```
ENTITY inversor IS
       PORT (a : IN BIT; q : OUT BIT);
END inversor;

ARCHITECTURE data_flow OF inversor IS
BEGIN
       q <= NOT a;
END data_flow;
```

a) inversor

```
ENTITY and2 IS
       PORT (a,b : IN BIT; q : OUT BIT);
END and2;

ARCHITECTURE data_flow OF and2 IS
BEGIN
       q <= a AND b;
END data_flow;
```

b) and

```
ENTITY or2 IS
       PORT (a,b : IN BIT; q : OUT BIT);
END or2;

ARCHITECTURE data_flow OF or2 IS
BEGIN
       q <= a OR b;
END data_flow;
```

c) or

Fig. 3.8 Descrições das entidades and, or e inversor utilizados na arquitetura da Fig. 3.7b.

3.4.3 Os Processos e Os Sinais

As descrições comportamentais em VHDL dão-se fundamentalmente nos processos por meio de uma seqüencia de instruções e sua declaração começa com a palavra `PROCESS`. Um processo em VHDL pode ter uma lista sensitiva a qual define quais sinais acionarão a execução do processo. Quando algum sinal da lista sensitiva mudar de valor, ou seja acontece um evento no sinal, o processo é acionado. Se o processo não tiver uma lista sensitiva, ele deve ter uma instrução `WAIT` no seu corpo. A função desta instrução é a de provocar um retardo na execução ou a modificação dinâmica da lista sensitiva do processo.

Em um modelo VHDL, os processos são explicitamente definidos e as ligações entre eles são definidas por meio de sinais. Um modelo VHDL consiste de uma estrutura hierárquica de instâncias de componentes e processos, os quais são interconectados com sinais. Esses componentes e processos são especificados usando entidades e suas correspondentes arquiteturas. Uma arquitetura pode conter instâncias de componentes cujas portas são interconectadas por sinais.

Os processos, no seu corpo seqüencial, podem afetar os valores dos sinais, por meio das atribuições. Quando um processo atribui um valor a um sinal `s`, diz-se que o processo contém um *driver* para `s`.

A Fig. 3.9 mostra um exemplo de modelo hierárquico. A cada instância lhe corresponde o par entidade/arquitetura, cuja ampliação pode ser elaborada substituindo o corpo da respectiva arquitetura. Os sinais do corpo da arquitetura substituido, são ligados via as portas na instância, com os sinais externos.

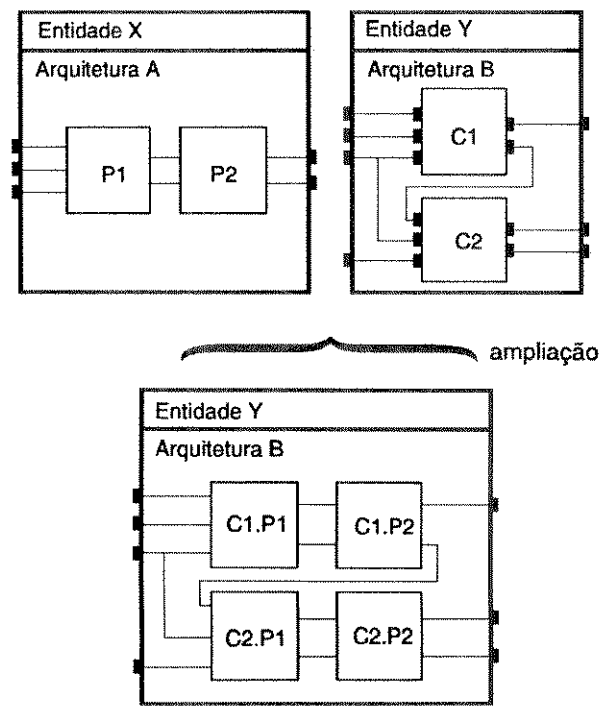


Fig. 3.9 Exemplo de modelo hierárquico.

No exemplo da Fig. 3.9, observa-se que a entidade `X` tem uma arquitetura `A` com dois processos `P1` e `P2`. Por sua vez, a entidade `Y` possui a arquitetura `B` com duas instâncias, `C1` e `C2`, correspondendo a cada uma o par `X/A`. A atribuição do corpo da arquitetura `A` para `C1` e `C2`, resulta em uma estrutura com duas instâncias de `P1` (`C1.P1` e `C2.P1`) e duas instâncias de `P2` (`C1.P2` e `C2.P2`).

Este processo de atribuição é realizado antes da simulação, tendo-se como resultado uma coleção de processos VHDL, cada um contendo *drivers* para um ou vários sinais.

3.5 Simulação em VHDL

Existem dois métodos para verificação da descrição de um sistema digital: a simulação e a verificação formal[Bou92]. A simulação compara resultados e testa as descrições do sistema, feitas previamente pelo projetista, verificando assim rigorosamente as especificações. Já a verificação formal é um método mais exaustivo.

A VHDL, que é uma linguagem essencialmente de simulação de *hardware*, suporta simulações de descrições comportamentais, descrições *data-flow* (consideradas também como RTL) e descrições a nível de portas lógicas.

Uma característica muito importante da simulação de descrições comportamentais ou de modelos de alto nível de abstração, é a eficiência. Por outro lado as descrições a nível de portas, que são simuladas muito mais lentamente que os seus equivalentes comportamentais, oferecem uma exatidão maior.

A informação da temporização do sistema digital, requerida para a escrita de descrições mais exatas, não estará disponível até que a simulação do circuito seja realizada. Em conseqüencia, o projeto de sistemas digitais gera um fluxo de informações de projeto (requerimentos, restrições, etc) dos níveis de modelação abstrata ao nível de realização do circuito. Outro fluxo, contrário, é gerado, também (principalmente da temporização), do nível de realização do circuito ao nível de portas e, possívelmente, a níveis mais altos. O processo de incorporar informações, obtidas de simulações em níveis de abstração mais baixos, nos modelos dos níveis mais abstratos, com o propósito de aumentar a precisão sem depreciar significativamente o tempo de simulação, é conhecida como de *Back-annotation*[Eck92, Nav91].

3.6 Modelamento em VHDL Visando Síntese

Apesar da VHDL ter ganhado popularidade como linguagem de modelamento, simulação e documentação, ela tem vários problemas quando é utilizada como linguagem de síntese. As características da VHDL tais como o suporte de estruturas complexas de dados, aritmética de números reais e manuseio de arquivos, tornam-a conveniente para o modelamento de sistemas complexos, porém isto exige características muito complexas das ferramentas de síntese. Seguindo algumas regras de modelamento, dependendo do nível de descrição, é possível utilizar a VHDL como ferramenta de síntese.

Como já foi mencionado, o nível arquitetural permite ao projetista modelar o comportamento de um sistema digital, por meio de processos, que por sua vez podem incluir

várias instruções seqüenciais. As ferramentas de síntese suportam muitas instruções seqüenciais, porém existem algumas restrições dependendo da ferramenta específica. A seguir enumeram-se algumas das restrições[Nag91] que comumente apresentam as ferramentas de síntese no nível arquitetural (HLS):

1. As instruções do tipo "`WAIT FOR 10.2 ns;`" não são permitidas, devido à ferramenta de síntese não poder garantir que o circuito implementado apresente o atraso especificado.

2. As atribuições a sinais, no interior dos processos, são permitidas apenas aos sinais especificados na interface.

3. As declarações de procedimentos dentro de um processo, são sintetizadas como módulos separados. A VHDL permite a uma arquitetura ter múltiplos processos, porém, algumas ferramentas de síntese impõem a restrição de que apenas um processo possa ser modelado.

4. Um processo é executado assíncrona e concorrentemente junto com outros processos na descrição. Como não é possível conhecer o tempo total de execução do processo antes da síntese, as descrições devem conter indicações de comunicação inter-processo. Com isto assegura-se que o projeto seja simulado corretamente.

No nível RT a VHDL proporciona várias formas de descrever um mesmo sistema digital. As ferramentas de síntese RTL freqüentemente "inferem" a intenção do projetista, a partir de seu estilo de descrição. Por exemplo, a VHDL não possui construções específicas para descrever elementos tais como registradores ou *latches*. A ferramenta de síntese infere estes elementos das atribuições a sinais e variáveis sob condições das bordas do sinal de relógio. Portanto, o projetista deverá tomar o cuidado de não fazer atribuições a variáveis e sinais em ambas as bordas do sinal de relógio. A VHDL também não possui construções especiais para a descrição de máquinas de estado, embora elas sejam muito utilizadas nos projetos. Neste caso, a ferramenta de síntese também deverá inferir a máquina de estados a partir das descrições VHDL.

No processo de projeto requer-se especificar as restrições de projeto, como por exemplo, limites mínimos e máximos ou faixa de valores aceitáveis para os parâmetros, os quais devem ser atingidos com exatidão. Um exemplo típico, pode ser achado na síntese lógica, onde é necessário especificar os tempos de atraso mínimo e máximo. Na síntese HLS, podem também existir restrições de mínimo e máximo em termos de ciclos de relógio.

Em VHDL, é necessário especificar a faixa de valores de cada `type` ou `subtype`. Como a VHDL é uma linguagem baseada em um conceito estrito de tipo, são feitas revisões de tipo durante a compilação de código e verificações em tempo de execução na simulação. Então, pode-se utilizar o recurso do `subtype` para a especificação das faixas de valores nos parâmetros de projeto, para serem usadas como restrições de projeto[Eck92]. Para tanto, será necessário utilizar o conceito de *back-annotation*, para os valores dos parâmetros, antes e depois da síntese.

3.7 O Ambiente *Falcon Framework* da *Mentor Graphics*

O ambiente *Falcon Framework* da empresa *Mentor Graphics*, disponível nos laboratórios do DSIF/FEE/UNICAMP, é um conjunto de ferramentas voltadas ao Projeto Assistido por Computador (PAC). Na Fig.3.10 ilustra-se o fluxo de projeto neste ambiente.

Começando pela especificação informal, o projetista realiza um particionamento estrutural hierárquico do sistema digital sendo projetado. Depois, cada bloco é definido por meio de uma captura de esquemático ou uma descrição VHDL, seja esta comportamental, RTL ou a nível de portas. A ferramenta utilizada neste estágio é o ***Design Architect.***

No seguinte estágio vem a simulação, com a finalidade de comprovar a exatidão das descrições VHDL. Se houver necessidade, pode-se voltar ao estágio anterior para o refinamento ou correção das descrições. A simulação VHDL pode ser puramente comportamental, *data-flow* ou a nível de portas. É freqüente ter simulações mistas. O ***QuickSim II*** é a ferramenta utilizada neste estágio.

A ferramenta ***AutoLogic*** é aquela que gera uma base de dados a nível de portas e em uma tecnologia específica, do sistema digital, partindo das descrições VHDL simuladas no estágio anterior. Ela também inclui as restrições do *timing*, na base de dados gerada.

Após a verificação da estrutura do projeto no que se refere a *fanout*, *fanin*, lógica por conexão (*or, and*) por meio da ferramenta ***QuickCheck,*** é realizada a simulação do *timing*, outra vez com a ferramenta *QuickSim II.* Neste ponto, é possível voltar ao primeiro estágio para correções ou refinamento das descrições. Caso não haja modificações a fazer, o processo de projeto continua com as ferramentas PAC padrão do domínio físico.

No fluxo de projeto mostrado, apresentaram-se alguns passos críticos. A verificação da equivalência entre a especificação e a implementação é feita por meio da simulação, a qual não garante uma revisão exaustiva das possibilidades de funcionamento e depende fortemente da escolha dos vetores padrão de teste. A simulação do *timing* tem um alto custo computacional. O processo de síntese também tem um alto custo computacional, além do requerimento das restrições de projeto bem definidas.

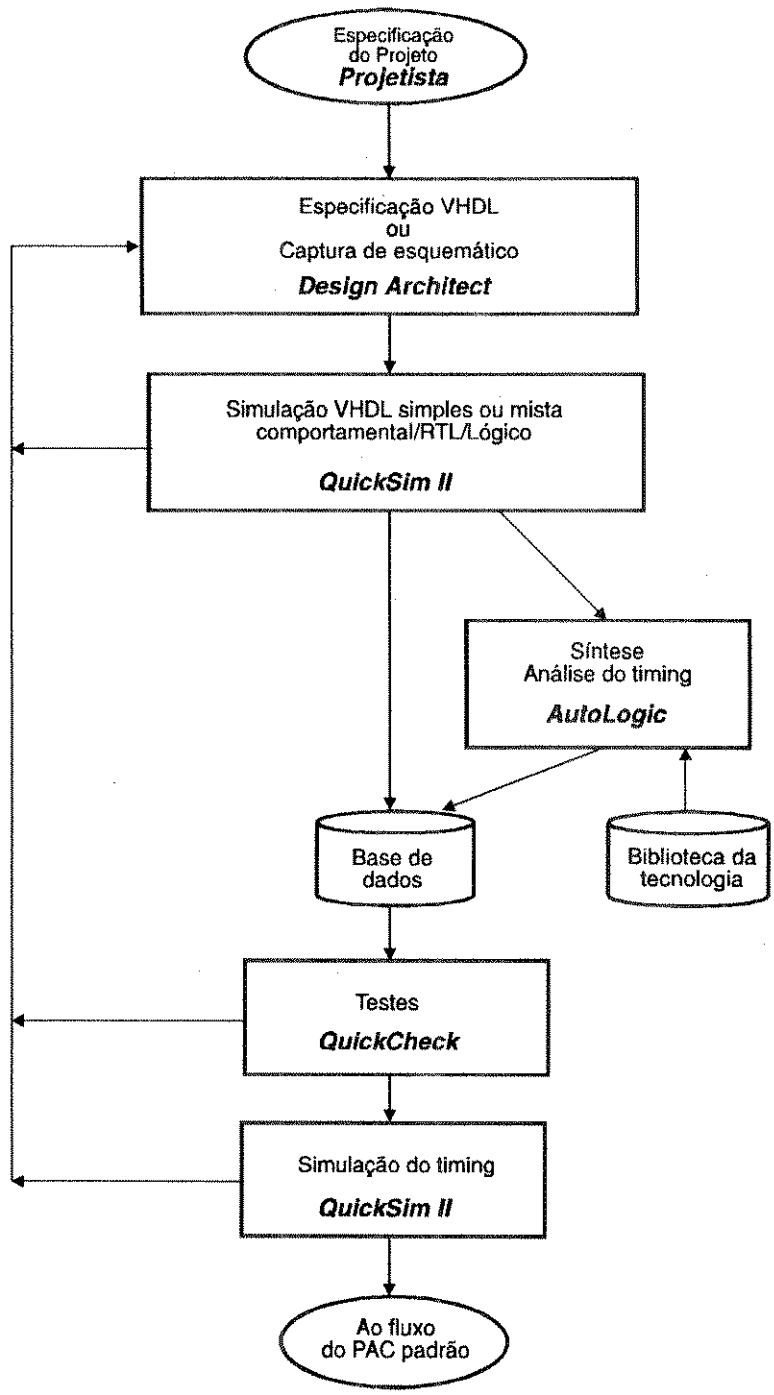


Fig. 3.10 Fluxo de projeto no ambiente *Falcon Framework* da *Mentor Graphics*.

Capítulo 4

Modelamento e Simulação em VHDL do Sistema ABACUS

No presente capítulo vamos apresentar o modelamento do sistema ABACUS realizado na linguagem de descrição VHDL, com a finalidade de simulá-lo detalhadamente e objetivando a sua implementação física. Para tanto nos basearemos nas descrições dadas no capítulo 2.

O modelamento foi feito em dois níveis de abstração das descricões do sistema. No primeiro nível com descrições comportamentais, preocupamo-nos mais com o funcionamento do sistema, descrevendo-o comportamentalmente. Já no segundo nível, com descrições RTL, mostra-lo-emos com mais detalhes, obtendo uma idéia da arquitetura e dos blocos que a conformam. Em ambos os níveis foram avaliados os tempos de simulação dos modelos dos dispositivos contemplados.

O procedimento de modelamento e simulação empregado, segue as duas primeiras fases do diagrama de fluxo mostrado na Fig. 3.10 do capítulo anterior. Ou seja, numa primeira fase realiza-se a especificação VHDL ou a captura esquemática por meio da ferramenta *Design Architect*. Nesta fase inclui-se a compilação do código VHDL gerado. Na segunda fase realiza-se a simulação VHDL, seja esta comportamental ou RTL, por meio da ferramenta *QuickSimII*.

4.1 Primeiro Nível de Abstração (Comportamental)

No capítulo 2 foi mencionado que o sistema ABACUS consta de um processador gerenciador (hospedeiro), um arranjo de processadores de modelos (MPHs) e uma rede de interconexão formada por chaves. Também foi discutida a estrutura dos canais de comunicação: entre o hospedeiro e os MPHs; entre o hospedeiro e as chaves; e, entre os MPHs por meio das chaves da rede de interconexão.

Existem três processos diferenciados na operação do sistema ABACUS. O primeiro é o de interfaceamento com o usuário, análise do circuito a simular e configuração dos MPHs e das chaves. O segundo é o da simulação propiamente dita. O terceiro é o de leitura dos

resultados nos MPHs, em cada instante da simulação.

No primeiro e terceiro processos são utilizados os barramentos de dados, endereços e controle para a transferência de dados entre o hospedeiro, os MPHs e as chaves. No segundo processo, são utilizados apenas os canais de comunicação entre os MPHs (por meio das chaves), além dos sinais que indicam a convergência local em cada MPH (`conv`) e dos sinais de controle utilizados pelo hospedeiro para a inicialização do sistema todo e inicialização de cada instante da simulação (`RESET` e `START`).

Para simplificar o modelamento, o sistema ABACUS será descrito considerando nas interfaces de seus blocos, apenas os sinais utilizados no segundo processo mencionado no parágrafo anterior. A configuração dos MPHs e das chaves será realizada durante a fase de simulação, acessando diretamente os sinais que armazenam essas configurações.

O modelamento do sistema ABACUS, neste nível de abstração, foi dividido em três blocos: a unidade de processamento de modelos, o elemento de interconexão e o processador gerenciador. Finalmente, juntando estes blocos conforma-se o sistema.

4.1.1 Descrição dos Elementos de Processamento de Modelos (MPHs)

No diagrama da Fig. 4.1 observa-se a interface de um elemento de processamento de modelos; sua descrição em VHDL é mostrada na Fig. 4.2.

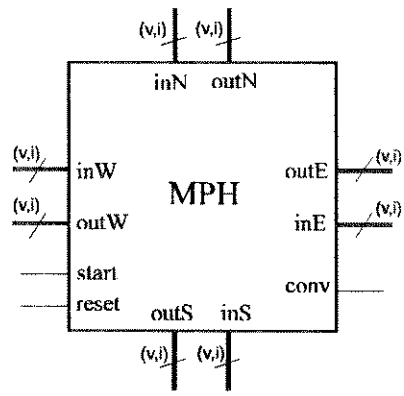


Fig. 4.1 Interface de um elemento de processamento de modelos(MPH).

Os pares de sinais `inW/outW`, `inN/outN`, `inE/outE` e `inS/outS` são utilizados como entradas/saídas dos canais oeste, norte, este e sul respectivamente. O sinal `reset` inicializa o MPH, enquanto que `start` começa um novo instante da simulação. A convergência local é indicada pelo sinal `conv`.

Os sinais `reset`, `start` e `conv` têm uma estrutura simples e são declarados como sendo de tipo `BIT`, isto quer dizer que podem adotar os valores lógicos '0' e '1'. Entretanto, a estrutura dos sinais utilizados para os canais é mais complexa, chamada de `vc_base`, cuja declaração se mostra a seguir:

```
CONSTANT VOLT : Integer := 0;
CONSTANT CURR : Integer := 1;
TYPE vc_base IS ARRAY (VOLT TO CURR) OF REAL;
```

Esta estrutura consiste de dois números com representação em ponto flutuante, um para a tensão e outro para a corrente. Por exemplo, o sinal `inW` tem uma componente `VOLT` (tensão) e outra `CURR` (corrente). É importante salientar que neste nível de abstração esses sinais ainda não são tratados em formato binário, o que facilita o processo de modelamento.

```
USE WORK.aba_dados.ALL;

ENTITY mph IS

    PORT (
          -- entradas oeste, norte, este e sul
          inW, inN, inE, inS      : IN vc_base;
          -- entradas oeste, norte, este e sul
          outW, outN, outE, outS  : OUT vc_base;
          -- inicializa o MPH
          reset    : IN BIT;
          -- começa o seguinte instante da simulação
          start    : IN BIT;
          -- indica o estado de convergência local
          conv     : OUT BIT;
          );

END mph;
```

Fig. 4.2 Descrição VHDL da interface de um elemento de processamento de modelos (MPH)

A arquitetura VHDL de um MPH tem uma descrição comportamental e corresponde aos algoritmos mostrados nas Figs. 2.7 e 2.9. Essa arquitetura consta de dois processos: `start_cont` e `mph_behav`. O primeiro verifica a presença de bordas de subida nos sinais `reset` e `start` para a inicialização do MPH ou para o início de próximo instante da simulação, respectivamente. O segundo é o processo principal, executado quando há alguma mudança nos sinais de entrada `inW`, `inN`, `inE` ou `inS`. Este processo calcula as tensões e correntes `v(POLE)` e `i(POLE)` em cada polo do dispositivo simulado (vide cap. 2) em função das tensões e correntes que chegam dos MPHs vizinhos; avalia o modelo correspondente, obtendo novos valores de tensão e corrente nos polos; analiza o erro existente entre esses valores e determina o estado da convergência. Os novos valores de tensão e corrente são colocados nos canais de saída (FESs de saída), para serem lidos pelos MPHs a eles ligados.

Inclui-se na descrição VHDL do MPH uma estrutura correspondente ao bloco MEL, o qual é preenchido, antes de começar a simulação, com a configuração relativa à topologia do entorno do elemento simulado com os elementos vizinhos (vide apêndice 3). No apêndice 1 tem-se uma listagem completa das descrições dessa arquitetura.

O diagrama de tempos da Fig. 4.3 mostra os resultados da simulação de um MPH. Os tempos de processamento dos modelos dos dispositivos são essencialmente aqueles mostrados na tabela 2.2 do capítulo anterior.

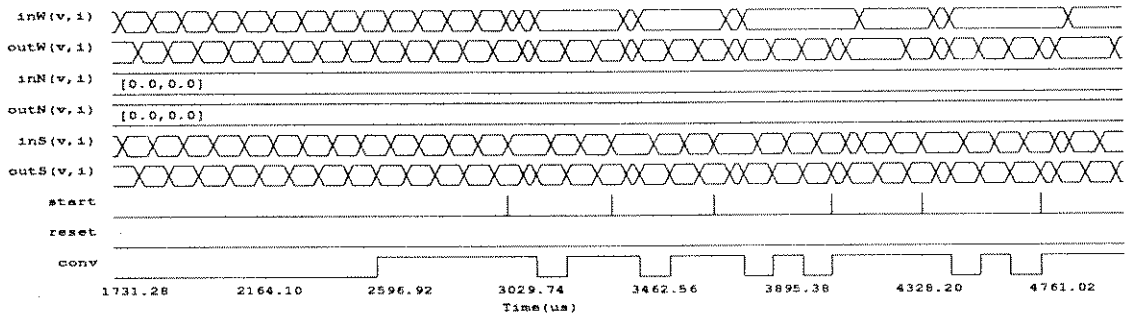


Fig. 4.3 Diagrama de tempos de um elemento de processamento de modelos (MPH).

4.1.2 Descrição dos Elementos de Interconexão (chaves)

A rede de interconexão é formada por elementos chamados de chaves. Na arquitetura mencionada no capítulo 2 observou-se que existem três tipos de chaves: quadradas, pentagonais e hexagonais. Com a finalidade de simplificar o modelamento consideraremos apenas os caso das chaves quadradas, pois as outras são similares.

A interface de uma chave quadrada, que chamaremos simplesmente chave, mostra-se na Fig. 4.4 e sua descrição VHDL na Fig. 4.5.

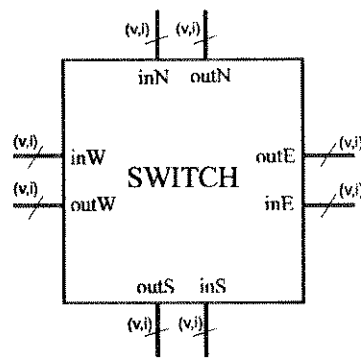


Fig. 4.4 Interface de uma chave quadrada.

Como no caso da interface dos MPHs, nesta interface tem-se quatro sinais de entrada e quatro de saída correspondentes aos canais oeste, norte, este e sul. Esses sinais são também de tipo `vc_base`, pois é necessária sua compatibilidade com aqueles dos MPHs.

Uma descrição comportamental implementa a arquitetura correspondente à interface da Fig. 4.5. Essa descrição compreende básicamente as operações de transferência de dados entre os canais indicados pelo registrador `reg_config`. Este registrador com uma dimensão de quatro bits é declarado na arquitetura e preenchido com a informação correspondente aos canais que trocam dados (vide apêndices 1 e 3 respectivamente).

```
WORK.aba_dados.ALL;

ENTITY switch IS

     PORT (
          -- entradas oeste, norte, este e sul
          inW, inN, inE, inS      : IN vc_base;
          -- saídas oeste, norte, este e sul
          outW, outN, outE, outS : OUT vc_base
          );

END switch;
```

Fig. 4.5 Descrição VHDL da interface de uma chave quadrada.

Pode-se observar no diagrama de tempos obtido após a simulação da chave quadrada, que os atrasos são pequenos, isto considerando que uma chave é composta principalmente por blocos multiplexadores e demultiplexadores (vide Fig. 4.6).

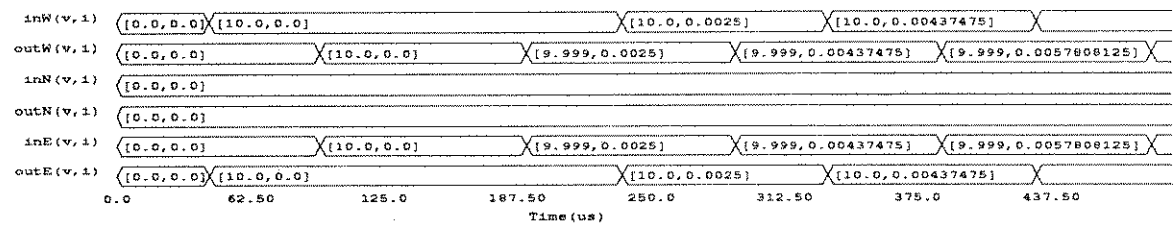


Fig. 4.6 Diagrama de tempos de uma chave quadrada.

4.1.3 Descrição do Processador Gerenciador (HOST)

O procedimento de configuração do arranjo de processadores e da rede de interconexão, o interfaceamento com o usuário e a verificação da convergência do arranjo, podem ser realizados por um computador hospedeiro que tenha as características descritas no capítulo 2. Para efeitos de teste, nas descrições a seguir consideran-se no processador gerenciador, apenas as operações de inicialização e de verificação da convergência do arranjo.

Note-se, nas interfaces das Figs. 4.7 e 4.8, a utilização apenas dos sinais `reset`, `start` e `conv`, de tipo `BIT`.

As descrições comportamentais da arquitetura do gerenciador seguem em parte os algoritmos mostrados nas Figs. 2.2 e 2.4. Foram consideradas na arquitetura os processos HCP e AMP. O processo AMP essencialmente se dedica à verificação da convergência do arranjo, verificando para tanto a presença de níveis lógicos '1' no sinal `conv`, o qual é obtido

a partir da realização de uma operação `and` de todos os MPHs do arranjo. Caso o sinal `conv` apresente um '1', ao sinal `start` é atribuido um pulso de duração `Pdelay` e a contagem do tempo é incrementada em um passo de simulação. O sinal `reset` é utilizado apenas no primeiro instante da simulação, com a finalidade de inicializar todos os MPHs do arranjo.

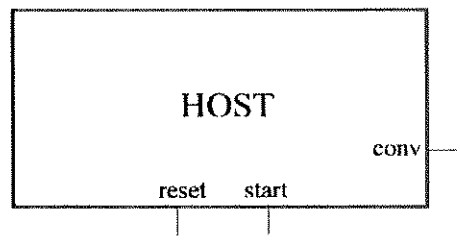


Fig. 4.7 Interface do processador gerenciador.

```
USE WORK.aba_dados.ALL;

ENTITY host IS

     PORT (
          -- inicializa o arranjo de MPHs
          reset   : OUT BIT;
          -- começa o seguinte instante da simulação
          start   : OUT BIT;
          -- indica o estado de convergência global
          conv    : IN BIT;
          );

END host;
```

Fig. 4.8 Descrição VHDL da interface do processador gerenciador.

4.1.4 Conformação do Sistema ABACUS.

Nos parágrafos anteriores foram apresentadas as descrições dos MPHs, das chaves e do gerenciador. Com estes três elementos será formada a estrutura para a simulação de alguns circuitos teste e um arranjo pequeno para a simulação de circuitos com um número reduzido de elementos.

Dadas as descrições das interfaces mencionadas anteriormente é possível gerar símbolos para os MPHs, chaves e o processador gerenciador, por meio da ferramenta *Design Architect*. Também captura-se esquemáticos correspondentes a circuitos teste ou mesmo um arranjo de MPHs por exemplo de 3x3 processadores.

Um primeiro exemplo de circuito teste é o de um retificador de meia onda, cujo diagrama se observa na Fig. 4.9. Uma estrutura para esse circuito mostra-se na Fig. 4.10. As ligações entre o MPH que simula a fonte V e o MPH que simula o diodo D, são chamadas de `A(v,i)` e `B(v,i)`. `C(v,i)` e `D(v,i)` são as ligações entre o MPH-diodo e o MPH-resistor.

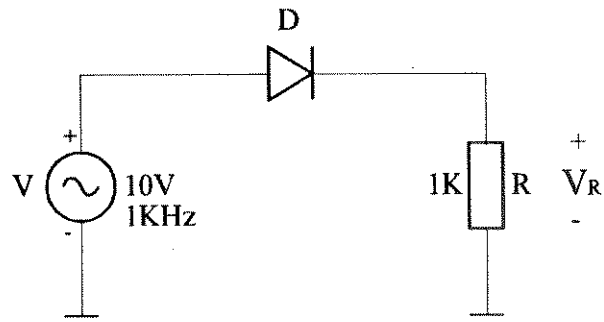


Fig. 4.9 Circuito retificador de meia onda.

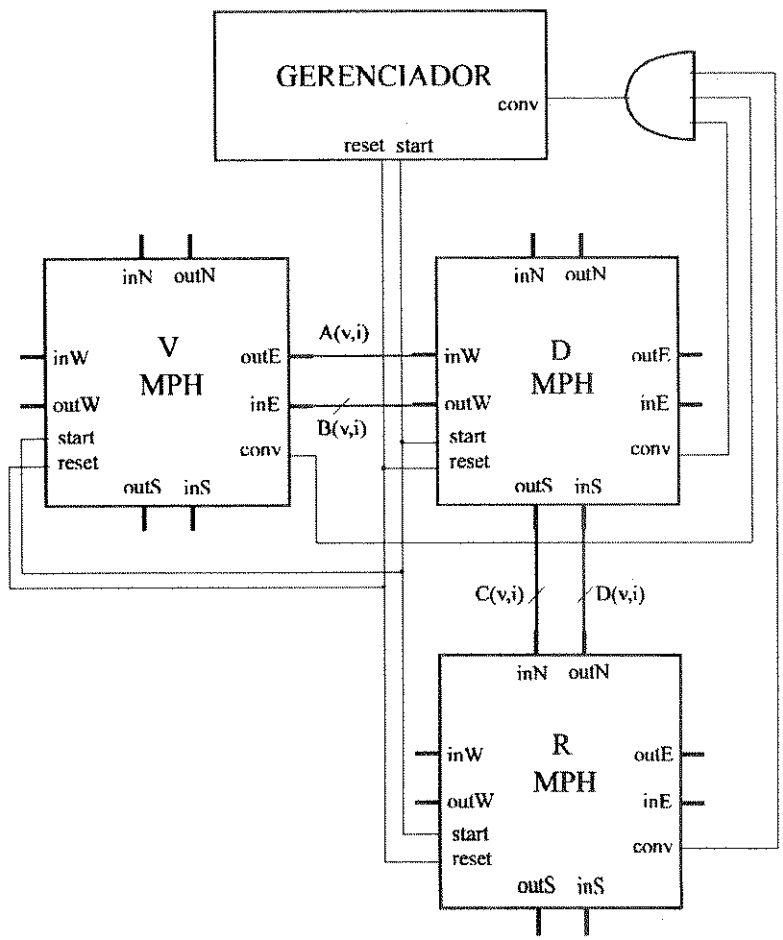


Fig. 4.10 Arranjo para a simulação do circuito retificador de meia onda.

O diagrama de tempos da Fig. 4.11 mostra os resultados da simulação do circuito retificador de meia onda. O diagrama da Fig. 4.12 mostra as tensões na fonte (V) e no resistor (V_R), verificando desta forma a funcionalidade do arranjo da Fig. 4.10.

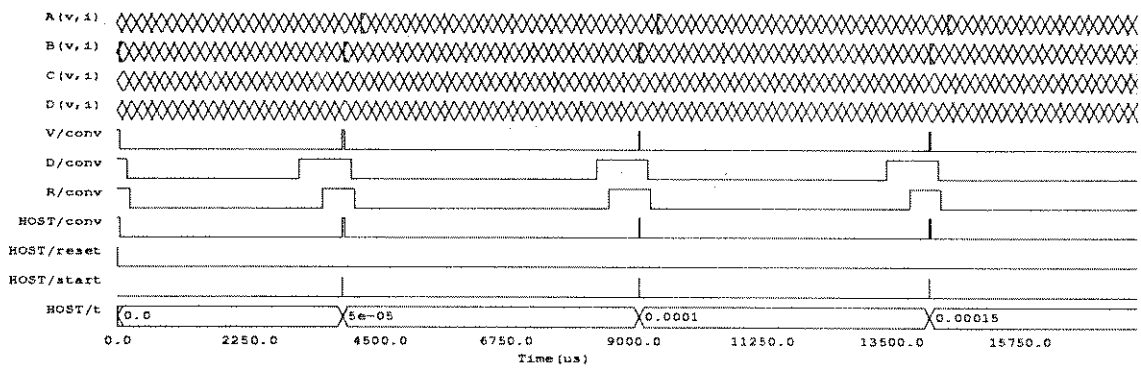


Fig. 4.11 Diagrama de tempos do circuito retificador de meia onda.

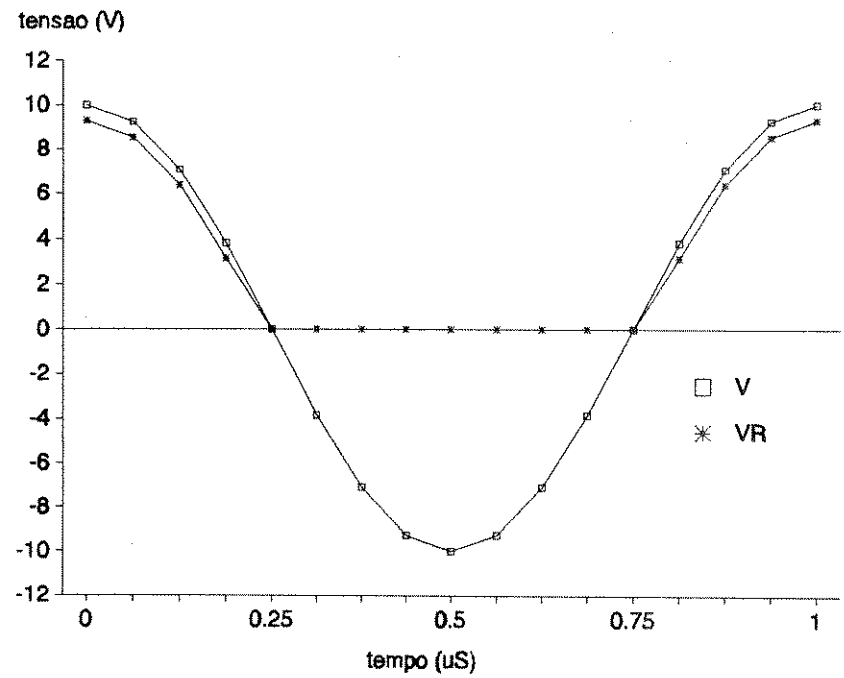


Fig. 4.12 Resultados da simulação do circuito retificador de meia onda. Curvas das tensões V e V_R.

O circuito filtro passa-altas mostrado na Fig. 4.13 é utilizado como segundo exemplo de circuito teste. Também neste caso utilizou-se um arranjo particular (vide Fig. 4.14). `A(v,i)` e `B(v,i)` são as ligações entre o MPH-fonte e o MPH-capacitor. `C(v,i)` e `D(v,i)` são as ligações entre o MPH-capacitor e o MPH-resistor. As formas de onda deste circuito mostram-se na Fig. 4.15 e as curvas das tensões V, V_R e V_C, resultantes da simulação na Fig. 4.16.

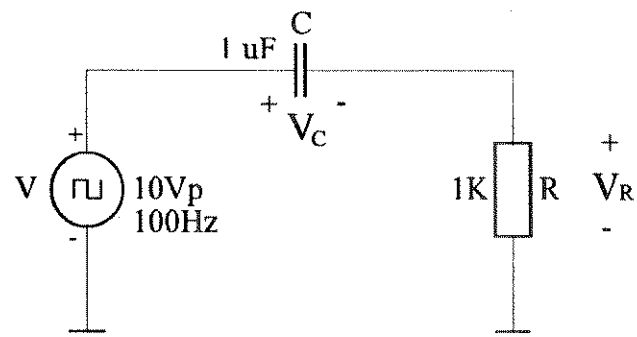


Fig. 4.13 Circuito filtro passa-altas.

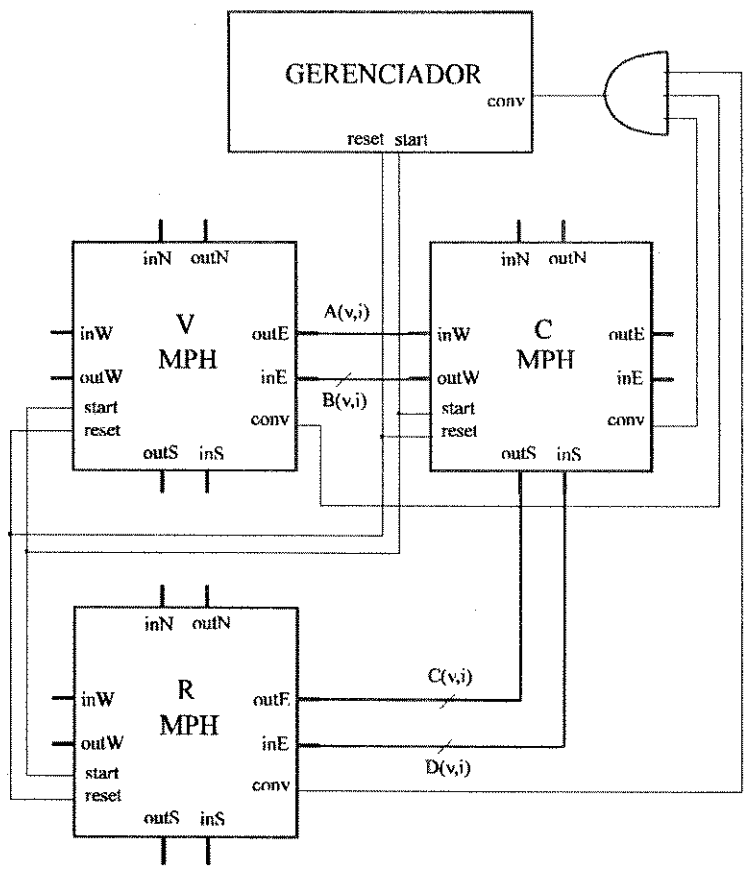


Fig. 4.14 Arranjo para a simulação do circuito filtro passa-altas.

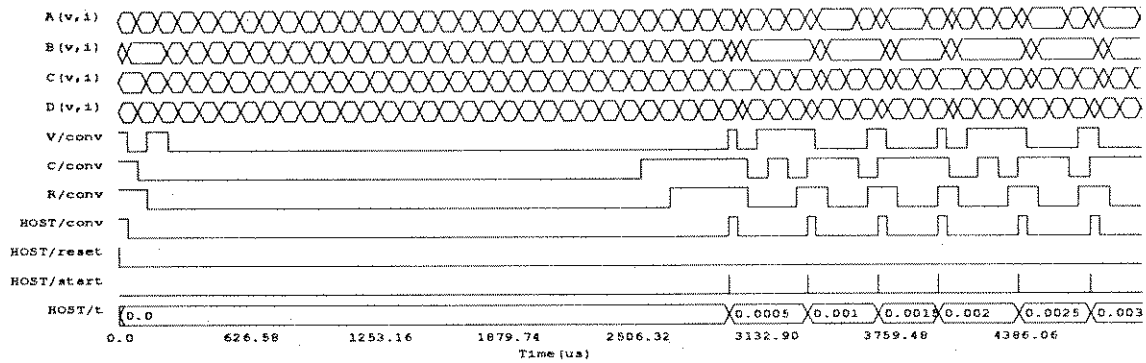


Fig. 4.15 Diagrama de tempos do circuito filtro passa-altas.

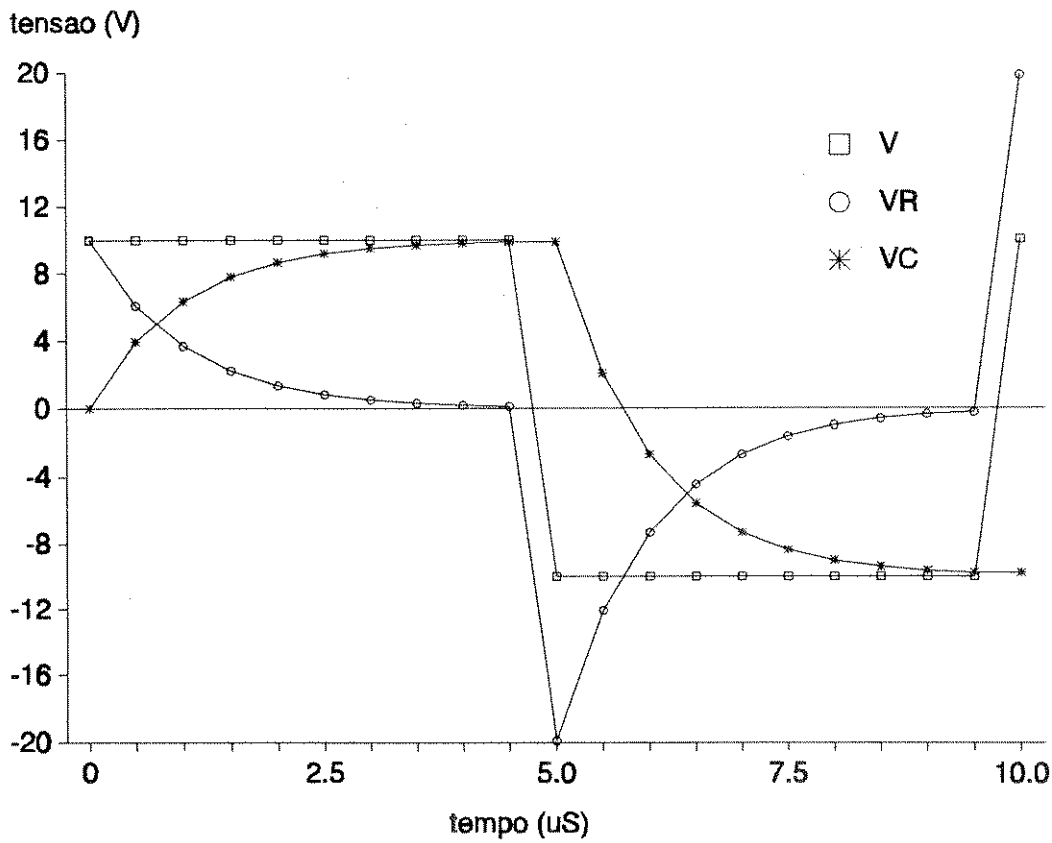


Fig. 4.16 Resultados da simulação do circuito filtro passa-altas. Curvas das tensões V, V_C e V_R.

O ideal seria simular um arranjo com dimensão suficiente para suportar circuitos com um número grande de elementos, como foi mencionado no capítulo 2. Devido a limitações tanto de tempo de simulação quanto de capacidade de memória do computador (Estação de trabalho SPARC 2 com 20 MB de memória RAM) onde foram feitas as simulações isto não foi possível. No entanto, foi implementado um arranjo de uma camada com uma dimensão de 3x3 MPHs e uma rede de interconexão formada por chaves quadradas. Essa estrutura é mostrada na Fig. 4.17. Nela podem ser simulados circuitos de até 9 elementos.

Fizeram-se testes com diversos circuitos lineares e não lineares. Os circuitos retificador de meia onda e filtro passa-altas foram simulados novamente neste arranjo. Na Fig. 4.18 observa-se o diagrama de tempos resultante da simulação do circuito retificador de meia onda. Note-se a semelhança deste diagrama de tempos com aquele da Fig. 4.11. Neste caso foram utilizados o MPH11 como a fonte de tensão senoidal, o MPH12 como o diodo retificador e o MPH22 como o resistor. Na Fig. 4.19 mostram-se os resultados da simulação do circuito filtro passa-altas, obtendo-se resultados similares aos do diagrama da Fig. 4.16. Utilizaram-se também neste caso o MPH11 como fonte de tensão quadrada, o MPH12 como capacitor e o MPH21 como resistor. Em ambos os casos foram configuradas diversas chaves para a interligação entre os processadores. No apêndice 3 mostram-se os arquivos que contém as ditas configurações e que são carregados na fase de simulação.

No arranjo da Fig. 4.17 considere-se o caso da interligação mais longa entre dois MPHs, por exemplo o MPH11 e o MPH33. Para que um dado chegue do MPH11 ao MPH33 ele terá que passar por uma certa quantidade de chaves, neste caso 15. Isto significa que haverá um atraso de 300 ns, considerando que cada chave essencialmente consta de um bloco multiplexador e outro demultiplexador cada um com um atraso de 10 ns. Por outro lado, para o processamento do modelo de um dispositivo são gastos no mínimo 45950 ns (modelo do resistor vide tabela 2.2). Podemos concluir que o tempo de transmissão de um dado de um MPH a outro, no pior dos casos, representa 0.6% do tempo de processamento do modelo menos complexo. É claro que num arranjo maior essa parcela será maior. Com o mesmo raciocínio, no caso de um arranjo de 32x32x32 o número de chaves existentes entre dois MPHs que tenham a interligação mais longa, um deles num vértice da camada superior do cubo e o outro no vértice oposto da camada inferior, pode ser calculado assim:

$$2 (PE + 2CH) + L = \text{Número de chaves}$$

onde PE é o número de MPHs em uma linha de uma camada, L é o número de camadas e CH é o número de grupos de chaves de acesso a outras camadas. Com PE=32, L=32 e CH=32 obtém-se 224 chaves. Portanto, o tempo gasto na transmissão de um dado será de 4.480 ns, o que representa 9,7% do tempo de processamento do modelo menos complexo. A média será obviamente menor.

Por outro lado, a implementação ideal dos canais de comunicação entre dois MPHs seria por meio de uma transmissão em paralelo dos dados que contêm a informação das tensões e correntes, e que estão em formato binário. A largura desses canais poderia ser de 32 bits como foi dito no capítulo 2. O dimensionamento desses canais depende em parte da parcela de tempo analisada no parágrafo anterior. Com os valores calculados acima pode-se considerar o caso da transmissão em série dos dados, opção esta que seria vantajosa do ponto de vista do número de pinos necessários à interface de um MPH.

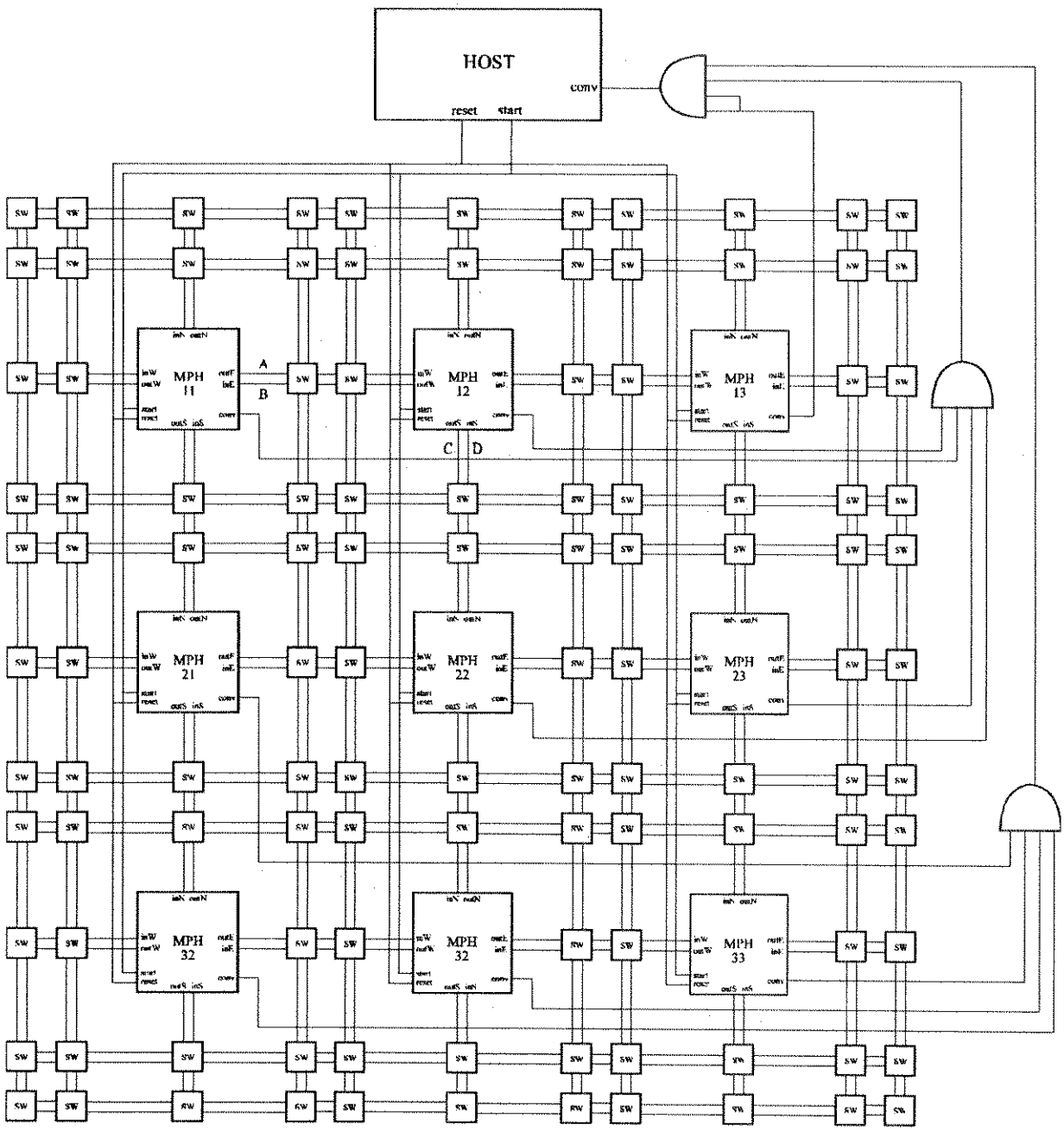


Fig. 4.17 Estrutura de 3x3 MPHs para a simulação de circuitos.

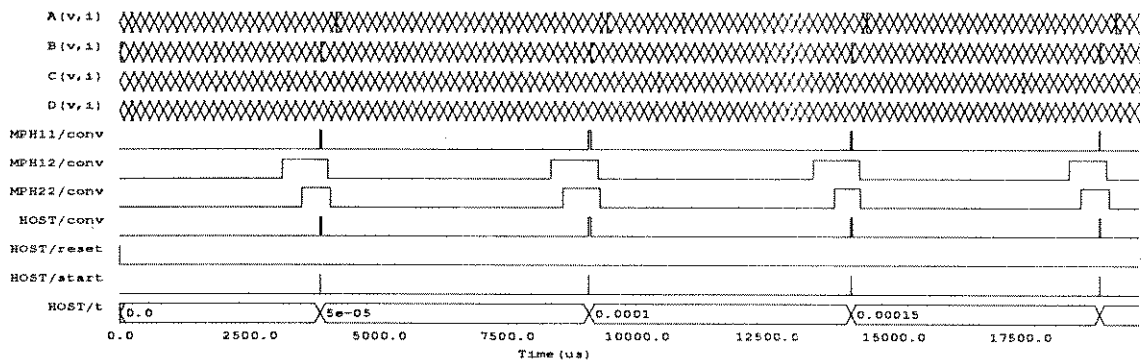


Fig. 4.18 Diagrama de tempos do circuito retificador de meia onda simulado no arranjo de 3x3 MPHs.

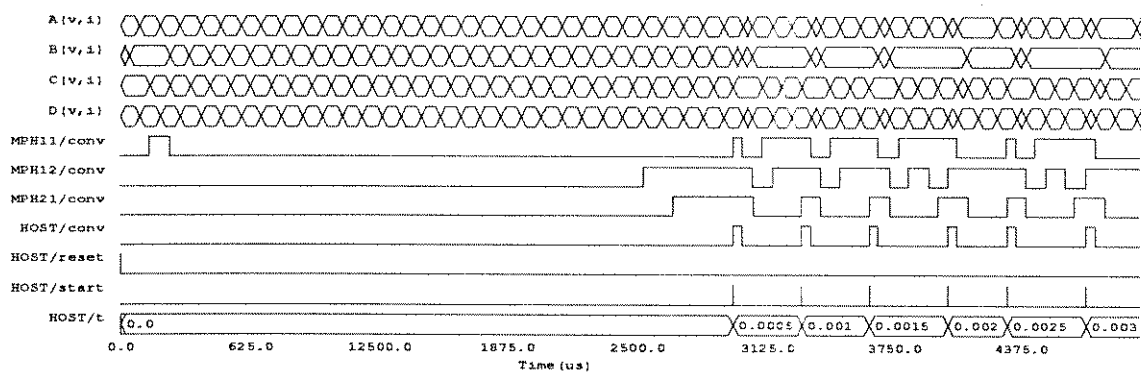


Fig. 4.19 Diagrama de tempos do circuito filtro passa-altas simulado no arranjo de 3x3 MPHs.

4.2 Segundo Nível de Abstração (RTL)

O modelamento no segundo nível de abstração centrou-se principalmente nos elementos de processamento de modelos (MPHs) e nos elementos de interconexão (chaves). Em cada caso foi evidenciada uma arquitetura e no caso dos MPHs foi modelado um esquema de temporização sob o qual a unidade de controle elementar (UCE) aciona uma máquina de estados. Note-se que essas características eram transparentes no primeiro nível de abstração.

As descrições realizadas neste nível caem na classificação de RTL (*Register-Transfer Level*), pois o modelamento é realizado considerando o fluxo de dados entre os blocos que conformam tanto os MPHs quanto as chaves. No entanto, a maior parte desses blocos ainda é descrita comportamentalmente.

4.2.1 O Elemento de Processamento de Modelos (MPH)

A arquitetura básica de um elemento de processamento de modelos foi mostrada no capítulo 2 (Fig. 2.6). Neste nível de abstração com base nessa arquitetura apresenta-se uma versão ampliada, como é mostrado na Fig. 4.20. Esta nova arquitetura é formada basicamente pelos mesmos blocos. Nela têm-se as quatro FESs de entrada, a ULA, a UCE e a MEL da arquitetura anterior, além dos blocos novos chamados de IN-PROC, REG-VI, CMP, OUT-PROC, CLOCK e OR4. A memória onde são armazenados os microprogramas dos modelos dos dispositivos (UMA), neste nível de abstração encontra-se "dispersa" na maior parte dos blocos e embutida principalmente na UCE e na ULA. Todas as FESs de saída foram inseridas no bloco OUT-PROC para facilitar o modelamento e porque elas são formadas fundamentalmente por registradores.

Com base nas descrições dos MPHs do primeiro nível e do fluxo de dados foram gerados os blocos novos mencionados no parágrafo anterior. Os sinais principais para a interconexão dos blocos têm a estrutura de pares de tensão/corrente (tipo `vc_base`) nos blocos de entrada, e de tensões e correntes por separado nos blocos internos. Na arquitetura da Fig. 4.20 mostra-se cada um dos blocos antes mencionados incluindo-se os sinais das suas interfaces, destacando os sinais utilizados para sua interligação. Todavia neste nível esses sinais não foram tratados em formato binário, pois isto depende fortemente da arquitetura da ULA, a qual ainda é considerada como um bloco. Deve-se salientar que a consideração desses sinais como simples números é um recurso muito poderoso da linguagem VHDL, pois isso facilita em grande maneira o modelamento.

Os atrasos em cada um dos blocos da nova arquitetura foram considerados em base dos tempos de execução das operações aritméticas de ponto flutuante neles incluídos e foram considerados similares aos do Transputer T-800 da INMOS[INM88] (vide tabela 2.1).

4.2.1.1 Os Blocos FES de Entrada

Os blocos **FES**s de entrada, como foi mencionado no capítulo 2, encarregam-se da recepção dos dados vindos dos MPHs vizinhos correspondentes às tensões e correntes. Na Fig. 4.21 mostra-se a descrição VHDL da interface destes blocos. Note-se que a estrutura dos sinais de entrada (`vi_in`) e saída (`vi_out`) de cada FES é do tipo `vc_base`.

A descrição comportamental da arquitetura de uma FES (vide apêndice 2) compreende a deteção de mudanças (eventos) na sua entrada `vi_in`. Quando um evento é detetado na entrada, o sinal `ready` sinaliza à UCE por meio do sinal `rdyW` (caso da FES oeste) apresentando o valor lógico '1'. Os resultados da simulação mostram-se no diagrama de tempos da Fig. 4.22. Observe-se que o atraso produzido no sinal de saída `vi_out` é pequeno e semelhante ao retardo de um registrador CMOS.

4.2.1.2 O Bloco OR4

O bloco **OR4** realiza uma operação OR dos sinais `rdyW`, `rdyN`, `rdyE` e `rdyS` vindos das quatro FESs e o sinal resultante `rdyFES` é passado ao bloco UCE para o início de uma nova iteração. É utilizada esta função lógica devido a que uma mudança nas entradas de qualquer uma FES é suficiente para dar início a uma nova iteração.

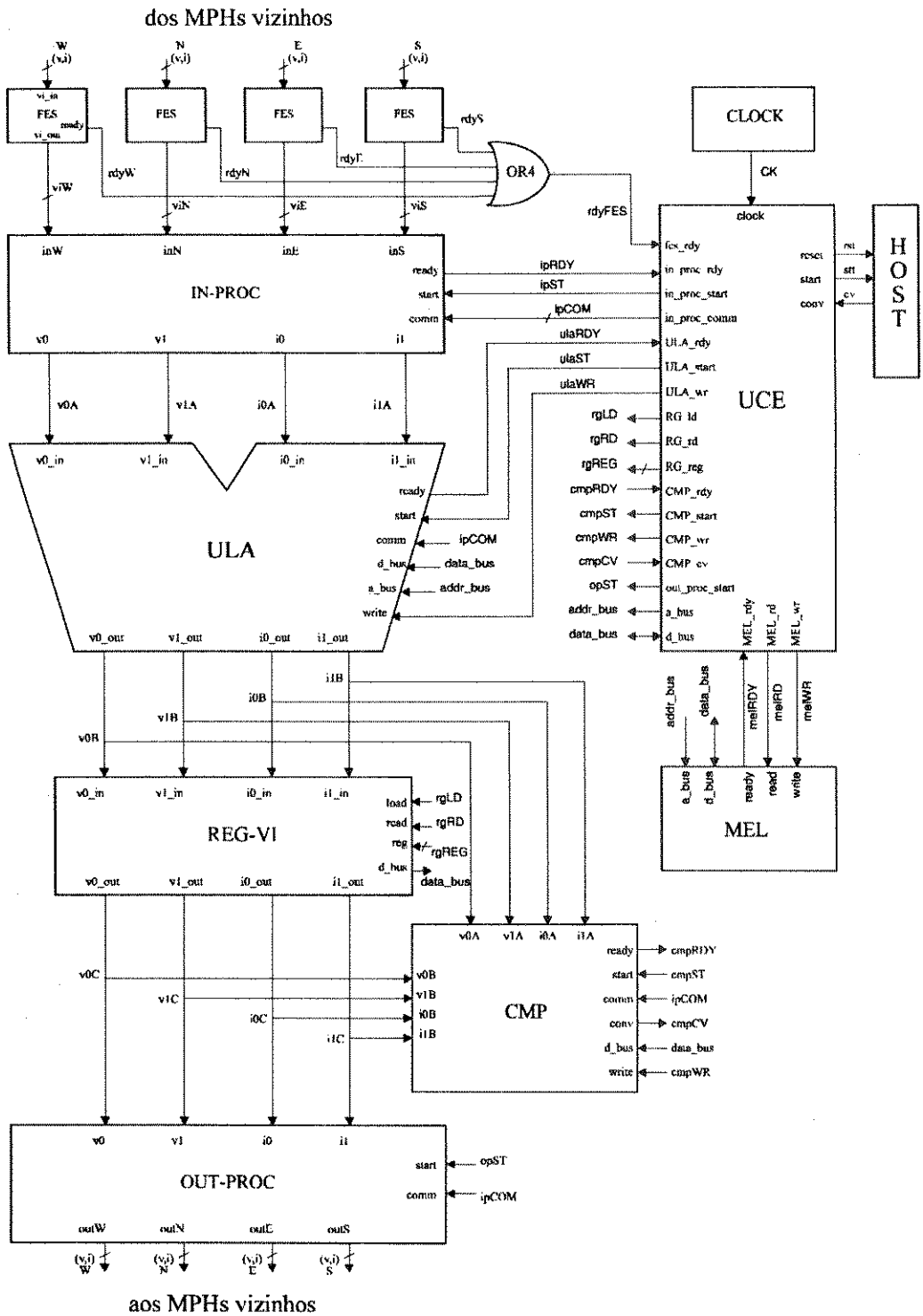


Fig. 4.20 Arquitetura RTL de um elemento de processamento de modelos (MPH).

```
USE WORK.dados.ALL;

ENTITY fes IS

    PORT (
          -- par tensão/corrente de entrada
          vi_in    : IN vc_base;
          -- par tensão/corrente de saída
          vi_out   : OUT vc_base;
          -- avisa da presença de um novo dado
          ready    : OUT BIT );

END fes;
```

Fig. 4.21 Descrição VHDL da interface de uma FES.

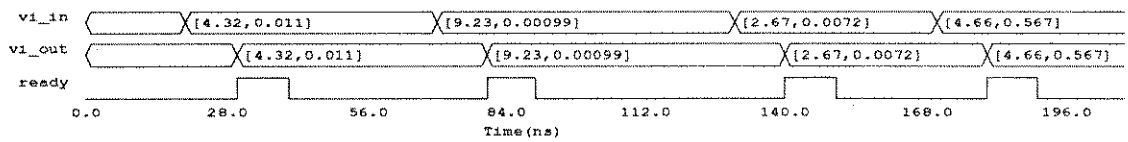


Fig. 4.22 Diagrama de tempos do bloco FES.

4.2.1.3 O Bloco Processador de Entrada (IN-PROC)

O bloco **IN-PROC** recebe os dados de tensão e corrente das FESs de entrada por meio dos sinais `viW`, `viN`, `viE`, `viS` cujas estruturas são do tipo `vc_base`. Inicia a sua operação quando recebe um '1' no sinal `ipST`. Processa os dados de tensão e corrente das entradas e obtém as tensões `v0`, `v1` e as correntes `i0`, `i1` correspondentes aos polos 0 e 1 baseado na topologia de cada polo do elemento simulado. Quando essa operação termina, o bloco IN-PROC comunica ao bloco UCE do fato por meio de um '1' no sinal `ipRDY`.

Os dados relativos à topologia do elemento que está sendo simulado são fornecidos pelo sinal `comm`, cuja estrutura é como se mostra na Fig. 4.23. Essa estrutura é formada por dados que detalham o seguinte: de qual polo do elemento uma FES recebe os dados; se o elemento simulado é uma fonte de alimentação; se algum polo do elemento está ligado a alguma fonte; qual o polo ligado à fonte; se algum polo está ligado ao terra; e qual o polo do elemento simulado pelo MPH vizinho, para poder determinar o sentido das correntes.

A descrição VHDL da interface do bloco IN-PROC mostra-se na Fig. 4.24 e a listagem da descrição da sua arquitetura pode ser vista no apêndice 2.

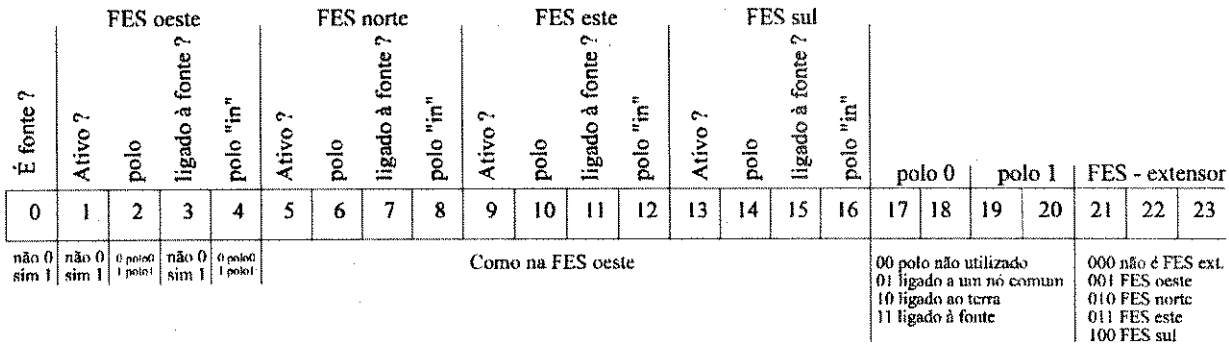


Fig. 4.23 Estrutura do sinal comm do bloco IN-PROC.

```
USE WORK.dados.ALL;

ENTITY in_proc IS

    PORT (
          -- pares tensão/corrente de entrada
          inW, inN, inE, inS  : vc_base;
          -- sinaliza à UCE quando a operaçao termina
          ready      : OUT BIT;
          -- inicia a operação do IN-PROC
          start      : IN BIT;
          -- topologia entorno do elemento simulado
          comm       : IN bit24;
          -- tensões resultantes nos polos 0 e 1
          v0, v1     : Out Real;
          -- corrente resultante no elemento
          i0, i1     : Out Real
          );

END in_proc;
```

Fig. 4.24 Descrição VHDL da interface do bloco IN-PROC.

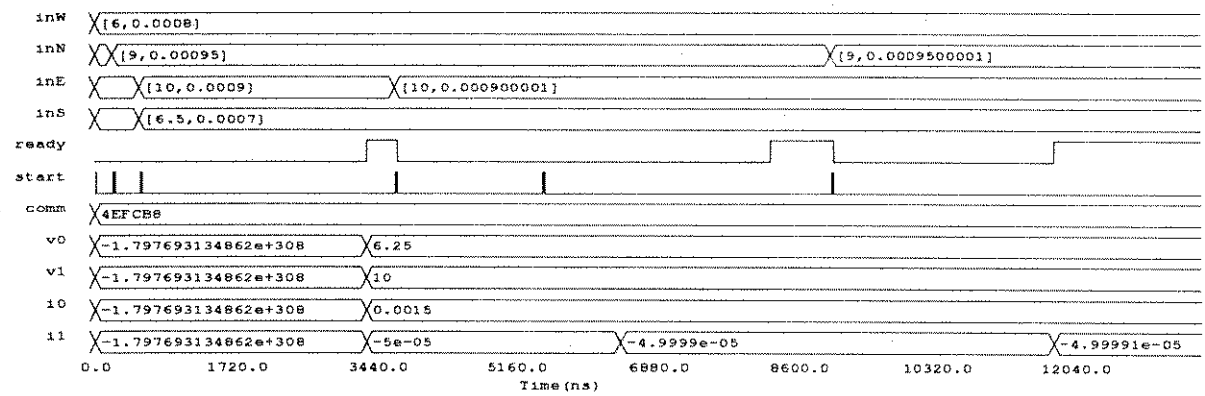


Fig. 4.25 Diagrama de tempos do bloco IN-PROC.

Observe-se nos resultados da simulação do bloco IN-PROC mostrados na Fig. 4.25 que os atrasos nos sinais de saída são consideráveis (aproximadamente 2,7 μs a partir do acionamento do sinal `start` até quando o sinal `ready` tem o nível lógico 1). Isto deve-se a que no bloco IN-PROC são realizadas operações de somas, subtração, multiplicação e divisão em ponto flutuante com os dados das correntes e tensões ingressantes.

4.2.1.4 A Unidade Lógica Aritmética (ULA)

A **ULA** é o bloco que fornece ao modelo do dispositivo que está sendo simulado as tensões e correntes dos polos que a ele chegam do bloco IN-PROC por meio dos sinais `v0A`, `v1A`, `i0A` e `i1A`, calculando assim novos valores de tensões e correntes que são atribuídos aos sinais de saída `v0_in`, `v1_in`, `i0_in` e `i1_in` da sua interface. O sinal `ulaST` inicia a operação da ULA, enquanto que o sinal `ulaRDY` avisa à UCE da sua finalização. Alguns dados relativos aos modelos são necessários na ULA. Esses dados são transferidos da MEL para uma memória interna à ULA, cuja escrita é realizada por meio do sinal `write`. Para tanto a ULA encontra-se ligada aos barramentos internos de dados (`data_bus`) e endereços (`addr_bus`).

Na Fig. 4.26 pode-se ver a descrição VHDL da interface do bloco ULA. Tanto as entradas quanto as saídas para os dados de tensão/corrente são de tipo `REAL`, ou seja são números em formato de ponto flutuante. Note-se a utilização do barramento de dados apenas como entrada. Usa-se apenas 3 linhas do barramento de endereços para o direcionamento das posições da memória interna. As estruturas de ambos os barramentos será discutida mais adiante. O sinal `comm` utilizado em IN-PROC novamente é utilizado neste bloco, apesar de que algumas informações deste sinal não serem necessárias a este bloco (vide listagen do apêndice 2), porém no próximo nível de abstração deverá estabelecer-se uma estrutura particular para seu uso na ULA.

O diagrama de tempos da Fig. 4.27 mostra os resultados da simulação do bloco ULA.

Os retardos que se observam nos sinais de saída das tensões/correntes correspondem ao tempo de execução dos modelos dos dispositivos (o exemplo corresponde ao de um resistor). Esses atrasos são calculados considerando os tempos de execução das operações aritméticas envolvidas no cálculo de cada modelo e que dependem do caminho utilizado no algoritmo. Como já foi mencionado utilizaram-se os tempos de execução que se mostram na tabela 2.2.

Os atrasos neste bloco são os mais significativos e representam o gargalo no tempo de processamento dos modelos. Por isto é muito importante a escolha da arquitetura no

```
USE WORK.dados.ALL;

ENTITY ula IS

     PORT (
          -- tensões de entrada
          v0_in, v1_in    : IN REAL;
          -- corrente de entrada
          i0_in, i1_in    : IN REAL;
          -- avisa à UCE do término da operação
          ready     : OUT BIT;
          -- começa a operação de cálculo
          start     : IN BIT;
          -- contém a topologia do entorno
          comm      : IN bit24;
          -- barramento de endereços
          d_bus     : IN real_data;
          -- barramento de dados
          a_bus     : IN INTEGER;
          -- sinal de escrita na memória interna
          write     : IN BIT;
          -- tensões de saída
          v0_out, v1_out  : OUT REAL;
          -- corrente de saída
          i0_out, i1_out  : OUT REAL
          );

END ula;
```

Fig. 4.26 Descrição VHDL da interface do bloco ULA.

seguinte nível de abstração para poder diminuir ao máximo os retardos mencionados.

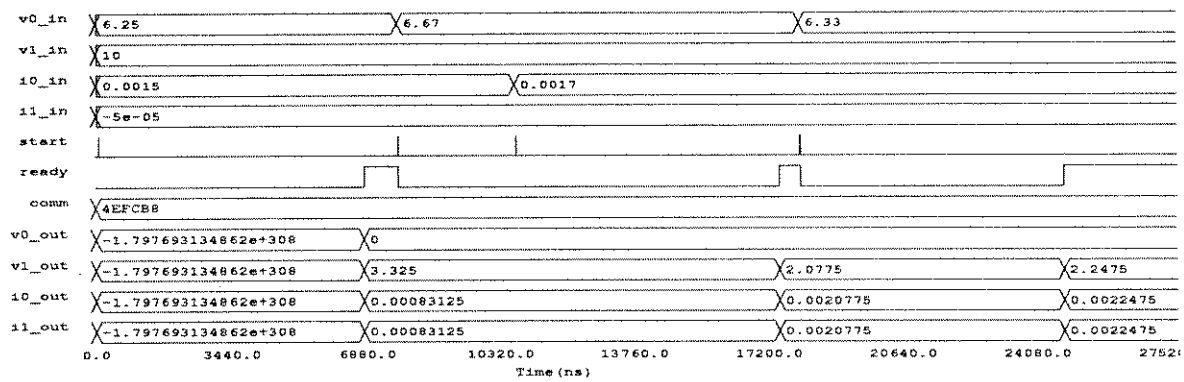


Fig. 4.27 Diagrama de tempos do bloco ULA.

4.1.2.5 O Bloco Registrador de Tensões e Correntes (REG-VI)

Este bloco tem por finalidade armazenar os dados das tensões e correntes, calculadas no bloco ULA, de cada polo do elemento num conjunto de registradores. Esses dados podem ser acessados sempre que o bloco UCE os precisar, tendo para tanto o bloco REG-VI ligação ao barramento de dados.

Os dados vindos do bloco ULA por meio dos sinais `v0B`, `v1B`, `i0B` e `i1B` são carregados nos registradores quando a UCE der um '1' no sinal `rgLD`, isto é, os sinais da interface `v0_out`, `v1_out`, `i0_out` e `i1_out` são atribuidos com esses dados de tensões e correntes. O sinal `read` é utilizado para a leitura pela UCE dos valores armazenados nos registradores, os quais são endereçados pelo sinal `reg` que conta na sua estrutura com 2 bits. Esses dados quando solicitados, são colocados no barramento de dados `data_bus` por meio do sinal `d_bus` da interface.

A arquitetura comportamental para este bloco é formada por dois processos. Um para o processo de armazenagem dos dados nos registradores e o outro para a leitura desses dados pela UCE. No apêndice 2 encontra-se a listagem completa das descrições VHDL para essa arquitetura. Na Fig. 4.28 apresenta-se a descrição VHDL da interface correspondente.

Observe-se do diagrama de tempos mostrado na Fig. 4.29 que os atrasos neste bloco são pequenos. De fato, são similares aos retardos que apresentam os registradores comuns, por exemplo da tecnologia CMOS.

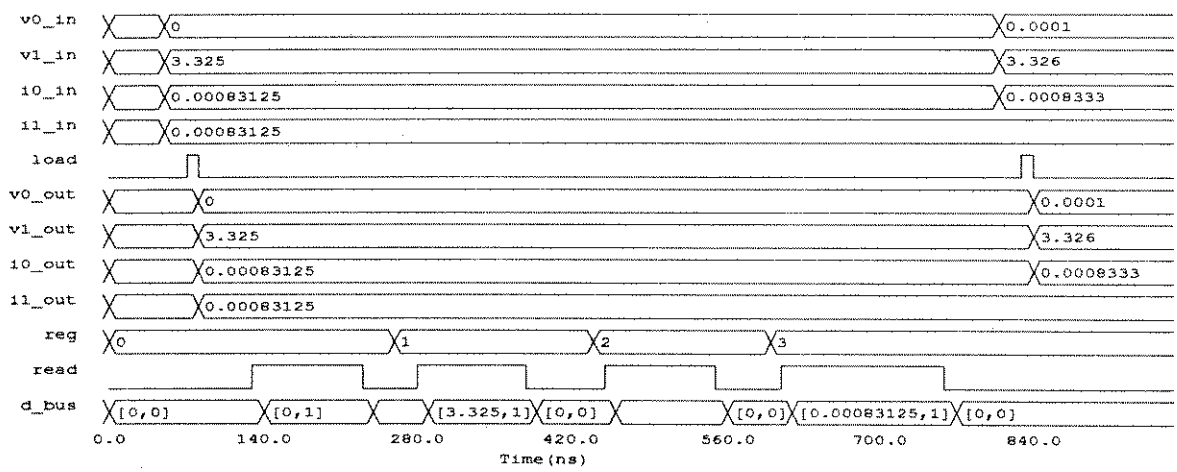
```
USE WORK.dados.ALL;

ENTITY reg_vi IS

    PORT (
        -- tensões de entrada
        v0_in, v1_in    : IN REAL;
        -- correntes de entrada
        i0_in, i1_in    : IN REAL;
        -- carrega os registradores
        load      : IN BIT;
        -- lê os registradores
        read      : IN BIT;
        -- endereça o registrador a ser lido
        reg       : IN bit2;
        -- ao barramento de dados
        d_bus     : OUT real_bus := NOT_DRIVEN;
        -- tensões de saída
        v0_out, v1_out  : OUT REAL;
        -- correntes de saída
        i0_out, i1_out  : OUT REAL
        );

END reg_vi;
```

Fig. 4.28 Descrição VHDL da interface do bloco REG-VI.



Fig, 4.29 Diagrama de tempos do bloco REG-VI.

4.1.2.6 O Bloco Comparador (CMP)

Os sinais v0B, v1B, i0B e i1B produzidos pelo bloco ULA, são comparados por este bloco aos sinais v0C, v1C, i0C e i1C, disponíveis nas saídas do bloco REG-VI, respectivamente. Desta maneira, tensões e correntes que correspondem às iterações anterior e atual são comparadas. Essa comparação é realizada em torno de um valor de erro determinado pelo usuário. Se a diferença entre os sinais de entrada ao comparador estão dentro da margem de erro, então é estabelecida a condição de convergência local e comunicada à UCE para a sua transmissão ao processador gerenciador.

O valor do erro é transferido da MEL ao bloco comparador e para tanto existe uma ligação apenas de entrada com o barramento de dados (vide descrição VHDL da interface do bloco CMP na Fig. 4.30). O sinal write escreve o valor referido num registrador interno (ERR) ao bloco comparador. Os sinais ready e start sincronizam o início e fim respectivamente, da operação do bloco CMP. O estado de convergência é passado à UCE por meio do sinal conv e cmpCV. A configuração da topologia do elemento é provida através do sinal comm.

O diagrama de tempos da Fig. 4.31 mostra os resultados da simulação do bloco comparador. Os sinais de saída, após o acionamento do sinal start com um '1', apresentam seus valores com um retardo de aproximadamente 7,5 µs. Este retardo aparentemente excessivo para um comparador, se deve ao fato de que são realizadas algumas operações aritméticas em ponto flutuante no interior do bloco CMP que naturalmente provocam atrasos significativos. As descrições VHDL detalhadas são apresentadas no apêndice 2.

```
USE WORK.dados.ALL;

ENTITY cmp IS

     PORT (
          -- tensões da iteração em curso
          v0A, v1A   : IN REAL;
          -- correntes da iteração em curso
          i0A, i1A   : IN REAL;
          -- tensões da iteração anterior
          v0B, v1B   : IN REAL;
          -- correntes da iteração anterior
          i0A, i1A   : IN REAL;
          -- avisa à UCE quando termina a operação
          ready      : OUT BIT;
          -- começa uma nova comparação
          start      : IN BIT;
          -- topologia em torno do elemento
          comm       : IN bit24;
          -- sinaliza o estado de convergência local
          conv       : OUT BIT;
          -- conetado ao barramento de dados
          d_bus      : IN real_data;
          -- escreve no registrador ERR
          write      : IN BIT );

END cmp;
```

Fig. 4.30 Descrição VHDL da interface do bloco CMP.

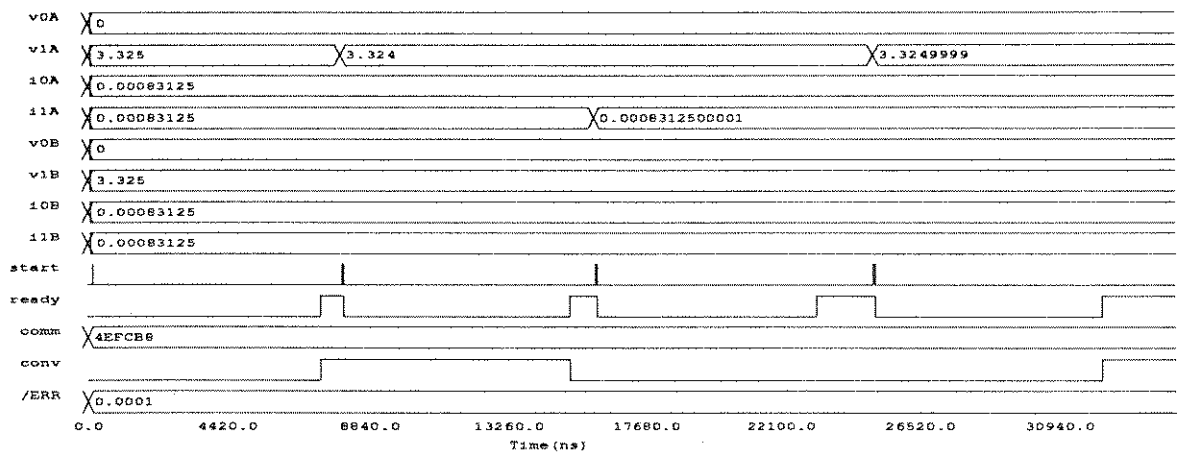


Fig. 4.31 Diagrama de tempos do bloco CMP.

4.2.1.7 O Bloco Processador de Saída (OUT-PROC)

O diagrama da Fig. 4.32 mostra a descrição VHDL da interface do bloco processador de saída (OUT-PROC). A função deste bloco consiste em converter os dados das tensões e correntes (de tipo `REAL`) provenientes do bloco REG-VI e recebidos nos sinais `v0`, `v1`, `i0` e `i1`, em pares tensão/corrente com a estrutura `vc_base`, que serão transmitidos aos canais de saída oeste, norte, este e sul do MPH. Para tanto, este bloco recebe um '1' no sinal `start` e dá início ao processo de conversão. Não é preciso notificar ao bloco UCE do término da conversão. Por outro lado é necessário saber quais FESs de saída estão sendo utilizadas, usando para tal caso o sinal `comm` da interface.

O processo de conversão consome tempos pequenos como pode ser observado nos resultados obtidos no diagrama de tempos da Fig. 4.33. Isto principalmente corresponde ao tempo de transferência dos dados como no caso do bloco REG-VI.

As descrições VHDL da arquitetura comportamental para este bloco se encontram nas listagens do apêndice 2.

```
USE WORK.dados.ALL;

ENTITY out_proc IS

     PORT (
          -- tensões dos polos 0 e 1
          v0, v1   : IN REAL;
          -- corrente no elemento
          i0, i1   : IN REAL;
          -- começa a operação de conversão
          start    : IN BIT;
          -- indica quais FESs estão sendo utilizadas
          comm     : IN bit24;
          -- canais de saída oeste, norte, este e sul
          outW, outN, outE, outS  : OUT vc_base
          );

END out_proc;
```

Fig. 4.32 Descrição VHDL da interface do bloco OUT-PROC.

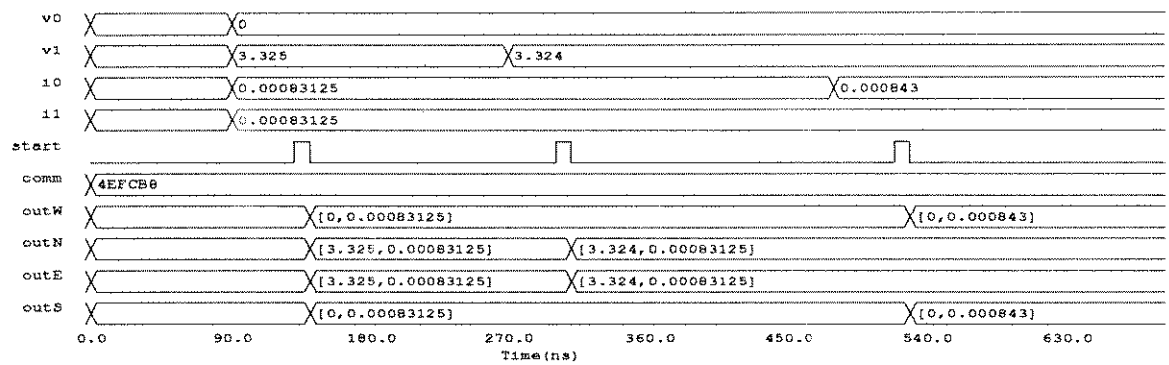


Fig. 4.33 Diagrama de tempos do bloco OUT-PROC.

4.2.1.8 A Memória de Escrita e Leitura (MEL)

Como foi mencionado no capítulo 2, a MEL armazena a configuração do MPH no que se refere à sua topologia e os parâmetros iniciais do dispositivo para a sua simulação. Na sua interface (vide Fig. 4.34) inclui sinais para leitura (`read`), escrita (`write`) e sinalização de disponibilidade (`ready`) de dados e também as ligações para os barramentos de dados e endereços.

Neste caso, a arquitetura é do tipo RTL, como pode ser visto na listagem do apêndice 2. Observe-se que o modelamento dessa arquitetura é a nível de fluxo de dados. Inicialmente a MEL coloca o barramento de dados em "alta impedância" atribuindo a este último o valor `NOT_DRIVEN` e aguarda por um comando de leitura ou escrita. Caso existir o comando de leitura a MEL coloca o dado do endereço indicado no barramento de endereços, no barramento de dados e sinaliza a disponibilidade do dado, aguardando nesse estado até que o sinal de leitura seja desativado. No caso do comando de escrita, a MEL sinaliza a disponibilidade e espera o sinal de escrita ser desativado para poder armazenar o dado no endereço indicado. O sinal `mem` declarado no interior da arquitetura da MEL é quem "memoriza" os dados armazenados.

Analizaremos a seguir as estruturas dos barramentos de endereços e de dados. O barramento de endereços foi definido simplesmente de tipo `INTEGER`. No entanto o barramento de dados conta com uma estrutura mais complexa e que é chamada de `real_bus`.

Há vários blocos que utilizam o barramento de dados, seja para escrita ou para leitura. No caso de leitura não há maiores problemas pois é utilizado como uma entrada. Já no caso de escrita é necessário um gerenciamento, pois apenas um bloco por vez pode escrever nele. O problema surge quando mais de um bloco quer acessá-lo, portanto é necessário gerar um indicador de uso (bandeira) do barramento. Em conseqüência a estrutura do barramento estará consitituida do dado propriamente dito e do indicador de uso. Em VHDL isto é implementado por meio das chamadas "funções de resolução". A estrutura é como segue:

```
CONSTANT DATA     : INTEGER := 0;
CONSTANT DRIVE    : INTEGER := 1;
TYPE real_data IS ARRAY (DATA TO DRIVE) OF REAL;
TYPE real_data_vec IS ARRAY (NATURAL RANGE <> ) OF real_data;
FUNCTION resolve_real_bus (vec : real_data_vec) RETURN real_data;
SUBTYPE real_bus IS resolve_real_bus real_data;
CONSTANT NOT_DRIVEN := (0.0, 0.0);
```

Cada bloco deve encarregar-se da desativação do indicador de uso, logo após o término do acesso ao barramento de dados, pois do contrário os demais blocos não poderão utilizá-lo, podendo-se gerar um erro na simulação.

O estado de "alta impedância" é simulado por meio do uso da constante `NOT_DRIVEN` cujo valor (`0.0, 0.0`) indica que o barramento de dados não está sendo utilizado.

O diagrama de tempos da Fig. 4.35 mostra os resultados da simulação do bloco MEL.

```
USE WORK.dados.ALL;

ENTITY mel IS

    PORT (
          -- barramento de dados
          d_bus    : INOUT real_bus BUS;
          -- barramento de endereços
          a_bus    : IN INTEGER;
          -- lê um dado da memória
          read     : IN BIT;
          -- escreve um dado na memória
          write    : IN BIT;
          -- avisa da disponibilidade do dado
          ready    : OUT BIT
          );

END mel;
```

Fig. 4.34 Descrição VHDL da interface da Memória de Escrita e Leitura (MEL).

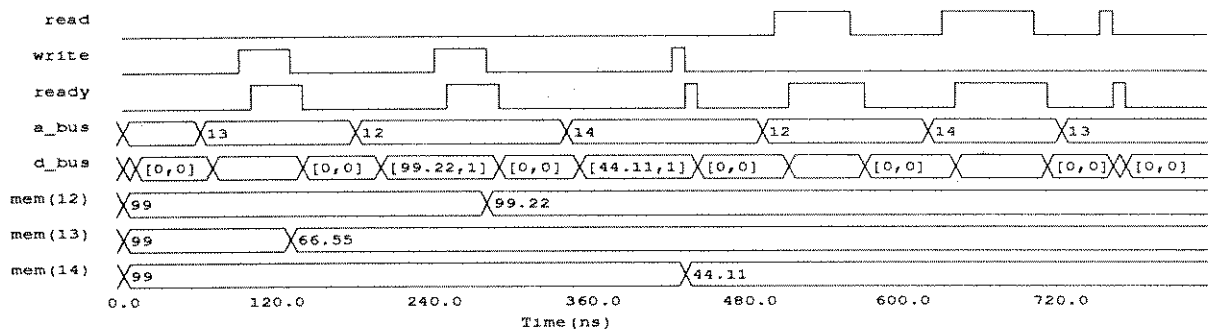


Fig. 4.35 Diagrama de tempos da Memória de Escrita e Leitura (MEL).

4.2.1.9 A Unidade de Controle Elementar (UCE)

Os blocos antes mencionados trabalham diretamente com o fluxo principal de dados das tensões e correntes correspondentes ao elemento simulado. No entanto, eles necessitam de um esquema de controle que providencie os sinais de sincronismo para o início e término da operação de cada bloco, assim como para a transferência de alguns dados entre a MEL e os blocos. Essa tarefa é realizada pela unidade de controle elementar (UCE), sob a temporização de um sinal de relógio gerado pelo bloco CLOCK (vide diagrama da Fig. 4.20) e de uma máquina de estados.

Neste nível de abstração a UCE é modelada considerando o fluxo de dados existente através dos blocos do MPH, portanto pode ser denominada de RTL.

Na Fig. 4.35 mostra-se a interface VHDL da UCE. Pode-se observar um grupo de sinais para a comunicação com o processador gerenciador (`conv`, `reset` e `start`); um sinal para receber o sinal de relógio (`clock`); outro para sinalizar a presença de dados nas FESs de entrada (`fes_rdy`); um grupo para o controle do bloco IN-PROC (`in_proc_rdy`, `in_proc_start` e `in_proc_comm`); outro grupo para o controle da ULA (`ULA_rdy`, `ULA_start` e `ULA_wr`); os sinais `RG_ld`, `RG_rd` e `RG_reg` para o controle do bloco REG-VI; um grupo de sinais para a operação do bloco comparador (`CMP_rdy`, `CMP_start`, `CMP_wr` e `CMP_cv`); o sinal `out_proc_start` para iniciar a operação do bloco OUT-PROC; um grupo de sinais para leitura e escrita da MEL (`MEL_rd`, `MEL_wr` e `MEL_rdy`); e, finalmente os sinais `a_bus` para o barramento de endereços e o sinal `d_bus` para o barramento de dados.

A descrição VHDL da arquitetura da UCE contém quatro processos. Três deles dedicam-se ao controle dos sinais `reset`, `start` e `fes_rdy`, implementando-se para esse fim três registradores chamados de `ffd_reset`, `ffd_start` e `ffd_fes_rdy`, respectivamente. O quarto processo chamado de `uce_proc` é o responsável pelo gerenciamento de todos os blocos do MPH. Esse processo consta de cinco procedimentos auxiliares quais sejam: o procedimento `read_MEL` usado para a leitura de dados da MEL; os procedimentos `MEL_to_ULA` e `Tx_to_ULA` utilizados para a transferência dos dados relativos aos modelos dos dispositivos da ULA à MEL; o procedimento `REGvi_to_MEL` utilizado para a transferência dos dados das tensões e correntes do bloco REG-VI à MEL; e o procedimento `send_comm` usado para enviar os dados da configuração do MPH, no formato da estrutura mostrada na Fig. 4.23, no sinal `in_proc_comm`.

A operação do processo `uce_proc` é baseada numa máquina de estados e o sinal de relógio `clock`. Os estados desta máquina são descritos nos parágrafos subseqüentes.

O primeiro estado dessa máquina de estados é `INIT`, que indica o estado inicial do MPH. Nele todos os blocos são inicializados; o tempo de simulação é inicializado com o primeiro instante; o sinal `in_proc_comm` é carregado com a topologia do elemento; os registradores da ULA são carregados com os dados relativos aos modelos; e o erro de convergência é escrito no bloco CMP. Chega-se a este estado quando o sinal `reset` for '1'.

O segundo estado chama-se de `NEXT_INSTANT`, o qual pode ser atingido mediante a presença de um '1' no sinal `reset` ou como conseqüência do estado `INIT`. Neste estado o sinal `conv` é desativado (atribui-se-lhe o valor '0') indicando o começo de uma iteração nova. Também o tempo de simulação é incrementado em um passo.

```
USE WORK.dados.ALL;

ENTITY uce IS

    PORT (
          -- Sinais para a comunicação com o gerenciador
          reset       : IN BIT;
          start       : IN BIT;
          conv        : OUT BIT;

          -- Sinal de relógio
          clock       : IN BIT;

          -- Indica a presença de um novo dado nas FESs
          fes_rdy     : IN BIT;

          -- Controle do bloco IN-PROC
          in_proc_rdy      : IN BIT; -- resultado pronto ?
          in_proc_start    : OUT BIT; -- inicia um novo cálculo
          in_proc_comm     : OUT bit24; -- configuração das FESs

          -- Controle da ULA
          ULA_rdy     : IN BIT; -- resultado pronto ?
          ULA_start   : OUT BIT; -- começa novo cálculo
          -- escreve na memória interna da ULA
          ULA_wr      : OUT BIT;

          -- Controle do bloco REG-VI
          RG_ld       : OUT BIT; -- carrega os registradores
          RG_rd       : OUT BIT; -- lê um registrador
          RG_reg      : OUT bit2; -- endereça um registrador

          -- Controle do comparador - CMP
          CMP_rdy     : IN BIT; -- comparação feita ?
          CMP_start   : OUT BIT; -- começa a comparação
          CMP_wr      : OUT BIT; -- escreve o erro no CMP
          -- indica o estado de convergência
          CMP_cv      : IN BIT;

          -- inicia a operação do bloco PROC-OUT
          out_proc_start   : OUT BIT;

          -- Controle da memória - MEL
          MEL_rd      : OUT BIT; -- leitura
          MEL_wr      : OUT BIT; -- escrita
          MEL_rdy     : IN BIT; -- disponibilidade do dado

          -- Barramento de endereços
          a_bus       : OUT INteger;

          -- Barramento de dados
          d_bus       : INOUT real_bus Bus := NOT_DRIVEN);

END uce;
```

Fig. 4.36 Descrição VHDL da Interface da Unidade Elementar de Controle (UCE).

Chega-se ao terceiro estado, chamado de READ_IN_PROC, ao finalizar o estado NEXT_INSTANT. Neste estado inicia-se a operação do bloco IN-PROC, dando-se um pulso no sinal in_proc_start, e aguarda-se até a sua finalização. Se no meio desse tempo os sinais reset, start ou fes_rdy apresentam um nível '1', volta-se ao estado INIT. Caso contrário continua-se com o seguinte estado (CALCULATE).

No estado CALCULATE dá-se início à operação da ULA por meio de um pulso no sinal ULA_start. Como no estado anterior, enquanto se aguarda pelo término da sua operação verifica-se a presença de pulsos nos sinais reset, start ou fes_rdy. Caso existam volta-se ao estado inicial.

Ao quinto estado, chamado de COMPARE, chega-se da decorrência do estado CALCULATE. Neste estado aciona-se o bloco comparador para a sua operação com os valores de tensões e correntes das interações presente e anterior. Pode-se voltar ao estado inicial se houver pulsos nos sinais reset, start ou fes_rdy. Caso não existam, o bloco comparador comunica à UCE o estado de convergência por meio do sinal cmpCV e passa-se ao estado STORE_AND_WRITE_FES.

As operações de escrita das tensões e correntes no bloco REG-VI, assim como o início da operação do bloco OUT-PROC, realizam-se no estado STORE_AND_WRITE_FES. Neste caso também verifica-se a presença de pulsos nos sinais reset, start e fes_rdy.

O último estado da máquina de estados chama-se de STANDBY, que indica a condição de espera do MPH por novos dados dos MPHs vizinhos.

Os resultados das simulações da UCE mostram-se nos diagramas de tempo das Figs. 4.37, 4.38 e 4,39. No primeiro diagrama observa-se o comportamento dos sinais de controle que foram mencionados na descrição do estado INIT. Mostram-se também os barramentos de dados e de endereços, algumas posições da memória interna da ULA e o registrador para o erro do bloco comparador.

No segundo diagrama de tempos mostra-se o fluxo das tensões e correntes começando nos blocos FESs de entrada, passando pelos blocos IN-PROC, ULA, REG-VI e culminando no bloco OUT-PROC. Notem-se os atrasos relativos a cada bloco a partir da ativação dos sinais de início ipST, ulaST, cmpST, rgLD e opST.

O terceiro diagrama de tempos mostra a resposta de um MPH. Tendo-se nas FESs de entrada alguns dados de tensões e correntes, obtem-se nas saídas do bloco OUT-PROC os novos valores calculados. Mostram-se também os sinais de controle para indicar o início e o término de todos os blocos.

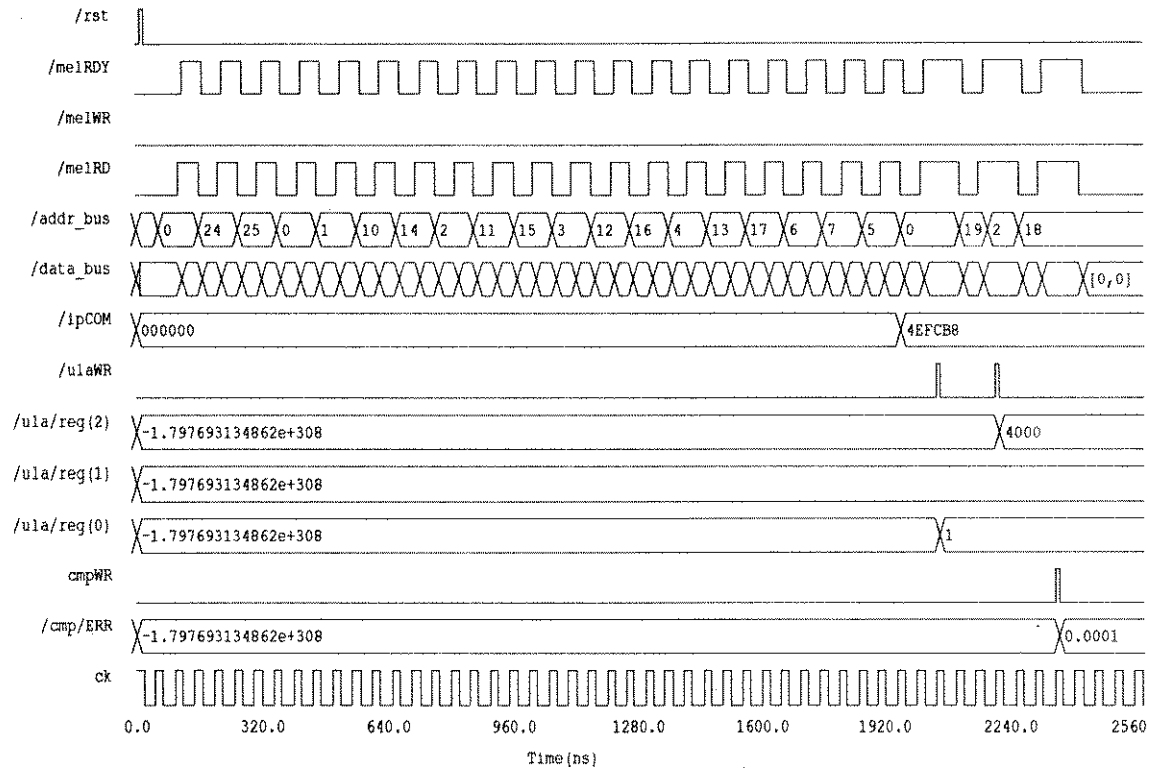


Fig. 4.37 Diagrama de tempos da UCE. Estado INIT.

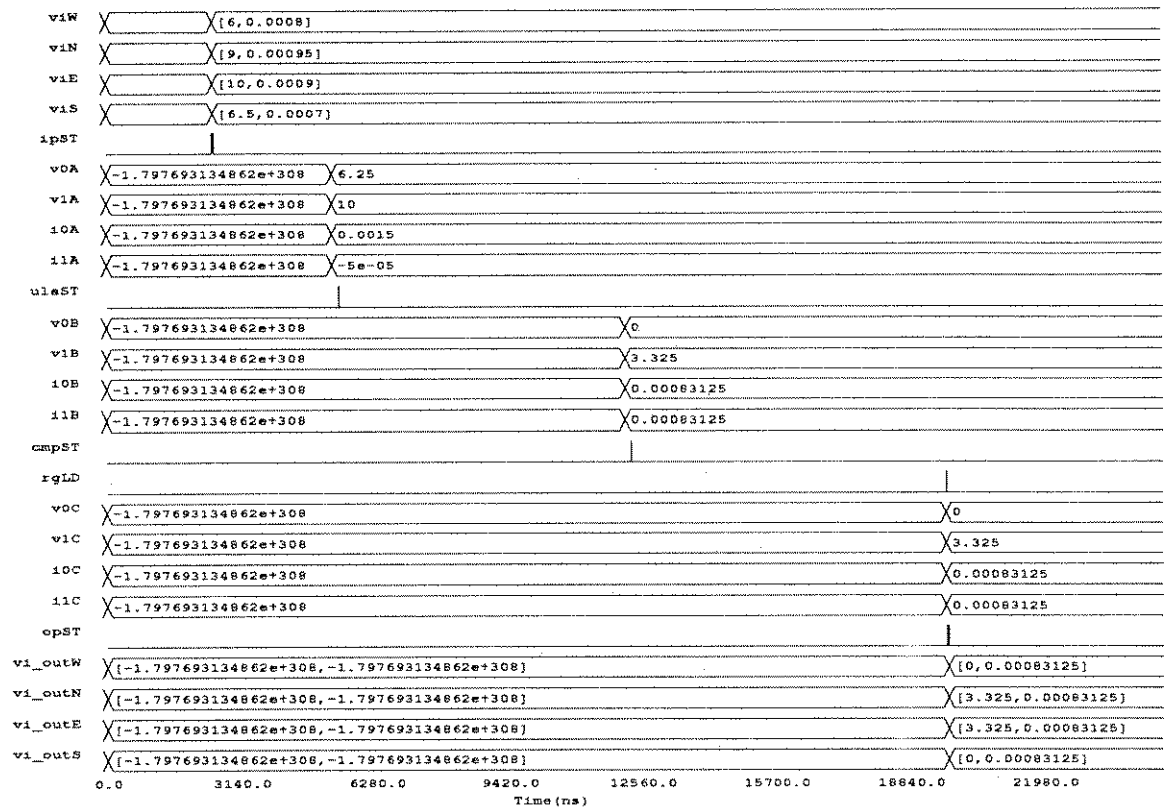


Fig. 4.38 Diagrama de tempos da UCE. Fluxo das tensões e correntes através dos blocos.

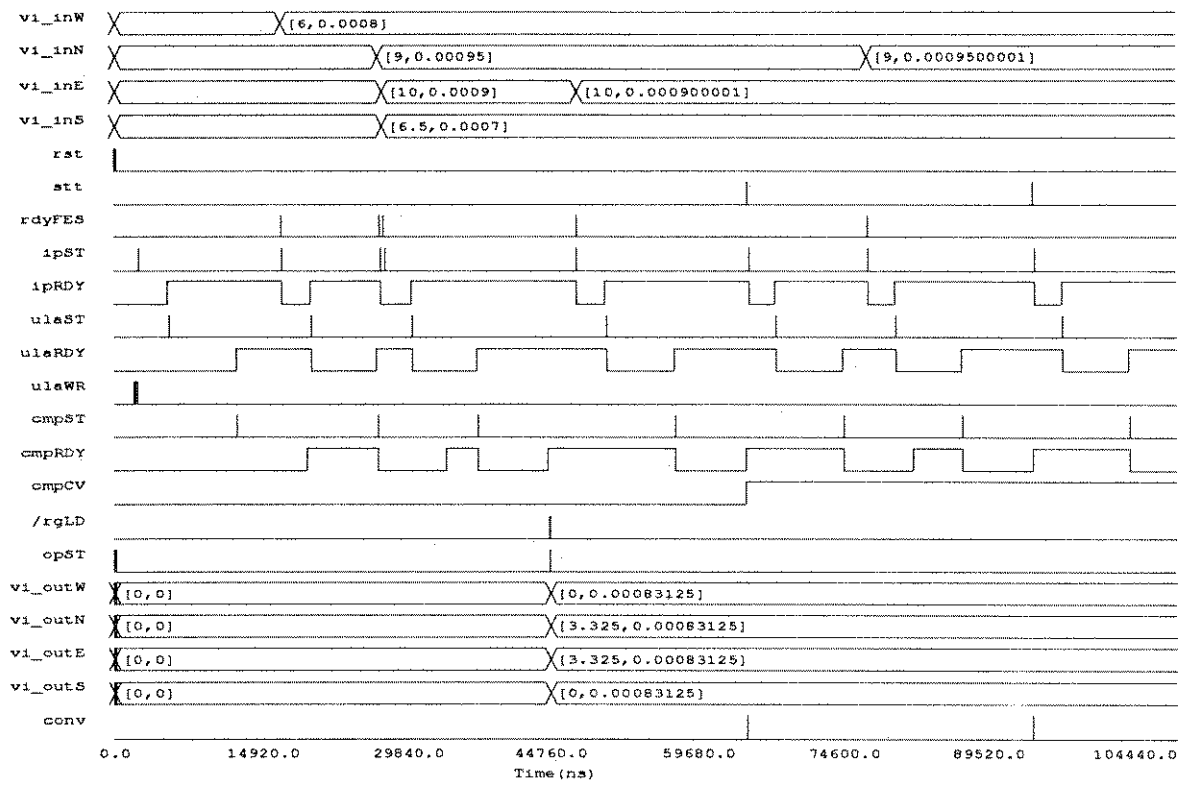


Fig. 4.39 Diagrama de tempos da UCE. Estímulos e saídas num MPH.

4.2.2 Os Elementos de Interconexão (chaves)

O interesse no primeiro nível de abstração pelos elementos de interconexão foi com o seu funcionamento e foram realizadas para tanto descrições VHDL comportamentais. Neste nível propõe-se uma estrutura que implementa as chaves com blocos multiplexadores e demultiplexadores. No diagrama da Fig. 4.40 apresenta-se dita estrutura.

De forma similar aos MPHs o modelamento neste nível preocupa-se mais com o fluxo de tensões e correntes. A configuração das chaves é realizada na fase da simulação mediante o acesso direto ao sinal `reg_config`.

A estrutura proposta para uma chave é composta por 2 blocos multiplexadores e 2 blocos demultiplexadores. Os multiplexadores selecionam de quatro canais um e os demultiplexadores realizam uma operação contrária. Um par multiplexador/demultiplexador é utilizado para a transferência em um sentido, `muxA/dmxA` no diagrama da Fig. 4.40 (de oeste para o leste, por exemplo), e outro par para a transferência no sentido contrário, `muxB/dmxB` (de leste para o oeste). Note-se que os blocos `muxA` e `dmxB` recebem as mesmas linhas de seleção e da mesma maneira `muxB` e `dmxA`.

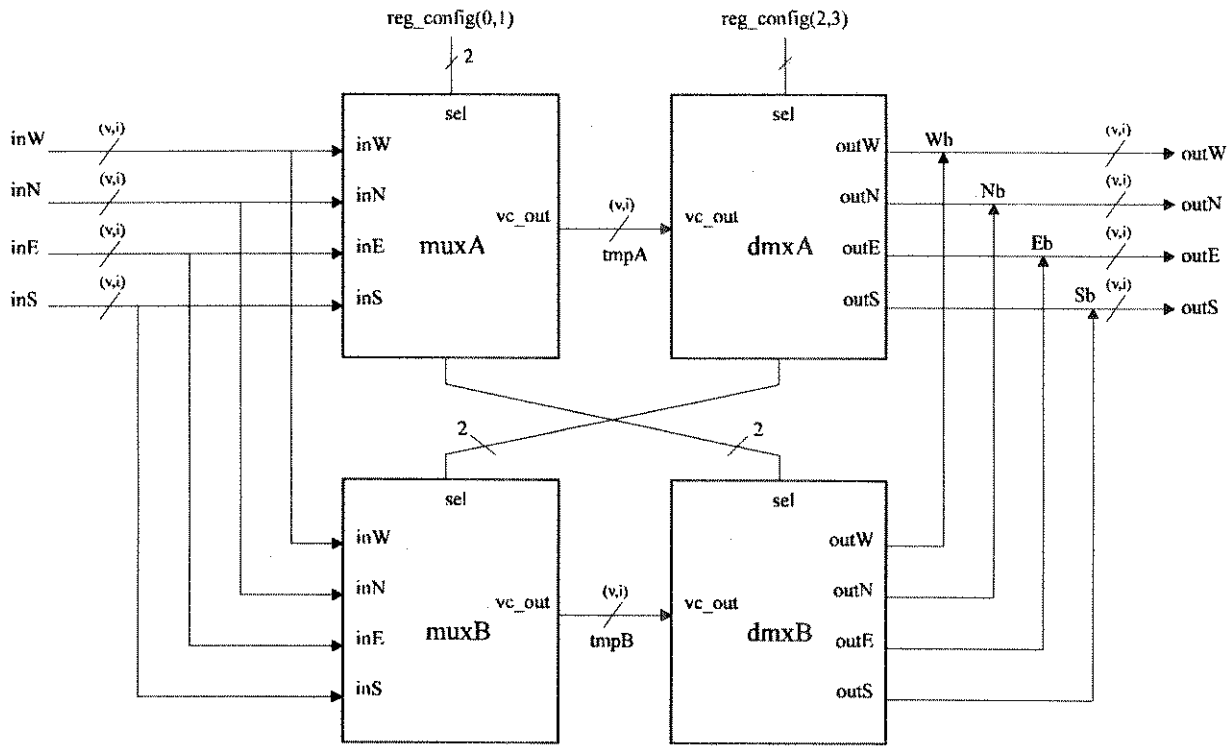


Fig. 4.40 Estrutura proposta para o elemento de interconexão.

O registrador `reg_config` é utilizado para selecionar os canais para a troca de dados. É importante mencionar que as parcelas `reg_config(0,1)` e `reg_config(2,3)` nunca devem conter a mesma configuração, considerando que a troca de dados acontece entre dois canais diferentes (oeste, norte, este ou sul). Na tabela 4.1 mostram-se os valores possíveis do registrador `reg_config`.

Canal 1	Canal 2	Transferência entre os canais
00	00	Não há transferência
00	01	oeste e norte
00	10	oeste e este
00	11	oeste e sul
01	10	norte e este
01	11	norte e sul
10	11	este e sul

Tabela 1. Valores para a configuração do registradores `reg_config` das chaves.

Considere-se o exemplo de troca de dados entre os canais oeste e sul. A parcela `reg_config(0,1)`, que configura os blocos `muxA` e `dmxB`, terá o valor '00'. Entretando, a parcela `reg_config(2,3)`, que configura `muxB` e `dmxA`, receberá o valor '11'. Portanto, o registrador `reg_config` terá o valor '0011'.

4.2.2.1 O Multiplexador de 4 Canais a 1

Na Fig. 4.41 observa-se a descrição VHDL da interface para este bloco. Os canais de entrada oeste, norte, este e sul são ligados aos sinais `inW`, `inN`, `inE` e `inS` respectivamente e são definidos de tipo `vc_base`. O sinal `sel` conformado por 2 bits seleciona um canal de entrada, o qual é transferido ao canal de saída ligado ao sinal `vc_out`.

```
USE WORK.dados.ALL;

ENTITY mux4to1 IS

    PORT(
         -- Entradas oeste, norte, este e sul
         inW, inN, inE, inS   : IN vc_base;
         -- Saida do multiplexador
         vc_out   : OUT vc_base;
         -- Seleciona um canal de entrada
         sel      : IN bit2 );

END mux4to1;
```

Fig. 4.41 Descrição VHDL da interface de um multiplexador de 4 canais a 1.

O diagrama de tempos para o bloco multiplexador mostra-se na Fig. 4.42. O atraso considerado para este bloco é de 10 ns. Esse valor foi utilizado considerando os retardos típicos nas portas CMOS.

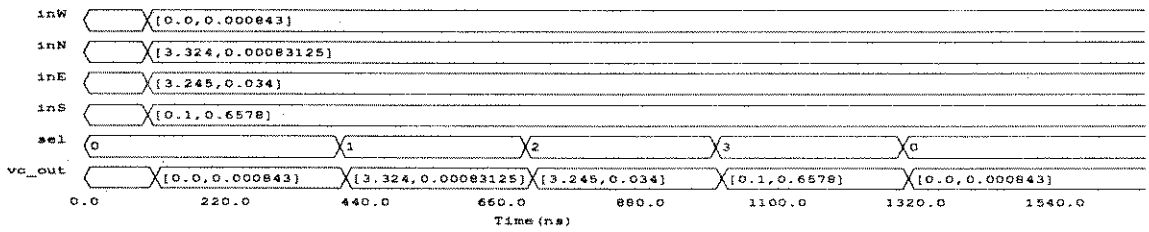


Fig. 4.42 Diagrama de tempos para o bloco multiplexador de 4 canais a 1.

4.2.2.2 O Demultiplexador de 1 a 4 Canais.

A função deste bloco é a atribuição do sinal `vc_in` de tipo `vc_base` ao canal de saída (`outW`, `outN`, `outE` ou `outS`), selecionado pelo sinal `sel`. Na Fig. 4.43 observa-se a descrição VHDL da interface correspondente a este bloco e os resultados da simulação no diagrama de tempos da Fig. 4.44.

Note-se no diagrama da Fig. 4.40 que as saídas dos blocos `dmxA` e `dmxB` compartilham os canais `Wb`, `Nb`, `Eb` e `Sb`. De forma similar que nos blocos que acessam o barramento de dados num MPH (vide Fig. 4.20), estes canais devem ser "resolvidos". Para tanto utiliza-se a função de resolução `resolve_vc_bus` mostrada nas listagens do apêndice 2.

```
USE WORK.dados.ALL;

ENTITY dmx1to4 IS

    PORT(
        -- Canal de entrada
        vc_in  : IN vc_base;
        -- Canais de saida oeste, norte, este e sul
        outW, outN, outE, outS : OUT vc_bus := NODRIVEN;
        -- Seleciona o canal de saida
        sel    : IN Bit2 );

END dmx1to4;
```

Fig. 4.43 Descrição VHDL da interface para o bloco demultiplexador de 1 a 4 canais.

Finalmente, os resultados da simulação de uma chave como bloco mostram-se na Fig. 4.45. Nesse diagrama foram considerados os sinais de entrada `inW`, `inN`, `inE` e `inS`, os sinais de saída `outW`, `outN`, `ouE` e `outS`, assim como as duas parcelas do registrador de configuração `reg_config`.

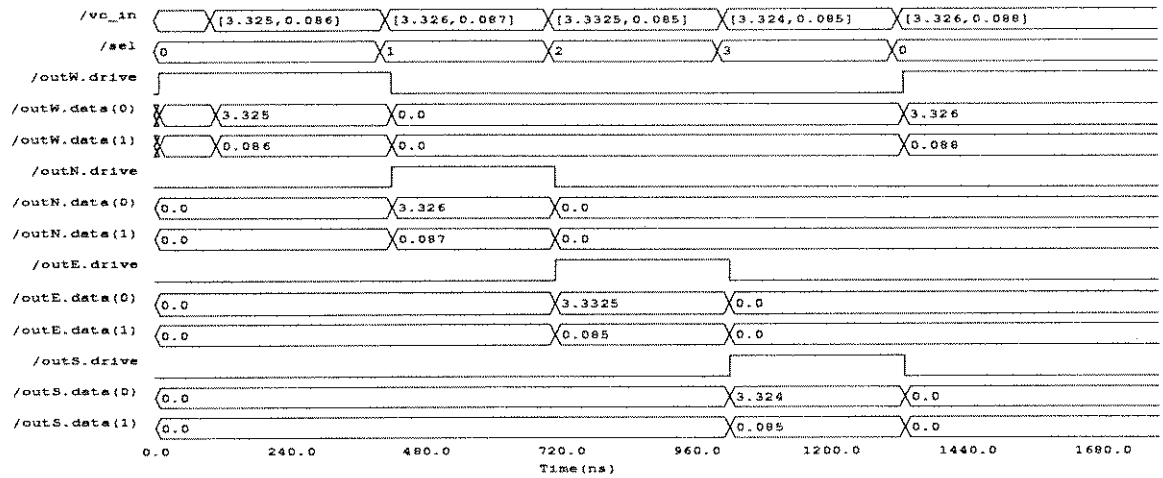


Fig. 4.44 Diagrama de tempos do bloco demultiplexador de 1 a 4 canais.

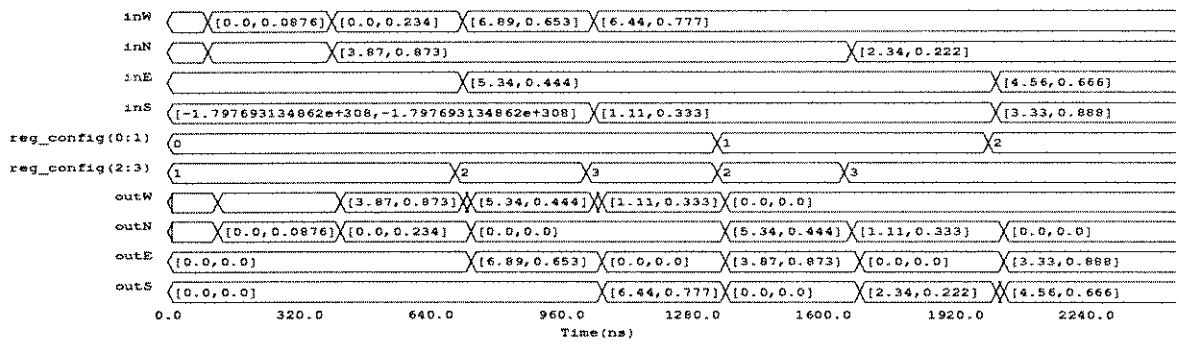


Fig. 4.45 Diagrama de tempos de uma chave quadrada.

4.3 Tempos de Execução dos Modelos dos Dispositivos

Com os resultados das simulações realizadas anteriormente podemos construir uma tabela comparativa dos tempos de execução dos modelos dos dispositivos dos níveis de modelamento (vide Tabela 4.2). Deve levar-se em conta que no nível de modelamento comportamental foram utilizados os tempos de simulação obtidos em [Mar92]. Entretanto, no nível RT, atribuiram-se às operações aritméticas em ponto flutuante, empregadas nos blocos desse nível, os tempos equivalentes das operações do Transputer T800-20.

Dispositivo	**1ro. Nível** (ciclos)	**2do. Nível** (ciclos)
Fonte de tensão independente	707	351,2
Fonte de tensão quadrada	879	300,2
Diodo	2555	1045,2
Resistor	919	347,4
Capacitor	1020	não implementado

(1 ciclo = 50 nS)

Tabela 4.2 Tempos de execução dos modelos dos dispositivos para ambos os níveis de modelamento.

Conclusão

Neste trabalho utilizou-se a linguagem de descrição de hardware VHDL para a representação das características principais do sistema de simulação elétrica de circuitos ABACUS conseguindo-se a verificação do seu funcionamento por meio da simulação.

A ferramenta de modelamento mostrou-se adequada, dadas as facilidades oferecidas para a representação de estruturas complexas de sinais por estruturas simples e fáceis de manusear. Especificamente para a representação dos canais de comunicação dos elementos de processamento de modelos e dos barramentos da rede de interconexão realizou-se por meio de simples números, ao invés de utilizar barramentos em formato binário com dimensões de palavra explicitadas.

O modelamento do ABACUS, em dois níveis de abstração, permitiu descrever o sistema com diferentes níveis de detalhamento. No primeiro nível pode verificar-se a funcionalidade do sistema, enquanto que no segundo nível a arquitetura dos MPHs e das chaves tornou-se visível. Os tempos de execução dos modelos dos dispositivos foram refinados no segundo nível de abstração, não obstante, dependem ainda fortemente da arquitetura de cada um dos blocos modelados nesse nível.

O tempo médio de travessia de um dado pela rede de interconexão que liga dois MPHs, é uma parcela pequena do tempo médio de processamento dos modelos dos dispositivos. Esse resultado leva a pensar que a dimensão dos canais de comunicação entre os MPHs e a rede de interconexão pode, no melhor dos casos, consistir apenas de uma linha por onde os dados seriam transmitidos serialmente. Porém, é necessário uma análise mais rigorosa da relação *tempo médio de travessia/tempo médio de execução do modelo* para melhor avaliar a sua influência na convergência do arranjo. Portanto, uma formalização matemática do sistema ABACUS seria conveniente.

Propõe-se para futuros trabalhos a procura por modelos de dispositivos que se ajustem melhor à metodologia de simulação de circuitos empregada no sistema ABACUS, assim como a procura de outros algoritmos de processamento para tais modelos.

Propõe-se também o estudo de algoritmos para o mapeamento e o roteamento do circuito no arranjo, ou a adequação de algoritmos existentes para tal fim ao sistema ABACUS.

APÊNDICES

1. LISTAGENS DAS DESCRIÇÕES VHDL DO PRIMEIRO NÍVEL DE ABSTRAÇÃO

```
-- ***************************************************************
-- Descrição comportamental dos Elementos de
-- Processamento de Modelos
-- MPH
-- ***************************************************************
Use std.math.ALL;
Use Work.aba_dados.ALL;

Entity mph Is

  Port(
    -- FES de entrada Oeste, Norte, ...
    inW,inN,inE,inS : In vc_base;
    -- FES de Saída Oeste, Norte, ...
    outW,outN,outE,outS : Out vc_base;
    -- Inicia a simulação do instante em curso
    start   : In Bit;
    -- Volta ao instante inicial da simulação
    reset   : In Bit;
    -- Indica o estado da convergência local
    conv    : Out Bit );

  -- Definição da memória de escrita e leitura
  Signal MEL : real_vector(0 To MEMSIZE) :=
            (Others=>NONE);

  -- Definição dos registradores
  -- Tensões e correntes no dispositivo
  Signal RegVI : real_vector(0 To RegsNUM)
                  := (Others=>0.0);
  -- Instante de simulação em curso
  Signal Tx   : Real := 0.0;
  -- Instante de transição do capacitor
  Signal Trans  : Real := 0.0;
  -- Constante de carga do capacitor
  Signal Tau  : Real := 0.0;
  -- Tensão de Steady State do capacitor
  Signal Vss  : Real := 0.0;
  -- Tensão Steady State inicial do capacitor
  Signal Vss0   : Real := 0.0;
  -- Carga armazenada no capacitor
  Signal Vcharge  : Real := 0.0;
  -- Corrente de curto circuito do capacitor
  Signal Izero    : Real := 0.0;

End mph;

Architecture mph_arq Of mph Is

Begin
  -- ***************************************************************
  -- Este processo verifica se há mudanças nos sinais start
  -- ou reset. Se o reset apresenta uma borda de subida o
  -- sinal Tx recebe o dado do tempo inicial da simulação.
  -- Se o start apresenta uma borda de subida o sinal
  -- Tx é incrementado em um passo de simulação.
  -- ***************************************************************
  start_cont : Process (start, reset)

  Variable tt : Real := 0.0;
  Begin

    If (reset'Event AND reset='1') OR
      (start'Event AND start='1') Then
      If reset'Event Then
        tt := MEL(T_INIT);
      Else
        tt := Tx + MEL(T_STEP);
        Vss0 <= Vss;
      End If;
      Tx <= tt;
    End If;

  End Process start_cont;

  -- ***************************************************************
  -- Este processo descreve o funcionamento de um MPH.
  -- Ele é acionado quando haja alguma mudança nas
  -- FES de entrada Oeste, Norte, ... ou no sinal Tx.
  -- ***************************************************************
  mph_behav : Process (inW, inN, inE, inS, Tx)

    -- Continua a definição dos registradores

    -- Corrente no dispositivo
    Variable i    : real_vector(POLE0 To POLE3);
    -- Tensões nos polos
    Variable v    : real_vector(POLE0 To POLE3);
    -- Vetores auxiliares para o cálculo dos desvios das
    -- correntes e tensões e determinação da convergência
    Variable di   : real_vector(POLE0 To POLE3);
    Variable dv     : real_vector(POLE0 To POLE3);
    Variable ei   : real_vector(POLE0 To POLE3);
    Variable ev     : real_vector(POLE0 To POLE3);
    Variable ct   : int_vector(POLE0 To POLE3);
    -- Vetor aux. para às FESs de saída
    Variable fes_out    : fes_base(fesW To fesS)
                    := (Others=>(0.0,0.0));
    -- Tempo utilizado para simular o dispositivo
    Variable fes_delay : time := 0 ns;
    -- Tensão entre os polos 1 e 0
    Variable v10  : Real;
    -- Corrente no elemento
    Variable i10    : Real;
    -- Tensão de threshold do diodo
    Variable vt   : Real;
```

```
    -- Tensão de steady state do capacitor
    Variable vf : Real;
    -- Carga armazenada no capacitor
    Variable vacum : Real;

    -- Definição das bandeiras de estado
    -- Indica se o dispositivo é uma fonte
    Variable supply : Boolean;
    -- Indica o estado de convergência local
    Variable cv   : Boolean := True;

    -- Definição das variáveis auxiliares
    Variable pW, pN, pE, pS, pEXT : Integer;   -- polos

  Begin -- Process mph_behav

    -- A MPH esta simulando algum dispositivo ?
    If MEL(ELEM)/=NONE Then

      -- Inicialização dos registradores
      i(POLE0 To POLE3) := (0.0,0.0,0.0,0.0);
      v(POLE0 To POLE3) := (0.0,0.0,0.0,0.0);
      ct(POLE0 To POLE3) := (0,0,0,0);

      -- O dispositivo é uma fonte ?
      supply := (MEL(ELEM)=VOL) Or (MEL(ELEM)=SQV);

      -- Converte a inteiro a informação dos polos
      pW := Integer(MEL(fesW_POLE));
      pN := Integer(MEL(fesN_POLE));
      pE := Integer(MEL(fesE_POLE));
      pS := Integer(MEL(fesS_POLE));

      -- Leitura da FES Oeste. Verifica se a FES esta
      -- conetada e se esta sendo utilizada como canal no
      -- modo de MPH extensor. Se a FES estiver
      -- conetada a uma fonte, no polo pW predomina
      -- a tensão da fonte. A corrente é somada
      -- ou substraida, dependendo da conetividade.
      If MEL(FES_EXT)/=Real(fesW)
        And MEL(fesW_POLE)/=NONE Then
        If MEL(PWD_fesW)=POWERED Then
          v(pW) := inW(VOLT);
        Else
          v(pW) := v(pW) + inW(VOLT);
          ct(pW) := ct(pW) + 1;
        End If;
        If supply Or MEL(PWD_fesW)=POWERED Then
          If MEL(fesW_POLE)=MEL(INPOLE_fesW) Then
            i(pW) := i(pW) + inW(CURR);
          Else
            i(pW) := i(pW) - inW(CURR);
          End If;
        Else
          If MEL(fesW_POLE)=MEL(INPOLE_fesW) Then
            i(pW) := i(pW) - inW(CURR);
          Else
            i(pW) := i(pW) + inW(CURR);
          End If;
        End If;

      -- Leitura da FES Norte. O funcionamento é similar
      -- ao da FES Oeste.
      If MEL(FES_EXT)/=Real(fesN)
        And MEL(fesN_POLE)/=NONE Then
        If MEL(PWD_fesN)=POWERED Then
          v(pN) := inN(VOLT);
        ElsIf MEL(PWD_POLE0+pN)/=POWERED Then
          v(pN) := v(pN) + inN(VOLT);
          ct(pN) := ct(pN) + 1;
        End If;
        If supply Or MEL(PWD_fesN)=POWERED Then
          If MEL(fesN_POLE)=MEL(INPOLE_fesN) Then
            i(pN) := i(pN) + inN(CURR);
          Else
            i(pN) := i(pN) - inN(CURR);
          End If;
        Else
          If MEL(fesN_POLE)=MEL(INPOLE_fesN) Then
            i(pN) := i(pN) - inN(CURR);
          Else
            i(pN) := i(pN) + inN(CURR);
          End If;
        End If;
      End If;

      -- Leitura da FES Este. O funcionamento é similar
      -- ao da FES Oeste.
      If MEL(FES_EXT)/=Real(fesE)
        And MEL(fesE_POLE)/=NONE Then
        If MEL(PWD_fesE)=POWERED Then
          v(pE) := inE(VOLT);
        ElsIf MEL(PWD_POLE0+pE)/=POWERED Then
          v(pE) := v(pE) + inE(VOLT);
          ct(pE) := ct(pE) + 1;
        End If;
        If supply Or MEL(PWD_fesE)=POWERED Then
          If MEL(fesE_POLE)=MEL(INPOLE_fesE) Then
            i(pE) := i(pE) + inE(CURR);
          Else
            i(pE) := i(pE) - inE(CURR);
          End If;
        Else
          If MEL(fesE_POLE)=MEL(INPOLE_fesE) Then
            i(pE) := i(pE) - inE(CURR);
          Else
            i(pE) := i(pE) + inE(CURR);
          End If;
        End If;
      End If;

      -- Leitura da FES Sul. O funcionamento é similar
      -- ao da FES Oeste.
      If MEL(FES_EXT)/=Real(fesS)
        And MEL(fesS_POLE)/=NONE Then
        If MEL(PWD_fesS)=POWERED Then
          v(pS) := inS(VOLT);
        ElsIf MEL(PWD_POLE0+pS)/=POWERED Then
          v(pS) := v(pS) + inS(VOLT);
```

```
        ct(pS) := ct(pS) + 1;
      End If;
      If supply Or MEL(PWD_fesS)=POWERED Then
        If MEL(fesS_POLE)=MEL(INPOLE_fesS) Then
          i(pS) := i(pS) + inS(CURR);
        Else
          i(pS) := i(pS) - inS(CURR);
        End If;
      Else
        If MEL(fesS_POLE)=MEL(INPOLE_fesS) Then
          i(pS) := i(pS) - inS(CURR);
        Else
          i(pS) := i(pS) + inS(CURR);
        End If;
      End If;
    End If;

    -- Calcula a media das tensões nos polos
    av1 : For p In POLE0 To POLE3 Loop
      If ct(p)/=0 Then
        v(p) := v(p)/Real(ct(p));
      End If;
    End Loop av1;

    -- *** MPH Extensor ***
    If MEL(ELEM)=EXT Then

      pEXT := Integer(MEL(FES_EXT));
      If MEL(FES_EXT)=Real(fesW) Then
        fes_out(fesN) := inW;
        fes_out(fesE) := inW;
        fes_out(fesS) := inW;
        fes_out(pEXT)(VOLT) := v(pW);
        fes_out(pEXT)(CURR) := i(pW);
      ElsIf MEL(FES_EXT)=Real(fesN) Then
        fes_out(fesW) := inN;
        fes_out(fesE) := inN;
        fes_out(fesS) := inN;
        fes_out(pEXT)(VOLT) := v(pN);
        fes_out(pEXT)(CURR) := i(pN);
      ElsIf MEL(FES_EXT)=Real(fesE) Then
        fes_out(fesW) := inE;
        fes_out(fesN) := inE;
        fes_out(fesS) := inE;
        fes_out(pEXT)(VOLT) := v(pE);
        fes_out(pEXT)(CURR) := i(pE);
      ElsIf MEL(FES_EXT)=Real(fesS) Then
        fes_out(fesW) := inS;
        fes_out(fesN) := inS;
        fes_out(fesE) := inS;
        fes_out(pEXT)(VOLT) := v(pS);
        fes_out(pEXT)(CURR) := i(pS);
      End If;
      fes_delay := EXT_DELAY;

    Else   -- *** Outros elementos

      -- Determinação da tensão entre os  polos e da
      -- corrente no elemento
      If MEL(PWD_POLE0)/=GROUNDED
        AND MEL(PWD_POLE1)/=GROUNDED Then
        i10 := (i(POLE1)+i(POLE0))/Real(2);
        v10 := v(POLE1) - v(POLE0);
      ElsIf MEL(PWD_POLE0)=GROUNDED Then
        i10 := i(POLE1);
        v10 := v(POLE1);
      Else
        i10 := i(POLE0);
        v10 := v(POLE0);
      End If;

      --- *** Resistor ***
      If MEL(ELEM)=RES Then
        i10 := (i10 + v10/MEL(VALUE))/2.0;
        v10 := i10*MEL(VALUE);
        fes_delay := RES_DELAY;

      --- *** Capacitor ***
      ElsIf MEL(ELEM)=CAP Then
        vf := v(POLE1);
        If Tx=MEL(T_INIT) OR (Abs(Vss0-vf) >
          Abs(0.1*Vss0) AND Tx/=MEL(T_INIT)) Then
          If Tx=MEL(T_INIT) Then
            v10 := 0.0;
          Else
            v10 := RegVI(VPOLE1)-RegVI(VPOLE0);
          End If;
          If MEL(PWD_POLE0)=GROUNDED OR
            MEL(PWD_POLE1)=GROUNDED Then
            v10 := (v(POLE1) - v(POLE0) + i10)/2.0
              + v10;
            If MEL(PWD_POLE0)=GROUNDED Then
              v(POLE1) := v10;
              vf := v10;
              v(POLE0) := 0.0;
            ElsIf MEL(PWD_POLE1)=GROUNDED Then
              v(POLE0) := -v10;
              vf := -v10;
              v(POLE1) := 0.0;
            End If;
          ElsIf MEL(PWD_POLE0)=POWERED Then
            v(POLE1) := v(POLE0) - v10;
          ElsIf MEL(PWD_POLE1)=POWERED Then
            v(POLE0) := v(POLE1) - v10;
          Else
            v(POLE0) := (v(POLE1) + v(POLE0))/2.0;
            v(POLE1) := v(POLE0) + v10;
            vf := v(POLE1);
          End If;
          Trans <= Tx;
          Izero <= i10;
          vacum := v10;
          Vcharge <= v10;
          Vss <= vf;
          If i10/=0.0 AND Tx=MEL(T_INIT) Then
            Tau <= Abs(vf*MEL(VALUE)/i10);
          End If;
        Else
          Assert Tau/=0.0
          Report "Erro no calculo do TAU"
```

```
      Severity Error;
      i10 := Izero*Exp(-(Tx-Trans)/Tau);
      v10 := Vss*(1.0-Exp(-(Tx-Trans)/Tau));
      v10 := v10 + Vcharge*Exp(-(Tx-Trans)/Tau);
    End If;
    fes_delay := CAP_DELAY;

--- *** Fonte de Tensão ***
ElsIf MEL(ELEM)=VOL Then
  If MEL(FREQ)=0.0 Then
    v10 := MEL(VOFFSET) + MEL(VALUE);
  Else
    v10 := MEL(VALUE) *
      Cos(2.0*PI*MEL(FREQ)*Tx);
    If Abs(v10) <
      Abs(MEL(VALUE)*MEL(CERROR)) Then
      v10 := 0.0;
    End If;
    v10 := v10 + MEL(VOFFSET);
  End If;
  If (now=0.0ns) Then
    i10 := 0.0;
  End If;
  If Tx'Event Then
    v10 := 0.99*(v10+MEL(CERROR));
  End If;
  fes_delay := VOL_DELAY;

--- *** Fonte de Tensão quadrada ***
ElsIf MEL(ELEM)=SQV Then
  v10 := MEL(VOFFSET);
  If Real(Integer(Tx*MEL(FREQ))) >
    Tx*MEL(FREQ) Then
    v10 := v10 - MEL(VALUE);
  Else
    v10 := v10 + MEL(VALUE);
  End If;
  If (now=0.0ns) Then
    i10 := 0.0;
  End If;
  If Tx'Event Then
    v10 := 0.99*(v10+MEL(CERROR));
  End If;
  fes_delay := SQV_DELAY;

--- *** Diodo ***
ElsIf MEL(ELEM)=DIO Then
  vt := 86.168E-6*(273.0+MEL(TEMPERATURE));
  If v10>=0.65 Then
    If i10=0.0 Then
      V10 := 0.65;
      i10 := MEL(ISAT)*(Exp(v10/vt)-1.0) +
        v10*1.0E-12;
    ElsIf i10>0.0 Then
      v10 := vt*Log(i10/MEL(ISAT)+1.0);
    Else
      v10 := -vt*Log(Abs(i10/MEL(ISAT)+1.0));
    End If;
  ElsIf v10<0.0 Then
    i10 := -MEL(ISAT)+v10*1.0E-12;
  Else
    If i10>0.0 Then
      v10 := vt*Log(i10/MEL(ISAT)+1.0);
    Else
      v10 := -vt*Log(Abs(i10/MEL(ISAT)+1.0));
    End If;
  End If;
  fes_delay := DIO_DELAY;

End If;

-- Atualização das tensões e corrente com os
-- valores calculados
If MEL(PWD_POLE0)=GROUNDED Then
  v(POLE1) := v10;
  v(POLE0) := 0.0;
ElsIf MEL(PWD_POLE1)=GROUNDED Then
  v(POLE0) := -v10;
  v(POLE1) := 0.0;
ElsIf MEL(PWD_POLE1)=POWERED Then
  v(POLE0) := v(POLE1) - v10;
Else
  v(POLE1) := v(POLE0) + v10;
End If;
i(POLE0) := i10;
i(POLE1) := i10;

-- Verificação da convergência local
cv := True;
For p In POLE0 To POLE3 Loop
  If (MEL(PWD_POLE0+p)/=GROUNDED) AND
    (MEL(PWD_POLE0+p)/=NONE) Then
    di(p) := Abs(i(p) - RegVI(Ipole0+p));
    dv(p) := Abs(v(p) - RegVI(Vpole0+p));
    ei(p) := Abs(MEL(CERROR)*i(p));
    ev(p) := Abs(MEL(CERROR)*v(p));
    cv := cv AND (di(p)<=ei(p))
      AND (dv(p)<=ev(p));
    If dv(p)=0.0 AND MEL(ELEM)=CAP Then
     v(p) := v(p)*(1.0-MEL(CERROR));
    End If;
    RegVI(Ipole0+p) <= i(p);
    RegVI(Vpole0+p) <= v(p);
  End If;
End Loop;

-- Atribuição das tensões e correntes
-- às FESs auxiliares
For f In fesW To fesS Loop
  If MEL(f)/=NONE Then -- FESx ativa ?
    If MEL(f)=Real(POLE0) Then
      fes_out(f)(VOLT) := v(POLE0);
      fes_out(f)(CURR) := i(POLE0);
    Else
      fes_out(f)(VOLT) := v(POLE1);
      fes_out(f)(CURR) := i(POLE1);
    End If;
  End If;
End Loop;
```

```
-- Passo para a simulação
Constant T_STEP : Integer := 25;

-- Constantes para os registradores para as correntes
-- e tensões (RegVI)
Constant Ipole0 : Integer := 0;
Constant Ipole1 : Integer := 1;
Constant Ipole2 : Integer := 2;
Constant Ipole3 : Integer := 3;
Constant Vpole0  : Integer := 4;
Constant Vpole1  : Integer := 5;
Constant Vpole2  : Integer := 6;
Constant Vpole3  : Integer := 7;

-- CONSTANTES PARA OS DISPOSITIVOS
-- A SEREM SIMULADOS

-- MPH de extensão
Constant EXT   : Real := 0.0;
-- Resistor
Constant RES   : Real := 1.0;
-- Capacitor
Constant CAP   : Real := 2.0;
-- Diodo
Constant DIO   : Real := 3.0;
-- Fonte de Tensão Independente
Constant VOL   : Real := 4.0;
-- Fonte de Tensão Quadrada Independente
Constant SQV   : Real := 5.0;

-- DEFINIÇÃO DAS CONSTANTES GLOBAIS

-- constantes para os polos do elemento
Constant POLE0  : Integer :=  0;
Constant POLE1  : Integer :=  1;
Constant POLE2  : Integer :=  2;
Constant POLE3  : Integer :=  3;
-- constante para as FESs Oeste, Norte, ...
--   usado como referência
Constant fesW  : Integer :=  1;
Constant fesN  : Integer :=  2;
Constant fesE  : Integer :=  3;
Constant fesS  : Integer :=  4;
-- Constante que indica valor NULO
Constant NONE   : Real := 99.0;
-- indica conexão a fonte de alimentação
Constant POWERED   : Real := 88.0;
-- indica conexão ao terra
Constant GROUNDED  : Real := 77.0;
-- indica não conexão ao terra nem a fonte
Constant NOTPWD   : Real := 66.0;
-- voltagem
Constant VOLT  : Integer := 0;
-- corrente
Constant CURR   : Integer := 1;
-- Tamanho da MEL
Constant MEMSIZE  : Integer := 31;
-- Número de registradores VI
Constant RegsNUM  : Integer := 8;

-- ESTRUTURAS GLOBAIS DE DADOS

-- Arranjo para o par tensão-corrente
Type vc_base Is Array(VOLT To CURR) Of Real;

-- Arranjo de pares tensão-corrente
Type fes_base Is Array(Natural Range <>) Of vc_base;

-- Arranjo de números reais
Type real_vector Is Array(Natural Range <>) Of Real;

-- Arranjo de números inteiros
Type int_vector Is Array(Natural Range <>) Of Integer;

-- FUNÇÕES DE PROPÓSITO GERAL

Function boolean2bit (b : Boolean) Return Bit;

-- CONSTANTES PARA O MÓDULO SWITCH

  -- Número de bits do reg_config
 Constant RC_BITNUM : Integer := 2;
  -- Atraso das chaves
 Constant SW_DELAY  : time    := 20 ns;
 Subtype bit4 Is Bit_Vector(0 To RC_BITNUM);
 -- Não há transferência de dados
 Constant NONE3  : bit4 := "0000";
 -- Transferência entre Oeste e Norte
 Constant W_N   : bit4 := "0001";
 -- Transferência  entre Oeste e Este
 Constant W_E   : bit4 := "0010";
 -- Transferência  entre Oeste e Sul
 Constant W_S   : bit4 := "0011";
 -- Transferência  entre Norte e Este
 Constant N_E   : bit4 := "0110";
 -- Transferência entre Norte e Sul
 Constant N_S   : bit4 := "0111";
 -- Transferência entre Este e Sul
 Constant E_S   : bit4 := "1011";

End aba_dados;

Package Body aba_dados Is

  -- Converte dados de tipo Boolean a tipo Bit
  Function boolean2bit (b : Boolean) Return Bit Is
  Begin
    If b Then Return '1'; Else Return '0';
    End If;
  End boolean2bit;

End aba_dados;
```

2. LISTAGENS DAS DESCRIÇÕES VHDL DO SEGUNDO NÍVEL DE ABSTRAÇÃO

2.1 DESCRIÇÃO RTL DAS MPHs

```
-- ************************************************************
-- Descrição estrutural das Unidades de Processamento
-- de Modelos
-- MPH
-- ************************************************************
Use Work.dados.ALL;

Entity mph Is

  Port(
    -- FES de entrada Oeste, Norte, ...
    vi_inW, vi_inN, vi_inE, vi_inS  : In vc_base;
    -- FES de Saída Oeste, Norte, ...
    vi_outW, vi_outN, vi_outE, vi_outS : Out vc_base;
    -- Inicia a simulação do instante em curso
    start   : In Bit;
    -- Volta ao instante inicial da simulação
    reset   : In Bit;
    -- Indica o estado da convergência local
    conv    : Out Bit );

End mph;

Architecture mph_stru Of mph Is

-- Declaração dos blocos da MPH

Component clock
  Port(  clk_out   : Out Bit);
End Component;

Component fes
  Port ( vi_in   :  In vc_base;
    vi_out  :  Out vc_base;
    ready   :  Out Bit );
End Component;

Component in_proc
  Port(  inW, inN, inE, inS  :  In vc_base;
    ready   :  Out Bit;
    start   : In Bit;
    comm  : In bit24;
    v0, v1  : Out Real;
    i0, i1  : Out Real );
End Component;

Component mei
  Port ( d_bus : InOut real_bus Bus;
    a_bus : In Integer;
    read  : In Bit;
    write : In Bit;
    ready : Out Bit );
End Component;

Component or4
  Port (  a,b,c,d : In Bit := '0';
    q  : Out Bit );
End Component;

Component ula
  Port ( v0_in, v1_in   : In Real;
    i0_in, i1_in  : In Real;
    ready   :  Out Bit;
    start   : In Bit;
    comm  : In bit24;
    d_bus   : In real_data;
    a_bus   : In Integer;
    write   : In Bit;
    v0_out, v1_out   : Out Real;
    i0_out, i1_out : Out Real );
End Component;

Component reg_vi
  Port ( v0_in, v1_in   : In Real;
    i0_in, i1_in  : In Real;
    load    : In Bit;
    read    : In Bit;
    reg   : In bit2;
    d_bus   : Out real_bus Bus;
    v0_out, v1_out   : Out Real;
    i0_out, i1_out : Out Real );
End Component;

Component cmp
  Port ( v0a, v1a : In Real;
    v0b, v1b : In Real;
    i0a, i1a   : In Real;
    i0b, i1b   : In Real;
    ready   :  Out Bit;
    start   : In Bit;
    comm  : In bit24;
    conv    : Out Bit;
    d_bus   : In real_data;
    write   : In Bit );
End Component;

Component out_proc
  Port ( v0, v1  : In Real;
    i0, i1   : In Real;
    start   : In Bit;
    comm  : In bit24;
    outW, outN, outE, outS    : Out vc_base );
```

```
End Component;

Component uce
  Port ( reset   : In Bit;
    start   : In Bit;
    conv    : Out Bit;
    clock   : In Bit;
    fes_rdy   : In Bit;
    in_proc_rdy   : In Bit;
    in_proc_start : Out Bit;
    in_proc_comm  : Out bit24;
    ULA_rdy : In Bit;
    ULA_start  : Out Bit;
    ULA_wr  : Out Bit;
    RG_ld : Out Bit;
    RG_rd   : Out Bit;
    RG_reg  : Out bit2;
    CMP_rdy   : In Bit;
    CMP_start  : Out Bit;
    CMP_wr : Out Bit;
    CMP_cv : In Bit;
    out_proc_start   : Out Bit;
    MEL_rd  : Out Bit;
    MEL_wr  : Out Bit;
    MEL_rdy   : In Bit;
    a_bus : Out Integer;
    d_bus : InOut real_bus Bus := NOT_DRIVEN);
End Component;

For fes_Oeste, fes_Norte, fes_Este, fes_Sul : fes Use Entity
Work.fes;
For or_rdy : or4 Use Entity Work.or4;
For proc_ent : in_proc Use Entity Work.in_proc;
For u_l_a : ula Use Entity Work.ula;
For memoria : mel Use Entity Work.mel;
For relogio : clock Use Entity Work.clock;
For control : uce Use Entity Work.uce;
For gerenciador : host Use Entity Work.host;
For registrador : reg_vi Use Entity Work.reg_vi;
For comparador : cmp Use Entity Work.cmp;
For proc_saida : out_proc Use Entity Work.out_proc;

-- Declaração dos sinais para a interconexão dos blocos

-- FESs de entrada e PROC-IN
Signal viW, viN, viE, viS   : vc_base;
Signal rdyW, rdyN, rdyE, rdyS : Bit;
-- PROC-IN e ULA
Signal v0A, v1A, i0A, i1A : Real;
-- ULA, REG-VI e CMP
Signal v0B, v1B, i0B, i1B : Real;
-- REG-VI, CMP e PROC-OUT
Signal v0C, v1C, i0C, i1C    : Real;
-- CLOCK e UCE
Signal ck   : Bit;
-- UCE sinais de controle
Signal rdyFES, ipRDY, ipST, ulaST, ulaRDY, ulaWR   : Bit;
Signal rgLD, rgRD    : Bit;
Signal rgREG   : bit2;
Signal cmpRDY, cmpST, cmpWR, cmpCV    : Bit;
Signal opST   : Bit;
Signal melRD, melWR, melRDY  : Bit;
Signal ipCOM    : bit24;
Signal data_bus    : real_bus;
Signal addr_bus    : Integer;

Begin

  -- FESs de entrada
  fes_Oeste  : fes Port Map (vi_inW, viW, rdyW);
  fes_Norte  : fes Port Map (vi_inN, viN, rdyN);
  fes_Este : fes Port Map (vi_inE, viE, rdyE);
  fes_Sul   : fes Port Map (vi_inS, viS, rdyS);

  -- OR dos sinais "ready" das FESs
  or_rdy : or4 Port Map (rdyW, rdyN, rdyE, rdyS, rdyFES);

  -- Processador de entrada - PROC-IN
  proc_ent : in_proc Port Map (
              viW, viN, viE, viS,
              ipRDY, ipST, ipCOM,
              v0A, v1A, i0A, i1A );

  -- Unidade Aritmética Lógica - ULA
  u_l_a  : ula Port Map (
              v0A, v1A,
              i0A, i1A,
              ulaRDY, ulaST, ipCOM,
              data_bus, addr_bus, ulaWR,
              v0B, v1B,
              i0B, i1B );

  -- Registradores para as tensões e a corrente - REG-VI
  registrador  : reg_vi Port Map (
              v0B, v1B,
              i0B, i1B,
              rgLD, rgRD, rgREG,
              data_bus,
              v0C, v1C,
              i0C, i1C );

  -- Comparador de tensões e correntes CMP
  comparador   : cmp Port Map (
              v0B, v1B,
              v0C, v1C,
              i0B, i1B,
              i0C, i1C,
              cmpRDY, cmpST, ipCOM, cmpCV,
              data_bus, cmpWR );

  -- Processador de saída - OUT-PROC
  proc_saida : out_proc Port Map (
              v0C, v1C,
              i0C, i1C,
              opST, ipCOM,
              vi_outW, vi_outN, vi_outE, vi_outS );

  -- Memória de Escrita e Leitura - MEL
  memoria : mel Port Map (
              data_bus, addr_bus,
```

```
                memRD, memWR, memRDY );

    -- Relógio - CLOCK
    relogio     : clock Port Map (ck);

    -- Unidade de Controle Elementar - UCE
    control     : uce Port Map (
                reset, start, conv,
                ck,
                rdyFES,
                ipRDY, ipST, ipCOM,
                ulaRDY, ulaST, ulaWR,
                rgLD, rgRD, rgREG,
                cmpRDY, cmpST, cmpWR, cmpCV,
                opST,
                memRD, memWR, memRDY,
                addr_bus, data_bus );

End mph_stru; -- fim da descrição da interface da MPH
```

```
-- ***************************************************************
-- Descrição comportamental da Unidade de Controle
-- Elementar
-- UCE
-- ***************************************************************
Use Work.dados.ALL;

Entity uce Is

  Port (
    -- Sinais para a ligação com o gerenciador
    reset   : In Bit;
    start   : In Bit;
    conv    : Out Bit;

    -- Sinal de relógio
    clock   : In Bit;

    -- Indica a presença de um novo dado nas FESs
    fes_rdy   : In Bit;

    -- Controle do bloco IN-PROC
    -- indica se IN-PROC tem o resultado pronto
    in_proc_rdy   :  In Bit;
    -- inicia um novo cálculo
    in_proc_start  : Out Bit;
    -- proporciona a configuração das FESs
    in_proc_comm   : Out bit24;

    -- Controle da ULA
    -- indica se a ULA tem o resultado pronto
    ULA_rdy : In Bit;
    -- inicia um novo ciclo de cálculo
    ULA_start   : Out Bit;
    -- escreve na memória interna da ULA
    ULA_wr  : Out Bit;

    -- Controle dos registradores de tensões e correntes
    -- REG-VI
    -- carrega o registradores
    RG_ld : Out Bit;
    -- lê um registrador interno
    RG_rd    : Out Bit;
    -- endereço do registrador interno
    RG_reg  : Out bit2;

    -- Controle do comparador - CMP
    -- avisa o final da comparação
    CMP_rdy    : In Bit;
    -- inicia o ciclo de comparação
    CMP_start  : Out Bit;
    -- escreve o erro de convergência no bloco CMP
    CMP_wr : Out Bit;
    -- indica o estado de convergência
    CMP_cv : In Bit;


    -- Inicia a operação do bloco PROC-OUT
    out_proc_start   : Out Bit;
```

```
    -- Controle da memória - MEL
    -- leitura
    MEL_rd  : Out Bit;
    -- escrita
    MEL_wr  : Out Bit;
    -- indica se o dado esta disponível
    MEL_rdy   : In Bit;

    -- Barramento de endereços
    a_bus : Out Integer;

    -- Barramento de dados
    d_bus : InOut real_bus Bus := NOT_DRIVEN);

End uce; -- fim da declaração da interface da UCE

-- Arquitetura comportamental da UCE

Architecture uce_arq Of uce Is

  -- Declaração dos Registradores internos
  Signal rst  : Bit;  -- reset
  Signal stt  : Bit;  -- start
  Signal frdy : Bit;  -- fes_rdy
  -- Sinais de reinicialização dos registradores internos
  Signal ffd_rst : Bit;  -- reset
  Signal ffd_stt : Bit;  -- start
  Signal ffd_fes  : Bit;  -- fes_rdy

Begin

  -- Controle do registrador para o sinal reset
  ffd_reset : Process (reset, ffd_rst)
  Begin
    -- borda de subida em reset ?
    If reset'Event And reset='1' Then
      rst <= '1' After Pdelay;
    -- borda de subida em ffd_rst ?
    ElsIf ffd_rst'Event And ffd_rst='1' Then
      rst <= '0' After Pdelay;
    End If;
  End Process ffd_reset;

  -- Controle do registrador para o sinal start
  ffd_start  : Process (start, ffd_stt)
  Begin
    -- borda de subida em start ?
    If start'Event And start='1' Then
      stt <= '1' After Pdelay;
    -- borda de subida em ffd_stt?
    ElsIf ffd_stt'Event And ffd_stt='1' Then
      stt <= '0' After Pdelay;
    End If;
  End Process ffd_start;


  -- Controle do registrador para o sinal fes_rdy
  ffd_fes_rdy : Process (fes_rdy, ffd_fes)
  Begin
    -- borda de subida em fes_rdy ?
    If fes_rdy'Event And fes_rdy='1' Then
      frdy <= '1' After Pdelay;
    -- borda de subida em ffd_fes ?
    ElsIf ffd_fes'Event And ffd_fes='1' Then
      frdy <= '0' After Pdelay;
    End If;
  End Process ffd_fes_rdy;

  -- ***************************************
  -- Processo de controle da UCE
  -- ***************************************
  uce_proc   : Process

    -- Declaração dos estados da máquina de estados
    Type uce_state Is
      ( INIT,
        NEXT_INSTANT,
        READ_IN_PROC,
        CALCULATE,
        COMPARE,
        STORE_AND_WRITE_FES,
        STANDBY );
    -- Registradores dos estados anterior
    -- e atual da máquina de estados
    Variable state0, state1 : uce_state;
    -- Registradores para o tempo inicial, passo de
    -- simulação e instante em curso
    Variable Tinit, Tstep, tt : Real := 0.0;
    -- Registradores temporais
    Variable tmp, disp : Real := NONE;

    -- Lê a Memória de Escrita e Leitura - MEL
    Procedure read_MEL(  addr    : In Integer;
                         result  : Out Real ) Is
    Begin
      -- coloca o endereço no barramento
      a_bus <= addr After Pdelay;
      -- aguarda pela borda próxima
      Wait Until clock='1';
      MEL_rd <= '1' After Pdelay;
      Loop
        -- aguarda até MEL_rdy estar ativo
        Wait Until clock='0';
        If MEL_rdy='1' Then
          -- colhe o dado do barramento
          result := d_bus(DATA);
          Exit;
        End If;
      End Loop;
      Wait Until clock='1';
      -- desativa o sinal MEL_rd
      MEL_rd <= '0' After Pdelay;
    End read_MEL;

    -- Transfere dados da MEL à ULA
    Procedure MEL_to_ULAreg(
      mel_addr, reg_addr  : In Integer ) Is
    Begin
      -- coloca o endereço da MEL no barramento
      a_bus <= mel_addr After Pdelay;
```

```
    Wait Until clock='1';
    -- ativa o sinal de leitura
    MEL_rd <= '1' After Pdelay;
    Loop
      Wait Until clock='0';
      -- aguarda até a MEL ficar pronta
      If MEL_rdy='1' Then
        -- coloca o endereço da memória da ULA
        a_bus <= reg_addr After Pdelay;
        Wait Until clock='1';
        -- transfere o dado da MEL à ULA
        ULA_wr <= '1', '0' After Pdelay;
        Exit;
      End If;
    End Loop;
    Wait Until clock='1';
    -- desativa o sinal de leitura
    MEL_rd <= '0' After Pdelay;
End MEL_to_ULAreg;

-- Escreve o dado do tempo na ULA
Procedure Tx_to_ULA( tx    : In Real ) Is
    Variable tmp  : real_data;
Begin
    -- coloca o endereço no barramento
    a_bus <= ULA_Tx After Pdelay;
    tmp(DATA) := tx;
    tmp(DRIVE) := 1.0;
    -- coloca o dado no barramento
    d_bus <= tmp After Pdelay;
    Wait Until clock='1';
    -- escreve o dado na memória da ULA
    ULA_wr <= '1', '0' After Pdelay;
    Wait Until clock='1';
    -- coloca o barramento em "alta impedância"
    d_bus <= NOT_DRIVEN After Pdelay;
End Tx_to_ULA;

-- Transfere uma tensão ou corrente do REG-VI à MEL
Procedure REGvi_to_MEL(
    ad_reg    : In bit2; ad_mel    : In Integer ) Is
Begin
    -- coloca o endereço no barramento
    a_bus <= ad_mel After Pdelay;
    -- endereça o registrador em REG-VI
    RG_reg <= ad_reg After Pdelay;
    -- ativa o sinal de escrita da MEL
    MEL_wr <= '1' After Pdelay;
    -- ativa o sinal de leitura de REG-VI
    RG_rd <= '1' After Pdelay;
    Loop
      -- espera o dado da MEL ficar pronto
      Wait Until clock='1';
      Exit When MEL_rdy='1';
    End Loop;
    -- desativa o sinal de escrita e leitura
    MEL_wr <= '0' After Pdelay;
    Wait Until clock='0';
    RG_rd <= '0' After Pdelay;
    d_bus <= NOT_DRIVEN After Pdelay;
End REGvi_to_MEL;

-- converte e envia o sinal de comm
Procedure send_comm Is
    Variable c0    : bit24;
    Variable tmp  : Real;
Begin
    -- é fonte ?
    read_MEL(ELEM, tmp);
    c0(supply) := boolean2bit(tmp=VOL Or tmp=SQV);
    -- configuração das FES Oeste, Norte, ...
    fes_loop : For f In fesW To fesS Loop
      read_MEL(fesW_POLE+f, tmp);
      c0(fesWused+4*f) := boolean2bit(tmp/=NONE);
      If tmp/=NONE Then
        c0(fesWpole+4*f) :=
          boolean2bit(Integer(tmp)=P1);
        read_MEL(PWD_fesW+f, tmp);
        c0(fesWpow+4*f) :=
          boolean2bit(tmp=POWERED);
        read_MEL(INPOLE_fesW+f, tmp);
        c0(fesWinpole+4*f) :=
          boolean2bit(Integer(tmp)=P1);
      End If;
    End Loop fes_loop;
    -- configuração dos polos 0 e 1
    pole_loop : For p In 0 To 1 Loop
      read_MEL(PWD_POLE0+p, tmp);
      If tmp=NONE Then
        c0(POLE0+2*p To POLE0+1+2*p) :=
          NOT_USED;
      ElsIf tmp=NOTPWD Then
        c0(POLE0+2*p To POLE0+1+2*p) :=
          COMNODE;
      ElsIf tmp=GROUNDED Then
        c0(POLE0+2*p To POLE0+1+2*p) := GNDED;
      ElsIf tmp=POWERED Then
        c0(POLE0+2*p To POLE0+1+2*p) := PWDED;
      End If;
    End Loop pole_loop;
    -- FES de extensao
    read_MEL(FES_EXT, tmp);
    If tmp=NONE Then
      c0(fesEXT To fesEXT+2) := NOEXT;
    ElsIf Integer(tmp)=fesW Then
      c0(fesEXT To fesEXT+2) := fes_W;
    ElsIf Integer(tmp)=fesN Then
      c0(fesEXT To fesEXT+2) := fes_N;
    ElsIf Integer(tmp)=fesE Then
      c0(fesEXT To fesEXT+2) := fes_E;
    ElsIf Integer(tmp)=fesS Then
      c0(fesEXT To fesEXT+2) := fes_S;
    End If;

    -- envia o sinal
    in_proc_comm <= c0 After Pdelay;
End send_comm;
```

```
Begin  -- *** uce_proc process ***

  -- coloca o barramento de dados em "alta impedância"
  d_bus <= NOT_DRIVEN After Pdelay;
  Wait Until clock = '1';
  -- reset ?
  If rst='1' Then
    -- desativa os sinais de leitura e escrita da MEL
    MEL_rd <= '0' After Pdelay;
    MEL_wr <= '0' After Pdelay;
    -- desativa o sinal de inicio do IN-PROC
    in_proc_start <= '0' After Pdelay;
    -- inicializa OUT-PROC
    out_proc_start <= '1', '0' After Pdelay;
    -- qual o dispositivo a simular
    read_MEL(ELEM,disp);
    If disp/=NONE Then
      -- lê os tempo inicial
      read_MEL(T_INIT, Tinit);
      -- lê o passo de simulação
      read_MEL(T_STEP, Tstep);
      -- envia o sinal comm
      send_comm;
      -- Escrita de dados nos registradores da ULA
      MEL_to_ULAreg(ELEM, ULA_ELEM);
      If disp=RES Then
        MEL_to_ULAreg(VALUE, ULA_VAL);
      ElsIf disp=SQV Or disp=VOL Then
        Tx_to_ULA(0.0);
        MEL_to_ULAreg(VALUE, ULA_VAL);
        MEL_to_ULAreg(VOFFSET, ULA_OFFSET);
        MEL_to_ULAreg(FREQ, ULA_FREQ);
      ElsIf disp=DIO Then
        MEL_to_ULAreg(TEMPERATURE, ULA_TEMP);
        MEL_to_ULAreg(ISAT, ULA_ISAT);
      End If;
      -- Escrita do ERRO de converg. no COMParador
      a_bus <= CERROR After Pdelay;
      Wait Until clock='1';
      MEL_rd <= '1' After Pdelay;
      Loop
        Wait Until clock='1';
        If MEL_rdy='1' Then
          CMP_wr <= '1', '0' After Pdelay;
          Exit;
        End If;
      End Loop;
      Wait Until clock='1';
      MEL_rd <= '0' After Pdelay;
    End If;
    state0 := INIT;
    ffd_rst <= '1', '0'After Pdelay;
  End If;

  -- A MPH esta simulando algum dispositivo ?
  If disp/=NONE Then

    -- o sinal start é ativo ?
    If stt='1' Then
      -- passa ao próximo estado
      state0 := NEXT_INSTANT;
      -- desativa o sinal de convergência
      conv <= '0' After Pdelay;
      -- reseta o registrador do sinal start
      ffd_stt <= '1', '0'After Pdelay;

  -- algum dado presenta nas FES ?
  ElsIf frdy='1' Then
    -- passa ao próximo estado
    state0 := READ_IN_PROC;
    -- reseta o registrador do sinal fes_rdy
    ffd_fes <= '1', '0'After Pdelay;
  Else
    state0 := state1;
  End If;

  -- Escolhe um estado da máquina de estados
  Case state0 Is

    -- estado inicial
    When INIT =>
      -- começa com o primeiro instante
      tt := Tinit;
      state1 := READ_IN_PROC;

    -- incrementa em um passo o tempo da simulação
    When NEXT_INSTANT =>
      tt := tt + Tstep;
      Wait Until clock='1';
      If disp=VOL Or disp=SQV Then
        Tx_to_ULA(tt);
      End If;
      state1 := READ_IN_PROC;

    -- processa as tensões e correntes
    When READ_IN_PROC =>
      Wait Until clock='1';
      -- inicia a operação do IN-PROC
      in_proc_start <=   '1' After Pdelay,
                         '0' After 2*Pdelay;
      Loop
        Wait Until clock='1';
        -- reset ? start? ou fes_rdy?
        If rst='1' Or stt='1' Or frdy='1' Then
          Exit;
        -- o IN-PROC terminou ?
        ElsIf in_proc_rdy='1' Then
          state1 := CALCULATE;
          Exit;
        End If;
      End Loop;

    -- calcula o modelo com os dados lidos
    When CALCULATE =>
      Wait Until clock='1';
      -- inicia a operação da ULA
      ULA_start <= '1' After Pdelay, '0' After 2*Pdelay;
      Loop
        Wait Until clock='1';
```

```
      -- reset? start? ou fes_rdy?
      If rst='1' Or stt='1' Or frdy='1' Then
        Exit;
      -- a ULA terminou?
      ElsIf ULA_rdy='1' Then
        state1 := COMPARE;
        Exit;
      End If;
    End Loop;

  -- compara os dados das iterações atual e anterior
  When COMPARE =>
    Wait Until clock='1';
    -- inicia a operação do comparador
    CMP_start <= '1' After Pdelay, '0' After 2*Pdelay;
    Loop
      Wait Until clock='1';
      -- reset ? start ? ou fes_rdy ?
      If rst='1' Or stt='1' Or frdy='1' Then
        Exit;
      -- terminou a comparação ?
      ElsIf CMP_rdy='1' Then
        -- transmite o estado de convergência
        conv <= CMP_cv After Pdelay;
        state1 := STORE_AND_WRITE_FES;
        Exit;
      End If;
    End Loop;

  -- Armazena as tensões e correntes nos
  -- do REG-VI.
  --  Escreve esses dados no bloco OUT-PROC
  -- Transfere as tenões e correntes à MEL
  When STORE_AND_WRITE_FES =>
    Wait Until clock='1';
    -- carrega os registradores de REG-VI
    RG_ld <=  '1' After Pdelay,
              '0' After 2*Pdelay;
    Wait Until clock='1';
    -- inicia a operação do OUT-PROC
    out_proc_start <=  '1' After Pdelay,
                       '0' After 2*Pdelay;
    -- transfere a tensão do polo 0 à MEL
    If rst='0' And stt='0' And frdy='0' Then
      REGvi_to_MEL(VPOLE0,VP0);
    End If;
    -- transfere a tensão do polo 1 à MEL
    If rst='0' And stt='0' And frdy='0' Then
      REGvi_to_MEL(VPOLE1,VP1);
    End If;
    -- transfere a corrente à MEL
    If rst='0' And stt='0' And frdy='0' Then
      REGvi_to_MEL(IPOLE0,IP0);
    End If;
    If rst='0' And stt='0' And frdy='0' Then
      REGvi_to_MEL(IPOLE1,IP1);
    End If;
    state1 := STANDBY;
      -- Estado de espera
    When STANDBY =>
      Null;

  End Case;

End If;   -- MEL(ELEM)/=NONE

End Process uce_proc;

End uce_arq;
```

```
-- ************************************************************************
-- Descrição RTL da Memória de Escrita e Leitura
-- MEL
-- ************************************************************************
Use Work.dados.ALL;

Entity MEL Is

  Port (
    -- barramento de dados
    d_bus : InOut real_bus Bus;
    -- barramento de endereços
    a_bus : In Integer;
    -- sinais de controle
    read  : In Bit; -- leitura
    write : In Bit; -- escrita
    ready : Out Bit -- disponibiliade
    );

  Signal mem   : real_vector(0 To MEMSIZE) :=
                 (Others=>NONE);

End MEL;

Architecture mel_arq Of MEL Is

Begin

  MEL_beh : Process
    Variable tmp  : real_data;
  Begin

    -- colocar o barramento de dados em "alta impedância"
    d_bus <= NOT_DRIVEN After Pdelay;
    ready <= '0' After Pdelay;

    -- Espera pelo commando de leitura ou escrita
    Wait Until (read='1') OR (write='1');

    If write='1' then
      ready <='1' After Pdelay;
      -- esperar até o final do ciclo de escrita
      Wait Until write='0';
      -- carregar na memoria
      mem(a_bus) <= d_bus(DATA);
    Else   -- read='1'
      -- colocar o dado no barramento de dados
      tmp(DATA) := mem(a_bus);
      tmp(DRIVE) := 1.0;
      d_bus <= tmp After Pdelay;
      ready <= '1' After Pdelay;
      -- aguardar ate o final do ciclo de leitura
      Wait Until read='0';
    End If;

  End Process MEL_beh;

End mel_arq;
```

```
-- ************************************************************************
-- Descrição comportamental do sinal de relógio
-- CLOCK
-- ************************************************************************
Use Work.dados.ALL;

Entity clock Is

  -- sinal de saída do relógio
  Port(  clk_out   : Out Bit );

End clock;

Architecture clk_arq Of clock Is

Begin

  clock_process: Process
  Begin
    -- atribuição ao sinal de relógio
    clk_out <=
          '1',
          '0' After HIGHdelay;
    -- aguarda pelo próximo ciclo
    Wait For CYCLE;
  End Process clock_process;

End clk_arq;
```

```
-- ***************************************************************************
-- Descrição comportamental das
-- Filas de Entrada e Saída
-- FES
-- ***************************************************************************
Use Work.dados.ALL;

Entity fes Is
  Port (
    -- canal de entrada
    vi_in   : In vc_base;
    -- canal de saída
    vi_out  : Out vc_base;
    -- sinal que avisa da disponibilidade do dado
    ready   : Out Bit );
End fes;

Architecture fes_arq Of fes Is
Begin
  data_in : Process (vi_in)
  Begin
    If Now/=0 ns Then
      -- ativa ready quando há um novo dado
      -- no canal de entrada
      ready <= '1' After Pdelay, '0' After 2*Pdelay;
      -- transfere o dado da entrada à saída
      vi_out <= vi_in After Pdelay;
    End If;
  End Process data_in;
End fes_arq;
```

```
-- ***************************************************************************
-- Descrição RTL de uma port OR de 4 entradas
-- OR4
-- ***************************************************************************
Entity or4 Is

 Port (
    -- entradas
    a,b,c,d : In Bit := '0';
    -- saída
    q  : Out Bit );

End or4;

Architecture or4_arq Of or4 Is

Begin

  -- atribuição ao sinal de saída
  q <= a OR b OR c OR d;

End or4_arq;
```

```
-- ************************************************************************
-- Descrição comportamental do bloco de processamento
-- de entrada
-- IN-PROC
-- ************************************************************************
Use Work.dados.ALL;

Entity in_proc Is
  Port(
    -- entradas oeste, norte, este e sul
    inW, inN, inE, inS  :  In vc_base;
    -- avisa o término da operação do bloco
    ready  :  Out Bit;
    -- inicia a operação
    start   : In Bit;
    -- configuração das FESs
    comm  : In bit24;
    -- tensões e correntes de saída
    v0, v1  : Out Real;
    i0, i1   : Out Real );
End in_proc;

Architecture in_proc_arq Of in_proc Is

  Signal rdy   : Bit := '0';

Begin

  in_proc_ctrl :  Process (start, rdy)
  Begin
    If rdy'Event And rdy='1' Then
      ready <= '1'After Pdelay;
    ElsIf start'Event And start='1' Then
      ready <= '0' After Pdelay;
    End If;
  End Process in_proc_ctrl;

  in_proc_behav :    Process (start)
    -- Corrente no dispositivo
    Variable i   : real_vector(P0 To P1);
    -- Tensões nos polos
    Variable v   : real_vector(P0 To P1);
    -- nro. de FES ligadas a um mesmo polo
    Variable ct  : int_vector(P0 To P1);
    -- polos
    Variable pW, pN, pE, pS  : Integer;
    -- FES de extensão
    Variable fes_ext    : bit3;
    -- acumula o tempo gasto no cálculo de v10 e i10.
    Variable T_delay   : time;

  Begin

    If start='1' Then

      -- Inicialização dos registradores
      i(P0 To P1) := (0.0,0.0);
      v(P0 To P1) := (0.0,0.0);
      ct(P0 To P1) := (0,0);
      T_delay := 0 ns;

-- Converte a inteiro a informação dos polos
pW := bit2int(comm(fesWpole));
pN := bit2int(comm(fesNpole));
pE := bit2int(comm(fesEpole));
pS := bit2int(comm(fesSpole));

-- Determina qual a FES de extensão
fes_ext(0 To 2) :=
(comm(fesEXT),comm(fesEXT+1),comm(fesEXT+2));

-- Leitura da FES Oeste. Verifica se a FES esta
-- conetada e se esta sendo utilizada como canal no
-- modo de MPH extensor. Se a FES estiver conetada
-- a uma fonte, no polo pW predomina a tensão da
-- fonte. A corrente é somada ou substraida,
-- dependendo da conetividade.

-- nao é FES de ext. e é ativa?
If fes_ext/=fes_W And comm(fesWused)='1' Then
  -- FES ligada à fonte ?
  If comm(fesWpow)='1' Then
    v(pW) := inW(VOLT);
    T_delay := T_delay + ASS_delay;
  Else
    v(pW) := v(pW) + inW(VOLT);
    ct(pW) := ct(pW) + 1;
    T_delay := T_delay + ADD_delay;
  End If;
  -- é fonte ou FES ligada à fonte?
  If comm(supply)='1' Or comm(fesWpow)='1' Then
    -- são polos iguais ?
    If comm(fesWpole)=comm(fesWinpole) Then
      i(pW) := i(pW) + inW(CURR);
      T_delay := T_delay + ADD_delay;
    Else
      i(pW) := i(pW) - inW(CURR);
      T_delay := T_delay + SUB_delay;
    End If;
  Else
    -- são polos iguais ?
    If comm(fesWpole)=comm(fesWinpole) Then
      i(pW) := i(pW) - inW(CURR);
      T_delay := T_delay + SUB_delay;
    Else
      i(pW) := i(pW) + inW(CURR);
      T_delay := T_delay + ADD_delay;
    End If;
  End If;
End If;

-- Leitura da FES Norte. O funcionamento é similar
-- ao da FES Oeste.
If fes_ext/=fes_N And comm(fesNused)='1' Then
  If comm(fesNpow)='1' Then
    v(pN) := inN(VOLT);
    T_delay := T_delay + ASS_delay;
  ElsIf bit2'(comm(POLE0+2*pN) &
    comm(POLE0+1+2*pN))/=PWDED Then
    v(pN) := v(pN) + inN(VOLT);
    ct(pN) := ct(pN) + 1;
```

```
        T_delay := T_delay + ADD_delay;
      End If;
      If comm(supply)='1' Or comm(fesNpow)='1' Then
        If comm(fesNpole)=comm(fesNinpole) Then
          i(pN) := i(pN) + inN(CURR);
          T_delay := T_delay + ADD_delay;
        Else
          i(pN) := i(pN) - inN(CURR);
          T_delay := T_delay + SUB_delay;
        End If;
      Else
        If comm(fesNpole)=comm(fesNinpole) Then
          i(pN) := i(pN) - inN(CURR);
          T_delay := T_delay + SUB_delay;
        Else
          i(pN) := i(pN) + inN(CURR);
          T_delay := T_delay + ADD_delay;
        End If;
      End If;
    End If;

    -- Leitura da FES Este. O funcionamento é similar
    -- ao da FES Oeste.
    If fes_ext/=fes_E And comm(fesEused)='1' Then
      If comm(fesEpow)='1' Then
        v(pE) := inE(VOLT);
        T_delay := T_delay + ASS_delay;
      ElsIf bit2'(comm(POLE0+2*pE) &
        comm(POLE0+1+2*pE))/=PWDED Then
        v(pE) := v(pE) + inE(VOLT);
        ct(pE) := ct(pE) + 1;
        T_delay := T_delay + ADD_delay;
      End If;
      If comm(supply)='1' Or comm(fesEpow)='1' Then
        If comm(fesEpole)=comm(fesEinpole) Then
          i(pE) := i(pE) + inE(CURR);
          T_delay := T_delay + ADD_delay;
        Else
          i(pE) := i(pE) - inE(CURR);
          T_delay := T_delay + SUB_delay;
        End If;
      Else
        If comm(fesEpole)=comm(fesEinpole) Then
          i(pE) := i(pE) - inE(CURR);
          T_delay := T_delay + SUB_delay;
        Else
          i(pE) := i(pE) + inE(CURR);
          T_delay := T_delay + ADD_delay;
        End If;
      End If;
    End If;

    -- Leitura da FES Sul. O funcionamento é similar
    -- ao da FES Oeste.
    If fes_ext/=fes_S And comm(fesSused)='1' Then
      If comm(fesSpow)='1' Then
        v(pS) := inS(VOLT);
        T_delay := T_delay + ASS_delay;
      ElsIf bit2'(comm(POLE0+2*pS) &
        comm(POLE0+1+2*pS))/=PWDED Then
        v(pS) := v(pS) + inS(VOLT);
        ct(pS) := ct(pS) + 1;
        T_delay := T_delay + ADD_delay;
      End If;
      If comm(supply)='1' Or comm(fesSpow)='1' Then
        If comm(fesSpole)=comm(fesSinpole) Then
          i(pS) := i(pS) + inS(CURR);
          T_delay := T_delay + ADD_delay;
        Else
          i(pS) := i(pS) - inS(CURR);
          T_delay := T_delay + SUB_delay;
        End If;
      Else
        If comm(fesSpole)=comm(fesSinpole) Then
          i(pS) := i(pS) - inS(CURR);
          T_delay := T_delay + SUB_delay;
        Else
          i(pS) := i(pS) + inS(CURR);
          T_delay := T_delay + ADD_delay;
        End If;
      End If;

      -- Calcula a media das tensoes nos polos
      av1 : For p In P0 To P1 Loop
        If ct(p)/=0 And ct(p)>1 Then
          v(p) := v(p)/Real(ct(p));
          T_delay := T_delay + DIV_delay;
        End If;
      End Loop av1;

      v0 <= v(P0) After T_delay;
      v1 <= v(P1) After T_delay;
      i0 <= i(P0) After T_delay;
      i1 <= i(P1) After T_delay;
      rdy <= '1' After T_delay-Pdelay, '0' After T_delay;

    End If;

  End Process in_proc_behav;

End in_proc_arq;
```

```
-- ************************************************************************
-- Descrição comportamental
-- da Unidade Lógica Aritmética
-- ULA
-- ************************************************************************
Use Work.dados.ALL;
Use std.math.ALL;

Entity ULA Is

  Port (
    -- tensões e correntes de entrada
    v0_in, v1_in   : In Real;
    i0_in, i1_in   : In Real;
    -- avisa a finalização da operação da ULA
    ready  :  Out Bit;
    -- inicia a operação da ULA
    start   : In Bit;
    -- configuração dos MPHs
    comm  : In bit24;
    -- barramento de dados
    d_bus  : In real_data;
    -- barramento de endereços
    a_bus  : In Integer;
    -- escreve na memória interna da ULA
    write   : In Bit;
    -- tensões e correntes de saída
    v0_out, v1_out   : Out Real;
    i0_out, i1_out : Out Real );

End ULA;

Architecture ULA_arq Of ULA Is

  -- memória interna da ULA
  Signal reg   : real_vector(0 To ULAMEMSIZE);
  Signal rdy   : Bit := '0';

Begin

  -- controla a execucão da ULA
  ULA_ctrl :  Process (start, rdy)
  Begin
    If start'Event And start='1' Then
      ready <= '0'After Pdelay;
    ElsIf rdy'Event And rdy='1' Then
      ready <= '1' After Pdelay;
    End If;
  End Process ULA_ctrl;

  -- escreve na memória interna da ULA
  reg_proc    : Process (write)
  Begin
    If write='1' Then
      reg(a_bus) <= d_bus(DATA) After Pdelay;
    End If;
  End Process reg_proc;

-- Cálculo do modelo do dispositivo
ULA_beh : Process (start)

  Variable v10   : Real;
  Variable i10   : Real;
  Variable vt  : Real;
  Variable dly   : Time;
  Variable tmp  : Real;

Begin

  If start='1' Then

    dly := 0 ns;

    -- Determinação da tensão entre os polos e da
    --  corrente no elemento
    -- os dois polos estão ligados ao terra?
    If bit2'(comm(POLE0) & comm(POLE0+1))/=GNDED
      And
      bit2'(comm(POLE1) & comm(POLE1+1))/=GNDED
      Then
      i10 := (i1_in+i0_in)/Real(2);
      v10 := v1_in - v0_in;
      dly := dly + 2*ASS_delay + DIV_delay
          + ADD_delay + SUB_delay;
    -- o polo 0 esta ligado ao terra?
    ElsIf bit2'(comm(POLE0) &
      comm(POLE0+1))/=GNDED Then
      i10 := i1_in;
      v10 := v1_in;
      dly := dly + 2*ASS_delay;
    Else
      i10 := i0_in;
      v10 := v0_in;
      dly := dly + 2*ASS_delay;
    End If;

    --- *** Resistor ***
    If reg(ULA_ELEM)=RES Then
      i10 := (i10 + v10/reg(ULA_VAL))/2.0;
      v10 := i10*reg(ULA_VAL);
      dly := dly + 2*ASS_delay + 2*DIV_delay
          + ADD_delay + MUL_delay;

    --- *** Fonte de Tensao ***
    ElsIf reg(ULA_ELEM)=VOL Then
      v10 := reg(ULA_OFFSET);
      dly := dly + ASS_delay + CMP_delay;
      If reg(ULA_FREQ)=0.0 Then
        v10 := v10 + reg(ULA_VAL);
        dly := dly + ADD_delay + ASS_delay;
      Else
        v10 := v10 + reg(ULA_VAL)
            *Cos(2.0*PI*reg(ULA_FREQ)*Reg(ULA_Tx));
        dly := dly + ADD_delay + 2*ASS_delay
            + 4*MUL_delay;
      End If;
```

```
--- *** Fonte de Tensao quadrada ***
ElsIf reg(ULA_ELEM)=SQV Then
  v10 := reg(ULA_OFFSET);
  tmp := reg(ULA_Tx)*reg(ULA_FREQ);
  dly := dly + 2*ASS_delay + MUL_delay
       + CMP_delay;
  If Real(Integer(tmp)) > tmp Then
    v10 := v10 - reg(ULA_VAL);
    dly := dly + ASS_delay + SUB_delay;
  Else
    v10 := v10 + reg(ULA_VAL);
    dly := dly + ASS_delay + ADD_delay;
  End If;

--- *** Diodo ***
ElsIf reg(ULA_ELEM)=DIO Then
  vt := 86.168E-6*(273.0+reg(ULA_TEMP));
  dly := dly + ASS_delay + MUL_delay
       + ADD_delay + CMP_delay;
  If v10>=0.65 Then
    dly := dly + CMP_delay;
    If i10=0.0 Then
      V10 := 0.65;
      i10 := reg(ULA_ISAT)*(Exp(v10/vt)-1.0)
           + v10*1.0E-12;
      dly := dly + 2*ASS_delay + 2*MUL_delay
           + EXP_delay;
      dly := dly + DIV_delay + SUB_delay
           + ADD_delay;
    ElsIf i10>0.0 Then
      v10 := vt*Log(i10/reg(ULA_ISAT)+1.0);
      dly := dly + CMP_delay + ASS_delay
           + MUL_delay;
      dly := dly + LOG_delay + DIV_delay
           + ADD_delay;
    Else
      v10 := -vt*Log(Abs(i10/reg(ULA_ISAT)+1.0));
      dly := dly + ASS_delay + MUL_delay
           + LOG_delay + DIV_delay + ADD_delay;
    End If;
  ElsIf v10<0.0 Then
    i10 := -reg(ULA_ISAT)+v10*1.0E-12;
    dly := dly + CMP_delay + ASS_delay
         + ADD_delay + MUL_delay;
  Else
    dly := dly + CMP_delay;
    dly := dly + ASS_delay + MUL_delay
         + LOG_delay + DIV_delay + ADD_delay;
    If i10>0.0 Then
      v10 := vt*Log(i10/reg(ULA_ISAT)+1.0);
    Else
      v10 := -vt*Log(Abs(i10/reg(ULA_ISAT)+1.0));
    End If;
  End If;

End If;

If bit2'(comm(POLE0) &
  comm(POLE0+1))/=GNDED Then
  v1_out <= v10 After dly;
  v0_out <= 0.0 After dly;
ElsIf bit2'(comm(POLE1) &
  comm(POLE1+1))/=GNDED Then
  v0_out <= -v10 After dly;
  v1_out <= 0.0 After dly;
ElsIf bit2'(comm(POLE1) &
  comm(POLE1+1))/=PWDED Then
  v0_out <= v1_in - v10 After dly;
Else
  v1_out <= v0_in + v10 After dly;
End If;
i0_out <= i10 After dly;
i1_out <= i10 After dly;

rdy <= '1' After dly-Pdelay, '0' After dly;

End If; -- start='1'

End Process ULA_beh;

End ULA_arq;
```

```
-- ************************************************************************
-- Descrição comportamental do bloco de registradores
-- de tensões e correntes
-- REG-VI
-- ************************************************************************
Use Work.dados.ALL;

Entity reg_vi Is

  Port (
    -- tensões e correntes de entrada
    v0_in, v1_in   : In Real;
    i0_in, i1_in : In Real;
    -- carrega os registradores
    load    : In Bit;
    -- lê um registrador
    read    : In Bit;
    -- endereça um registrador
    reg  : In bit2;
    -- barramento de dados
    d_bus : Out real_bus Bus := NOT_DRIVEN;
    -- tensões e correntes de saída
    v0_out, v1_out   : Out Real;
    i0_out, i1_out : Out Real );

End reg_vi;

Architecture reg_vi_arq Of reg_vi Is

  Signal v0, v1, i0, i1   : Real;

Begin

  -- lê o registrador endereçado por reg
  read_reg    : Process

    Variable tmp  : real_data;

  Begin

    -- coloca o barramento de dados em "alta impedância"
    d_bus <= NOT_DRIVEN After Pdelay;

    -- Espera pelo commando de leitura
    Wait Until (read='1');

    -- colocar o dado no barramento de dados
    Case reg Is
      When VPOLE0 =>
        tmp(DATA) := v0;
      When VPOLE1 =>
        tmp(DATA) := v1;
      When IPOLE0 =>
        tmp(DATA) := i0;
      When IPOLE1 =>
        tmp(DATA) := i1;
    End Case;
    tmp(DRIVE) := 1.0;
    d_bus <= tmp After Pdelay;
    -- aguardar ate o final do ciclo de leitura
    Wait Until read='0';

  End Process read_reg;

  -- carrega os registradores
  load_regs   : Process ( load )

  Begin
    If load='1' Then
      -- registradores para as tensoes
      v0 <= v0_in After Pdelay;
      v0_out <= v0_in After Pdelay;
      v1 <= v1_in After Pdelay;
      v1_out <= v1_in After Pdelay;
      -- registradores para a corrente
      i0 <= i0_in After Pdelay;
      i0_out <= i0_in After Pdelay;
      i1 <= i1_in After Pdelay;
      i1_out <= i1_in After Pdelay;
    End If;
  End Process load_regs;

End reg_vi_arq;
```

```
-- ************************************************************************
-- Descrição comportamental do
-- comparador de tensões e correntes
-- COMP
-- ************************************************************************
Use Work.dados.ALL;

Entity cmp Is

  Port (
    -- tensões e correntes a comparar
    v0a, v1a : In Real;
    v0b, v1b : In Real;
    i0a, i1a  : In Real;
    i0b, i1b  : In Real;
    -- avisa o fim da comparação
    ready  :  Out Bit;
    -- inicia a comparação
    start   : In Bit;
    -- configuração das FESs
    comm  : In bit24;
    -- avisa do estado de convergência
    conv   : Out Bit;
    -- barramento de dados
    d_bus  : In real_data;
    -- escreve no registrador ERR
    write   : In Bit );

End cmp;

Architecture cmp_arq Of cmp Is

  Signal rdy  : Bit := '0';
  Signal ERR   : Real;

Begin

  -- controla a operação do comparador
  cmp_ctrl :  Process (start, rdy)
  Begin
    If start'Event And start='1' Then
      ready <= '0'After Pdelay;
    ElsIf rdy'Event And rdy='1' Then
      ready <= '1' After Pdelay;
    End If;
  End Process cmp_ctrl;

  -- escreve no registrador ERR
  write_error  : Process (write)
  Begin
    If write='1' Then
      ERR <= d_bus(DATA) After Pdelay;
    End If;
  End Process write_error;

  -- compara as tensões e correntes de entrada
  verify_conv   : Process (start)

    Variable cv   : Boolean;
    Variable dly   : Time;
    Variable pole  : bit2;
    Variable di0, di1   : Real;   -- desvio na corrente
    Variable ei0, ei1   : Real;   -- erro na corrente
    Variable dv0, dv1   : Real;   -- desvio nas tensões
    Variable ev0, ev1   : Real;   -- erro nas tensões

  Begin

    -- o sinal start está ativo ?
    If start='1' Then

      dly := 0 ns;
      -- Verificação da convergencia para o polo 0
      cv := True;
      pole := bit2'(comm(POLE0) & comm(POLE0+1));
      If pole/=GNDED And pole/=NOT_USED Then
        di0 := Abs(i0a - i0b);
        dv0 := Abs(v0a - v0b);
        ei0 := Abs(ERR*i0a);
        ev0 := Abs(ERR*v0a);
        cv := cv AND (di0<=ei0) AND (dv0<=ev0);
        dly := dly + 4*ASS_delay + 2*SUB_delay
            + 2*MUL_delay + 2*CMP_delay;
      End If;

      -- Verificação da convergencia para o polo 1
      pole := bit2'(comm(POLE1) & comm(POLE1+1));
      If pole/=GNDED And pole/=NOT_USED Then
        di1 := Abs(i1a - i1b);
        dv1 := Abs(v1a - v1b);
        ei1 := Abs(ERR*i1a);
        ev1 := Abs(ERR*v1a);
        cv := cv AND (di1<=ei1) AND (dv1<=ev1);
        dly := dly + 4*ASS_delay + 2*SUB_delay
            + 2*MUL_delay + 2*CMP_delay;
      End If;

      conv <= boolean2bit(cv) After dly;
      rdy <= '1' After dly-Pdelay, '0' After dly;

    End If;   -- start='1'

  End Process verify_conv;

End cmp_arq;
```

```
-- ************************************************************************
-- Descrição comportamental
-- do bloco de processamento de saída
-- OUT-PROC
-- ************************************************************************
Use Work.dados.ALL;

Entity out_proc Is

  Port (
    -- tensões e correntes de entrada
    v0, v1  : In Real;
    i0, i1  : In Real;
    -- inicia a operação do bloco
    start   : In Bit;
    -- configuração das FESs de saída
    comm : In bit24;
    -- canais de saída
    outW, outN, outE, outS    : Out vc_base );

End out_proc;

Architecture out_proc_arq Of out_proc Is

Begin

  out_proc_behav    : Process (start)
    -- Vetor aux. para'as FESs de saida
    Variable fes_out    : fes_base(fesW To fesS)
                        := (Others=>(0.0,0.0));
  Begin

    If start='1' Then
      -- Atribuição das tensões e correntes
      -- às FESs auxiliares
      fes_loop : For f In fesW To fesS Loop
        -- FES ativa ?
        If comm(fesWused+4*f)='1' Then
          If comm(fesWpole+4*f)='0' Then
            fes_out(f)(VOLT) := v0;
            fes_out(f)(CURR) := i0;
          Else
            fes_out(f)(VOLT) := v1;
            fes_out(f)(CURR) := i1;
          End If;
        End If;
      End Loop fes_loop;
      outW <= fes_out(fesW) After Pdelay;
      outN <= fes_out(fesN) After Pdelay;
      outE <= fes_out(fesE) After Pdelay;
      outS <= fes_out(fesS) After Pdelay;

    End If;   -- start='1'

  End Process out_proc_behav;

End out_proc_arq;
```

```
-- ************************************************************************
-- Descrição das estruturas de dados, constantes
-- e funções globais
-- DADOS
-- ************************************************************************
Package dados Is

-- Constantes de atraso
  -- atraso unitario
  Constant Pdelay   : time := 10 ns;
  -- tempo do '1' no relogio
  Constant HIGHdelay : time := 20 ns;
  -- ciclo de relógio
  Constant CYCLE   : time := 50 ns;
  -- atribuição
  Constant ASS_delay    : time := 3*CYCLE;
  -- soma
  Constant ADD_delay    : time := 3*CYCLE;
  -- substração
  Constant SUB_delay    : time := 6*CYCLE;
  -- comparação
  Constant CMP_delay    : time := 6*CYCLE;
  -- divisão
  Constant DIV_delay  : time := 31*CYCLE;
  -- multiplicação
  Constant MUL_delay    : time := 18*CYCLE;
  -- exponenciação
  Constant EXP_delay    : time := 796*CYCLE;
  -- logaritmo
  Constant LOG_delay    : time := 690*CYCLE;

-- Constantes para o sinal "comm" do bloco IN-PROC
  Constant supply    : Integer := 0;
  Constant fesWused  : Integer := 1;
  Constant fesWpole  : Integer := 2;
  Constant fesWpow   : Integer := 3;
  Constant fesWinpole    : Integer := 4;
  Constant fesNused  : Integer := 5;
  Constant fesNpole  : Integer := 6;
  Constant fesNpow   : Integer := 7;
  Constant fesNinpole : Integer := 8;
  Constant fesEused  : Integer := 9;
  Constant fesEpole  : Integer := 10;
  Constant fesEpow : Integer := 11;
  Constant fesEinpole : Integer := 12;
  Constant fesSused  : Integer := 13;
  Constant fesSpole  : Integer := 14;
  Constant fesSpow : Integer := 15;
  Constant fesSinpole : Integer := 16;
  Constant POLE0   : Integer := 17;
  Constant POLE1   : Integer := 19;
  Constant fesEXT   : Integer := 21;

-- Endereços da memoria MEL
  Constant ELEM : Integer := 0;
  -- Polo ligado à FES Oeste, Norte, ...
    -- Usado como referência
    -- deve ser o endereço 1
  Constant fesW_POLE   : Integer := 1;
  Constant fesN_POLE   : Integer := 2;
```

```
Constant fesE_POLE   : Integer := 3;
Constant fesS_POLE   : Integer := 4;
-- FES entrada/saída do MPH extensora
Constant FES_EXT   : Integer := 5;
-- Indica se o polo está ligado à
-- fonte, terra ou nó comum
Constant PWD_POLE0   : Integer := 6;
Constant PWD_POLE1   : Integer := 7;
Constant PWD_POLE2   : Integer := 8;
Constant PWD_POLE3   : Integer := 9;
-- Indica se a FES está ligada à fonte
Constant PWD_fesW   : Integer := 10;
Constant PWD_fesN   : Integer := 11;
Constant PWD_fesE : Integer := 12;
Constant PWD_fesS : Integer := 13;
-- Indica o polo da MPH interligada
   -- as FES Oeste, Norte, ...
Constant INPOLE_fesW   : Integer := 14;
Constant INPOLE_fesN   : Integer := 15;
Constant INPOLE_fesE   : Integer := 16;
Constant INPOLE_fesS   : Integer := 17;

-- Erro para a verificação da convergência
Constant CERROR   : Integer := 18;
-- Valor da amplit., res., capac., ...
Constant VALUE   : Integer := 19;
-- Tensão de Offset para à fonte
Constant VOFFSET   : Integer := 20;
-- Frequência da fonte
Constant FREQ : Integer := 21;
Constant TEMPERATURE   : Integer := 22;
-- Corrente inversa de saturação do diodo
Constant ISAT   : Integer := 23;
-- Tempo inicial para a simulação
Constant T_INIT   : Integer := 24;
-- Passo para a simulação
Constant T_STEP : Integer := 25;
Constant VP0   : Integer := 26;  -- tensão do polo 0
Constant VP1   : Integer := 27;  -- tensão do polo 1
Constant IP0 : Integer := 28;  -- corrente no elemento
Constant IP1 : Integer := 29;

-- Registradores para a ULA
   -- dipositivo a ser simulado
   Constant ULA_ELEM   : Integer := 0;
   -- instante de simulação
   Constant ULA_Tx : Integer := 1;
   -- Valor da amplit., res., capac., ...
   Constant ULA_VAL   : Integer := 2;
   -- Tensão de Offset para 'a fonte
   Constant ULA_OFFSET   : Integer := 3;
   -- Frequência da fonte
   Constant ULA_FREQ   : Integer := 4;
   -- temperatura
   Constant ULA_TEMP   : Integer := 5;
   -- Corrente inversa de saturação do diodo
   Constant ULA_ISAT : Integer := 6;

-- Constantes para os dispositivos a serem simulados
   Constant EXT   : Real := 0.0; -- MPH de extensão
   Constant RES   : Real := 1.0; -- Resistor
   Constant CAP   : Real := 2.0; -- Capacitor
   Constant DIO   : Real := 3.0; -- Diodo
   Constant VOL   : Real := 4.0; -- Fonte de Tensão
   -- Fonte de Tensão Quadrada
   Constant SQV   : Real := 5.0;

-- Definição de tipos de dados
   Subtype bit24 Is Bit_Vector(0 To 23);
   Subtype bit4 Is Bit_Vector(0 To 3);
   Subtype bit3 Is Bit_Vector(0 To 2);
   Subtype bit2 Is Bit_Vector(0 To 1);

-- Constantes para o registrador de tensões e correntes
   Constant VPOLE0   : bit2 := B"00";  -- tensão do polo 0
   Constant VPOLE1   : bit2 := B"01";  -- tensão do polo 1
   Constant IPOLE0  : bit2 := B"10";  -- corrente no elemento
   Constant IPOLE1  : bit2 := B"11";

-- Definição de tipos e constantes para o bloco SWITCH
   Constant Wch   : bit2 := B"00";  -- canal Oeste
   Constant Nch   : bit2 := B"01";  -- canal Norte
   Constant Ech : bit2 := B"10";  -- canal Este
   Constant Sch : bit2 := B"11";  -- canal Sul
   -- Não há transferência de dados
   Constant NONE3   : bit4 := "0000";
   -- Transferência entre Oeste e Norte
   Constant W_N   : bit4 := "0001";
   -- Transferência entre Oeste e Este
   Constant W_E   : bit4 := "0010";
   -- Transferência entre Oeste e Sul
   Constant W_S   : bit4 := "0011";
   -- Transferência entre Norte e Este
   Constant N_E   : bit4 := "0110";
   -- Transferência entre Norte e Sul
   Constant N_S   : bit4 := "0111";
   -- Transferência entre Este e Sul
   Constant E_S   : bit4 := "1011";

-- Definicao das constantes globais
   Constant NOEXT   : bit3 := B"000"; -- MPH não extensora
   Constant fes_W   : bit3 := B"001"; -- FES oeste
   Constant fes_N : bit3 := B"010"; -- FES norte
   Constant fes_E : bit3 := B"011"; -- FES este
   Constant fes_S : bit3 := B"100"; -- FES sul
   -- polo não utilizado
   Constant NOT_USED   : bit2 := B"00";
   -- polo nao lig.à fonte nem ao terra
   Constant COMNODE   : bit2 := B"01";
   Constant GNDED : bit2 := B"10";  -- polo ligado ao terra
   Constant PWDED : bit2 := B"11";  -- polo ligadoà fonte
   Constant VOLT : Integer := 0;   -- Voltagem
   Constant CURR   : Integer := 1;   -- Corrente

   Constant P0   : Integer := 0;   -- polo 0
   Constant P1   : Integer := 1;   -- polo 1
   Constant fesW   : Integer := 0;
   Constant fesN   : Integer := 1;
   Constant fesE   : Integer := 2;
   Constant fesS   : Integer := 3;
```

```
  Constant NONE    : Real := 99.0;
  Constant POWERED   : Real := 88.0;
  Constant GROUNDED : Real := 77.0;
  Constant NOTPWD   : Real := 66.0;
  Constant MEMSIZE  : Integer := 31;  -- Tamanho da MEL
  -- Número de registradores - ULA
  Constant ULAMEMSIZE  : Integer := 6;

-- Definição das estruturas globais de dados

  -- Arranjo para o par tensão-corrente
  Type vc_base Is Array(VOLT To CURR) Of Real;

  -- Arranjo de pares tensão-corrente
  Type fes_base Is Array(Natural Range <>) Of vc_base;

  -- Arranjo de números reais
  Type real_vector Is Array(Natural Range <>) Of Real;

  -- Arranjo de números inteiros
  Type int_vector Is Array(Natural Range <>) Of Integer;

  -- Estrutura para a resolução do barramento de dados
  Constant DATA : Integer := 0;
  Constant DRIVE   : Integer := 1;
  Type real_data Is Array(DATA To DRIVE) Of Real;
  Type real_data_vec Is Array(Natural Range <>)
       Of real_data;
  Function resolve_real_bus(vec   : real_data_vec)
         Return real_data;
  Subtype real_bus Is resolve_real_bus real_data;
  Constant NOT_DRIVEN   : real_data := (0.0, 0.0);

  -- Estrutura para a resolução dos canais no bloco SWITCH
  Type vc_data Is Record
    data    : vc_base;
    drive   : Bit;
  End Record;
  Type vc_data_vec Is Array(Natural Range <>) Of vc_data;
  Function resolve_vc_bus(vec: vc_data_vec)
       Return vc_data;
  Subtype vc_bus Is resolve_vc_bus vc_data;
  Constant NODRIVEN   : vc_data := ((0.0, 0.0), '0');

-- Declaração das funções de propósito geral
  Function bit2int (b : Bit) Return Integer;
  Function boolean2bit (b : Boolean) Return Bit;

End dados;

Package Body dados Is

 -- Converte dados de tipo Bit a tipo Integer
  Function bit2int (b : Bit) Return Integer Is
  Begin
    If b='0' Then Return 0; Else Return 1;
    End If;
  End bit2int;

  -- Converte dados de tipo Boolean a tipo Bit
  Function boolean2bit (b : Boolean) Return Bit Is
  Begin
    If b Then Return '1'; Else Return '0';
    End If;
  End boolean2bit;

  -- Função de resolução para o barramento de dados
  Function resolve_real_bus(vec   : real_data_vec)
    Return real_data Is
    Variable result   : real_data := NOT_DRIVEN;
    Variable c   : Integer := 0;
  Begin
    c := 0;
    If vec'Length=0 Then
      Return NOT_DRIVEN;
    End If;
    For d In vec'Range Loop
      If vec(d)(DRIVE)/=0.0 Then
        c := c + 1;
        If c=1 Then
          result := vec(d);
        Else
          result := NOT_DRIVEN;
          Assert False
          Report "Detetou-se muitos drivers"
          Severity Error;
        End If;
      End If;
    End Loop;
    Return result;
  End resolve_real_bus;

  -- Função de resolução dos canais no bloco SWITCH
  Function resolve_vc_bus(vec: vc_data_vec) Return
                vc_data Is
    Variable result   : vc_data := NODRIVEN;
    Variable c   : Integer := 0;
  Begin
    c := 0;
    If vec'Length=0 Then
      Return NODRIVEN;
    End If;
    For d In vec'Range Loop
      If vec(d).drive/='0' Then
        c := c + 1;
        If c=1 Then
          result := vec(d);
        Else
          result := NODRIVEN;
          Assert False
          Report "Detetou-se muitos drivers"
          Severity Error;
        End If;
      End If;
    End Loop;
    Return result;
  End resolve_vc_bus;

End dados;
```

2.2 DESCRIÇÃO RTL DAS CHAVES

```
-- ************************************************************************
-- Descrição RTL das chaves
-- SWITCH
-- ************************************************************************

Use Work.dados.ALL;

Entity switch Is

  Port(
    -- Canais de entrada
    inW, inN, inE, inS  : In vc_base;
    -- Canais de saída
    outW, outN, outE, outS    :  Out vc_base );

  -- armazena a configuração da chave
  Signal reg_config : bit4 := W_N;

End switch;

Architecture sw_arq Of switch Is

-- demultiplexador
Component dmx1to4
  Port(
    vc_in   : In vc_base;
    outW, outN, outE, outS    :  Out vc_bus;
    sel   : In Bit2 );
End Component;

-- multiplexador
Component mux4to1
  Port(
    inW, inN, inE, inS  : In vc_base;
    vc_out    :  Out vc_base;
    sel   : In Bit2 );
End Component;

For muxA, muxB : mux4to1 Use Entity Work.mux4to1;
For dmxA, dmxB : dmx1to4 Use Entity Work.dmx1to4;

-- sinais "resolvidos" para os canais de saída
Signal Wb, Nb, Eb, Sb  : vc_bus := NODRIVEN;
-- sinais para a interligação dos blocos
Signal tmpA, tmpB    : vc_base;

Begin

  -- multiplexador A
  muxA :   mux4to1 Port Map(
                          inW, inN, inE, inS,
                          tmpA,
                          reg_config(0 To 1) );

  -- multiplexador B
  muxB :   mux4to1 Port Map(
                          inW, inN, inE, inS,
                          tmpB,
                          reg_config(2 To 3) );

  -- demultiplexador A
  dmxA :   dmx1to4 Port Map(
                          tmpA,
                          Wb, Nb, Eb, Sb,
                          reg_config(2 To 3) );
  -- demultiplexador B
  dmxB :   dmx1to4 Port Map(
                          tmpB,
                          Wb, Nb, Eb, Sb,
                          reg_config(0 To 1) );

  -- atribuição aos canais de saída
  outW <= Wb.data;
  outN <= Nb.data;
  outE <= Eb.data;
  outS <= Sb.data;

End sw_arq;
```

```
-- ************************************************************************
-- Descrição RTL de um multiplexador
-- de 4 canais a 1
-- mux4to1
-- ************************************************************************

Use Work.dados.ALL;

Entity mux4to1 Is

  Port(
    -- Entradas oeste, norte, este e sul
    inW, inN, inE, inS  : In vc_base;
    -- Saída do multiplexor
    vc_out    : Out vc_base;
    -- Seleciona o canal de entrada
    sel   : In Bit2 );

End mux4to1;

Architecture mux_arq Of mux4to1 Is

Begin

  With sel Select
    vc_out <= inW After Pdelay When Wch,
              inN After Pdelay When Nch,
              inE After Pdelay When Ech,
              inS After Pdelay When Sch;

End mux_arq;
```

```
-- ************************************************************************
-- Descrição RTL de um demultiplexador
-- de 1 a 4 canais
-- dmx4to1
-- ************************************************************************

Use Work.dados.ALL;

Entity dmx1to4 Is

  Port(
    -- Entrada
    vc_in   : In vc_base;
    -- Saídas oeste, norte, este e sul
    outW, outN, outE, outS    : Out vc_bus := NODRIVEN;
    -- Seleciona o canal de saída
    sel   : In Bit2 );

End dmx1to4;

Architecture dmx_arq Of dmx1to4 Is

Begin

  dmx_proc: Process (vc_in)
  Begin

    Case sel Is
      When Wch =>
        outW.data <= vc_in After Pdelay;
        outW.drive <= '1' After Pdelay;
        outN <= NODRIVEN After Pdelay;
        outE <= NODRIVEN After Pdelay;
        outS <= NODRIVEN After Pdelay;
      When Nch =>
        outW <= NODRIVEN After Pdelay;
        outN.data <= vc_in After Pdelay;
        outN.drive <= '1' After Pdelay;
        outE <= NODRIVEN After Pdelay;
        outS <= NODRIVEN After Pdelay;
      When Ech =>
        outW <= NODRIVEN After Pdelay;
        outN <= NODRIVEN After Pdelay;
        outE.data <= vc_in After Pdelay;
        outE.drive <= '1' After Pdelay;
        outS <= NODRIVEN After Pdelay;
      When Sch =>
        outW <= NODRIVEN After Pdelay;
        outN <= NODRIVEN After Pdelay;
        outE <= NODRIVEN After Pdelay;
        outS.data <= vc_in After Pdelay;
        outS.drive <= '1' After Pdelay;
    End Case;

  End Process dmx_proc;

End dmx_arq;
```

3. ARQUIVOS DE CONFIGURAÇÃO PARA A SIMULAÇÃO DOS CIRCUITOS TESTE

Arranjo para a simulação do circuito retificador de meia onda

```
// SET USer Scale -type Time 1e-09
// SETup FOrce -Charge

FORCe /I$1/mei(0) 4.0 0.0 -Abs

FORCe /I$1/mei(1) 99.0 0.0 -Abs
FORCe /I$1/mei(2) 99.0 0.0 -Abs
FORCe /I$1/mei(3) 1.0 0.0 -Abs
FORCe /I$1/mei(4) 99.0 0.0 -Abs

FORCe /I$1/mei(5) 99.0 0.0 -Abs

FORCe /I$1/mei(6) 77.0 0.0 -Abs
FORCe /I$1/mei(7) 66.0 0.0 -Abs
FORCe /I$1/mei(8) 99.0 0.0 -Abs
FORCe /I$1/mei(9) 99.0 0.0 -Abs

FORCe /I$1/mei(10) 99.0 0.0 -Abs
FORCe /I$1/mei(11) 99.0 0.0 -Abs
FORCe /I$1/mei(12) 99.0 0.0 -Abs
FORCe /I$1/mei(13) 99.0 0.0 -Abs

FORCe /I$1/mei(14) 99.0 0.0 -Abs
FORCe /I$1/mei(15) 99.0 0.0 -Abs
FORCe /I$1/mei(16) 1.0 0.0 -Abs
FORCe /I$1/mei(17) 99.0 0.0 -Abs

FORCe /I$1/mei(18) 1.0E-4 0.0 -Abs
FORCe /I$1/mei(19) 10.0 0.0 -Abs
FORCe /I$1/mei(20) 0.0 0.0 -Abs
FORCe /I$1/mei(21) 1000.0 0.0 -Abs
FORCe /I$1/mei(22) 27.0 0.0 -Abs
FORCe /I$1/mei(23) 0.0 0.0 -Abs
FORCe /I$1/mei(24) 0.0 0.0 -Abs
FORCe /I$1/mei(25) 0.00005 0.0 -Abs

FORCe /I$2/mei(0) 3.0 0.0 -Abs

FORCe /I$2/mei(1) 1.0 0.0 -Abs
FORCe /I$2/mei(2) 99.0 0.0 -Abs
FORCe /I$2/mei(3) 99.0 0.0 -Abs
FORCe /I$2/mei(4) 0.0 0.0 -Abs

FORCe /I$2/mei(5) 99.0 0.0 -Abs

FORCe /I$2/mei(6) 66.0 0.0 -Abs
FORCe /I$2/mei(7) 88.0 0.0 -Abs
FORCe /I$2/mei(8) 99.0 0.0 -Abs
FORCe /I$2/mei(9) 99.0 0.0 -Abs

FORCe /I$2/mei(10) 88.0 0.0 -Abs
FORCe /I$2/mei(11) 99.0 0.0 -Abs
FORCe /I$2/mei(12) 99.0 0.0 -Abs
FORCe /I$2/mei(13) 99.0 0.0 -Abs

FORCe /I$2/mei(14) 1.0 0.0 -Abs
FORCe /I$2/mei(15) 99.0 0.0 -Abs
FORCe /I$2/mei(16) 99.0 0.0 -Abs
FORCe /I$2/mei(17) 1.0 0.0 -Abs

FORCe /I$2/mei(18) 1.0E-4 0.0 -Abs
FORCe /I$2/mei(19) 0.0 0.0 -Abs
FORCe /I$2/mei(20) 0.0 0.0 -Abs
FORCe /I$2/mei(21) 0.0 0.0 -Abs
FORCe /I$2/mei(22) 27.0 0.0 -Abs
FORCe /I$2/mei(23) 1.0E-14 0.0 -Abs
FORCe /I$2/mei(24) 0.0 0.0 -Abs
FORCe /I$2/mei(25) 0.00005 0.0 -Abs

FORCe /I$3/mei(0) 1.0 0.0 -Abs

FORCe /I$3/mei(1) 99.0 0.0 -Abs
FORCe /I$3/mei(2) 1.0 0.0 -Abs
FORCe /I$3/mei(3) 99.0 0.0 -Abs
FORCe /I$3/mei(4) 99.0 0.0 -Abs

FORCe /I$3/mei(5) 99.0 0.0 -Abs

FORCe /I$3/mei(6) 77.0 0.0 -Abs
FORCe /I$3/mei(7) 66.0 0.0 -Abs
FORCe /I$3/mei(8) 99.0 0.0 -Abs
FORCe /I$3/mei(9) 99.0 0.0 -Abs

FORCe /I$3/mei(10) 99.0 0.0 -Abs
FORCe /I$3/mei(11) 99.0 0.0 -Abs
FORCe /I$3/mei(12) 99.0 0.0 -Abs
FORCe /I$3/mei(13) 99.0 0.0 -Abs

FORCe /I$3/mei(14) 99.0 0.0 -Abs
FORCe /I$3/mei(15) 0.0 0.0 -Abs
FORCe /I$3/mei(16) 99.0 0.0 -Abs
FORCe /I$3/mei(17) 99.0 0.0 -Abs

FORCe /I$3/mei(18) 1.0E-4 0.0 -Abs
FORCe /I$3/mei(19) 10000.0 0.0 -Abs
FORCe /I$3/mei(20) 0.0 0.0 -Abs
FORCe /I$3/mei(21) 0.0 0.0 -Abs
FORCe /I$3/mei(22) 27.0 0.0 -Abs
FORCe /I$3/mei(23) 0.0 0.0 -Abs
FORCe /I$3/mei(24) 0.0 0.0 -Abs
FORCe /I$3/mei(25) 0.00005 0.0 -Abs

FORCe /I$207/t_init 0.0 0.0 -Abs
FORCe /I$207/t_step 0.00005 0.0 -Abs
FORCe /I$207/t_end 0.1 0.0 -Abs
```

Arranjo para a simulação do circuito filtro passa-altas

```
// SET USer Scale -type Time 1e-09
// SETup FOrce -Charge

FORCe /I$1/mel(0) 5.0 0.0 -Abs

FORCe /I$1/mel(1) 99.0 0.0 -Abs
FORCe /I$1/mel(2) 99.0 0.0 -Abs
FORCe /I$1/mel(3) 1.0 0.0 -Abs
FORCe /I$1/mel(4) 99.0 0.0 -Abs

FORCe /I$1/mel(5) 99.0 0.0 -Abs

FORCe /I$1/mel(6) 77.0 0.0 -Abs
FORCe /I$1/mel(7) 66.0 0.0 -Abs
FORCe /I$1/mel(8) 99.0 0.0 -Abs
FORCe /I$1/mel(9) 99.0 0.0 -Abs

FORCe /I$1/mel(10) 99.0 0.0 -Abs
FORCe /I$1/mel(11) 99.0 0.0 -Abs
FORCe /I$1/mel(12) 99.0 0.0 -Abs
FORCe /I$1/mel(13) 99.0 0.0 -Abs

FORCe /I$1/mel(14) 99.0 0.0 -Abs
FORCe /I$1/mel(15) 99.0 0.0 -Abs
FORCe /I$1/mel(16) 1.0 0.0 -Abs
FORCe /I$1/mel(17) 99.0 0.0 -Abs

FORCe /I$1/mel(18) 1.0E-4 0.0 -Abs
FORCe /I$1/mel(19) 10.0 0.0 -Abs
FORCe /I$1/mel(20) 0.0 0.0 -Abs
FORCe /I$1/mel(21) 100.0 0.0 -Abs
FORCe /I$1/mel(22) 27.0 0.0 -Abs
FORCe /I$1/mel(23) 0.0 0.0 -Abs
FORCe /I$1/mel(24) 0.0 0.0 -Abs
FORCe /I$1/mel(25) 0.0005 0.0 -Abs

FORCe /I$3/mel(0) 2.0 0.0 -Abs

FORCe /I$3/mel(1) 1.0 0.0 -Abs
FORCe /I$3/mel(2) 99.0 0.0 -Abs
FORCe /I$3/mel(3) 99.0 0.0 -Abs
FORCe /I$3/mel(4) 0.0 0.0 -Abs

FORCe /I$3/mel(5) 99.0 0.0 -Abs

FORCe /I$3/mel(6) 66.0 0.0 -Abs
FORCe /I$3/mel(7) 88.0 0.0 -Abs
FORCe /I$3/mel(8) 99.0 0.0 -Abs
FORCe /I$3/mel(9) 99.0 0.0 -Abs

FORCe /I$3/mel(10) 88.0 0.0 -Abs
FORCe /I$3/mel(11) 99.0 0.0 -Abs
FORCe /I$3/mel(12) 99.0 0.0 -Abs
FORCe /I$3/mel(13) 99.0 0.0 -Abs

FORCe /I$3/mel(14) 1.0 0.0 -Abs
FORCe /I$3/mel(15) 99.0 0.0 -Abs
FORCe /I$3/mel(16) 99.0 0.0 -Abs
FORCe /I$3/mel(17) 1.0 0.0 -Abs

FORCe /I$3/mel(18) 1.0E-4 0.0 -Abs
FORCe /I$3/mel(19) 1.0E-6 0.0 -Abs
FORCe /I$3/mel(20) 0.0 0.0 -Abs
FORCe /I$3/mel(21) 0.0 0.0 -Abs
FORCe /I$3/mel(22) 27.0 0.0 -Abs
FORCe /I$3/mel(23) 0.0 0.0 -Abs
FORCe /I$3/mel(24) 0.0 0.0 -Abs
FORCe /I$3/mel(25) 0.0005 0.0 -Abs

FORCe /I$4/mel(0) 1.0 0.0 -Abs

FORCe /I$4/mel(1) 99.0 0.0 -Abs
FORCe /I$4/mel(2) 99.0 0.0 -Abs
FORCe /I$4/mel(3) 1.0 0.0 -Abs
FORCe /I$4/mel(4) 99.0 0.0 -Abs

FORCe /I$4/mel(5) 99.0 0.0 -Abs

FORCe /I$4/mel(6) 77.0 0.0 -Abs
FORCe /I$4/mel(7) 66.0 0.0 -Abs
FORCe /I$4/mel(8) 99.0 0.0 -Abs
FORCe /I$4/mel(9) 99.0 0.0 -Abs

FORCe /I$4/mel(10) 99.0 0.0 -Abs
FORCe /I$4/mel(11) 99.0 0.0 -Abs
FORCe /I$4/mel(12) 99.0 0.0 -Abs
FORCe /I$4/mel(13) 99.0 0.0 -Abs

FORCe /I$4/mel(14) 99.0 0.0 -Abs
FORCe /I$4/mel(15) 99.0 0.0 -Abs
FORCe /I$4/mel(16) 0.0 0.0 -Abs
FORCe /I$4/mel(17) 99.0 0.0 -Abs

FORCe /I$4/mel(18) 1.0E-4 0.0 -Abs
FORCe /I$4/mel(19) 1000.0 0.0 -Abs
FORCe /I$4/mel(20) 0.0 0.0 -Abs
FORCe /I$4/mel(21) 0.0 0.0 -Abs
FORCe /I$4/mel(22) 27.0 0.0 -Abs
FORCe /I$4/mel(23) 0.0 0.0 -Abs
FORCe /I$4/mel(24) 0.0 0.0 -Abs
FORCe /I$4/mel(25) 0.0005 0.0 -Abs

FORCe /I$5/t_init 0.0 0.0 -Abs
FORCe /I$5/t_step 0.0005 0.0 -Abs
FORCe /I$5/t_end 0.1 0.0 -Abs
```

Arranjo de 3x3 para a simulação do circuito retificador de meia onda

```
// SET USer Scale -type Time 1e-09
// SETup FOrce -Charge

FORCe /I$1/mel(0) 4.0 0.0 -Abs

FORCe /I$1/mel(1) 99.0 0.0 -Abs
FORCe /I$1/mel(2) 99.0 0.0 -Abs
FORCe /I$1/mel(3) 1.0 0.0 -Abs
FORCe /I$1/mel(4) 99.0 0.0 -Abs

FORCe /I$1/mel(5) 99.0 0.0 -Abs

FORCe /I$1/mel(6) 77.0 0.0 -Abs
FORCe /I$1/mel(7) 66.0 0.0 -Abs
FORCe /I$1/mel(8) 99.0 0.0 -Abs
FORCe /I$1/mel(9) 99.0 0.0 -Abs

FORCe /I$1/mel(10) 99.0 0.0 -Abs
FORCe /I$1/mel(11) 99.0 0.0 -Abs
FORCe /I$1/mel(12) 99.0 0.0 -Abs
FORCe /I$1/mel(13) 99.0 0.0 -Abs

FORCe /I$1/mel(14) 99.0 0.0 -Abs
FORCe /I$1/mel(15) 99.0 0.0 -Abs
FORCe /I$1/mel(16) 1.0 0.0 -Abs
FORCe /I$1/mel(17) 99.0 0.0 -Abs

FORCe /I$1/mel(18) 1.0E-4 0.0 -Abs
FORCe /I$1/mel(19) 10.0 0.0 -Abs
FORCe /I$1/mel(20) 0.0 0.0 -Abs
FORCe /I$1/mel(21) 1000.0 0.0 -Abs
FORCe /I$1/mel(22) 27.0 0.0 -Abs
FORCe /I$1/mel(23) 0.0 0.0 -Abs
FORCe /I$1/mel(24) 0.0 0.0 -Abs
FORCe /I$1/mel(25) 0.00005 0.0 -Abs

FORCe /I$2/mel(0) 3.0 0.0 -Abs

FORCe /I$2/mel(1) 1.0 0.0 -Abs
FORCe /I$2/mel(2) 99.0 0.0 -Abs
FORCe /I$2/mel(3) 99.0 0.0 -Abs
FORCe /I$2/mel(4) 0.0 0.0 -Abs

FORCe /I$2/mel(5) 99.0 0.0 -Abs

FORCe /I$2/mel(6) 66.0 0.0 -Abs
FORCe /I$2/mel(7) 88.0 0.0 -Abs
FORCe /I$2/mel(8) 99.0 0.0 -Abs
FORCe /I$2/mel(9) 99.0 0.0 -Abs

FORCe /I$2/mel(10) 88.0 0.0 -Abs
FORCe /I$2/mel(11) 99.0 0.0 -Abs
FORCe /I$2/mel(12) 99.0 0.0 -Abs
FORCe /I$2/mel(13) 99.0 0.0 -Abs

FORCe /I$2/mel(14) 1.0 0.0 -Abs
FORCe /I$2/mel(15) 99.0 0.0 -Abs
FORCe /I$2/mel(16) 99.0 0.0 -Abs
FORCe /I$2/mel(17) 1.0 0.0 -Abs

FORCe /I$2/mel(18) 1.0E-4 0.0 -Abs
FORCe /I$2/mel(19) 0.0 0.0 -Abs
FORCe /I$2/mel(20) 0.0 0.0 -Abs
FORCe /I$2/mel(21) 0.0 0.0 -Abs
FORCe /I$2/mel(22) 27.0 0.0 -Abs
FORCe /I$2/mel(23) 1.0E-14 0.0 -Abs
FORCe /I$2/mel(24) 0.0 0.0 -Abs
FORCe /I$2/mel(25) 0.00005 0.0 -Abs

FORCe /I$5/mel(0) 1.0 0.0 -Abs

FORCe /I$5/mel(1) 99.0 0.0 -Abs
FORCe /I$5/mel(2) 1.0 0.0 -Abs
FORCe /I$5/mel(3) 99.0 0.0 -Abs
FORCe /I$5/mel(4) 99.0 0.0 -Abs

FORCe /I$5/mel(5) 99.0 0.0 -Abs

FORCe /I$5/mel(6) 77.0 0.0 -Abs
FORCe /I$5/mel(7) 66.0 0.0 -Abs
FORCe /I$5/mel(8) 99.0 0.0 -Abs
FORCe /I$5/mel(9) 99.0 0.0 -Abs

FORCe /I$5/mel(10) 99.0 0.0 -Abs
FORCe /I$5/mel(11) 99.0 0.0 -Abs
FORCe /I$5/mel(12) 99.0 0.0 -Abs
FORCe /I$5/mel(13) 99.0 0.0 -Abs

FORCe /I$5/mel(14) 99.0 0.0 -Abs
FORCe /I$5/mel(15) 0.0 0.0 -Abs
FORCe /I$5/mel(16) 99.0 0.0 -Abs
FORCe /I$5/mel(17) 99.0 0.0 -Abs

FORCe /I$5/mel(18) 1.0E-4 0.0 -Abs
FORCe /I$5/mel(19) 10000.0 0.0 -Abs
FORCe /I$5/mel(20) 0.0 0.0 -Abs
FORCe /I$5/mel(21) 0.0 0.0 -Abs
FORCe /I$5/mel(22) 27.0 0.0 -Abs
FORCe /I$5/mel(23) 0.0 0.0 -Abs
FORCe /I$5/mel(24) 0.0 0.0 -Abs
FORCe /I$5/mel(25) 0.00005 0.0 -Abs


FORCe /I$10/reg_config(0) 0 0.0 -Abs
FORCe /I$10/reg_config(1) 0 0.0 -Abs
FORCe /I$10/reg_config(2) 1 0.0 -Abs
FORCe /I$10/reg_config(3) 0 0.0 -Abs

FORCe /I$902/reg_config(0) 0 0.0 -Abs
FORCe /I$902/reg_config(1) 0 0.0 -Abs
FORCe /I$902/reg_config(2) 1 0.0 -Abs
FORCe /I$902/reg_config(3) 0 0.0 -Abs

FORCe /I$215/reg_config(0) 0 0.0 -Abs
FORCe /I$215/reg_config(1) 1 0.0 -Abs
FORCe /I$215/reg_config(2) 1 0.0 -Abs
```

```
FORCe /I$215/reg_config(3) 1 0.0 -Abs

FORCe /I$911/reg_config(0) 0 0.0 -Abs
FORCe /I$911/reg_config(1) 1 0.0 -Abs
FORCe /I$911/reg_config(2) 1 0.0 -Abs
FORCe /I$911/reg_config(3) 1 0.0 -Abs


FORCe /I$656/t_init 0.0 0.0 -Abs
FORCe /I$656/t_step 0.00005 0.0 -Abs
FORCe /I$656/t_end 0.1 0.0 -Abs
```

Arranjo de 3x3 para a simulação do circuito filtro passa-altas

```
// SET USer Scale -type Time 1e-09
// SETup FOrce -Charge

FORCe /I$1/mel(0) 5.0 0.0 -Abs

FORCe /I$1/mel(1) 99.0 0.0 -Abs
FORCe /I$1/mel(2) 99.0 0.0 -Abs
FORCe /I$1/mel(3) 1.0 0.0 -Abs
FORCe /I$1/mel(4) 99.0 0.0 -Abs

FORCe /I$1/mel(5) 99.0 0.0 -Abs

FORCe /I$1/mel(6) 77.0 0.0 -Abs
FORCe /I$1/mel(7) 66.0 0.0 -Abs
FORCe /I$1/mel(8) 99.0 0.0 -Abs
FORCe /I$1/mel(9) 99.0 0.0 -Abs

FORCe /I$1/mel(10) 99.0 0.0 -Abs
FORCe /I$1/mel(11) 99.0 0.0 -Abs
FORCe /I$1/mel(12) 99.0 0.0 -Abs
FORCe /I$1/mel(13) 99.0 0.0 -Abs

FORCe /I$1/mel(14) 99.0 0.0 -Abs
FORCe /I$1/mel(15) 99.0 0.0 -Abs
FORCe /I$1/mel(16) 1.0 0.0 -Abs
FORCe /I$1/mel(17) 99.0 0.0 -Abs

FORCe /I$1/mel(18) 1.0E-4 0.0 -Abs
FORCe /I$1/mel(19) 10.0 0.0 -Abs
FORCe /I$1/mel(20) 0.0 0.0 -Abs
FORCe /I$1/mel(21) 100.0 0.0 -Abs
FORCe /I$1/mel(22) 27.0 0.0 -Abs
FORCe /I$1/mel(23) 0.0 0.0 -Abs
FORCe /I$1/mel(24) 0.0 0.0 -Abs
FORCe /I$1/mel(25) 0.0005 0.0 -Abs

FORCe /I$2/mel(0) 2.0 0.0 -Abs

FORCe /I$2/mel(1) 1.0 0.0 -Abs
FORCe /I$2/mel(2) 99.0 0.0 -Abs
FORCe /I$2/mel(3) 99.0 0.0 -Abs
FORCe /I$2/mel(4) 0.0 0.0 -Abs

FORCe /I$2/mel(5) 99.0 0.0 -Abs

FORCe /I$2/mel(6) 66.0 0.0 -Abs
FORCe /I$2/mel(7) 88.0 0.0 -Abs
FORCe /I$2/mel(8) 99.0 0.0 -Abs
FORCe /I$2/mel(9) 99.0 0.0 -Abs

FORCe /I$2/mel(10) 88.0 0.0 -Abs
FORCe /I$2/mel(11) 99.0 0.0 -Abs
FORCe /I$2/mel(12) 99.0 0.0 -Abs
FORCe /I$2/mel(13) 99.0 0.0 -Abs

FORCe /I$2/mel(14) 1.0 0.0 -Abs
```

```
FORCe /I$2/mel(15) 99.0 0.0 -Abs
FORCe /I$2/mel(16) 99.0 0.0 -Abs
FORCe /I$2/mel(17) 1.0 0.0 -Abs

FORCe /I$2/mel(18) 1.0E-4 0.0 -Abs
FORCe /I$2/mel(19) 1.0E-6 0.0 -Abs
FORCe /I$2/mel(20) 0.0 0.0 -Abs
FORCe /I$2/mel(21) 0.0 0.0 -Abs
FORCe /I$2/mel(22) 27.0 0.0 -Abs
FORCe /I$2/mel(23) 0.0 0.0 -Abs
FORCe /I$2/mel(24) 0.0 0.0 -Abs
FORCe /I$2/mel(25) 0.0005 0.0 -Abs

FORCe /I$4/mel(0) 1.0 0.0 -Abs

FORCe /I$4/mel(1) 99.0 0.0 -Abs
FORCe /I$4/mel(2) 99.0 0.0 -Abs
FORCe /I$4/mel(3) 1.0 0.0 -Abs
FORCe /I$4/mel(4) 99.0 0.0 -Abs

FORCe /I$4/mel(5) 99.0 0.0 -Abs

FORCe /I$4/mel(6) 77.0 0.0 -Abs
FORCe /I$4/mel(7) 66.0 0.0 -Abs
FORCe /I$4/mel(8) 99.0 0.0 -Abs
FORCe /I$4/mel(9) 99.0 0.0 -Abs

FORCe /I$4/mel(10) 99.0 0.0 -Abs
FORCe /I$4/mel(11) 99.0 0.0 -Abs
FORCe /I$4/mel(12) 99.0 0.0 -Abs
FORCe /I$4/mel(13) 99.0 0.0 -Abs

FORCe /I$4/mel(14) 99.0 0.0 -Abs
FORCe /I$4/mel(15) 99.0 0.0 -Abs
FORCe /I$4/mel(16) 0.0 0.0 -Abs
FORCe /I$4/mel(17) 99.0 0.0 -Abs

FORCe /I$4/mel(18) 1.0E-4 0.0 -Abs
FORCe /I$4/mel(19) 1000.0 0.0 -Abs
FORCe /I$4/mel(20) 0.0 0.0 -Abs
FORCe /I$4/mel(21) 0.0 0.0 -Abs
FORCe /I$4/mel(22) 27.0 0.0 -Abs
FORCe /I$4/mel(23) 0.0 0.0 -Abs
FORCe /I$4/mel(24) 0.0 0.0 -Abs
FORCe /I$4/mel(25) 0.0005 0.0 -Abs


FORCe /I$10/reg_config(0) 0 0.0 -Abs
FORCe /I$10/reg_config(1) 0 0.0 -Abs
FORCe /I$10/reg_config(2) 1 0.0 -Abs
FORCe /I$10/reg_config(3) 0 0.0 -Abs

FORCe /I$902/reg_config(0) 0 0.0 -Abs
FORCe /I$902/reg_config(1) 0 0.0 -Abs
FORCe /I$902/reg_config(2) 1 0.0 -Abs
FORCe /I$902/reg_config(3) 0 0.0 -Abs

FORCe /I$441/reg_config(0) 1 0.0 -Abs
FORCe /I$441/reg_config(1) 0 0.0 -Abs
FORCe /I$441/reg_config(2) 1 0.0 -Abs
FORCe /I$441/reg_config(3) 1 0.0 -Abs

FORCe /I$904/reg_config(0) 0 0.0 -Abs
FORCe /I$904/reg_config(1) 0 0.0 -Abs
FORCe /I$904/reg_config(2) 1 0.0 -Abs
FORCe /I$904/reg_config(3) 0 0.0 -Abs

FORCe /I$215/reg_config(0) 0 0.0 -Abs
FORCe /I$215/reg_config(1) 0 0.0 -Abs
FORCe /I$215/reg_config(2) 0 0.0 -Abs
FORCe /I$215/reg_config(3) 1 0.0 -Abs

FORCe /I$909/reg_config(0) 0 0.0 -Abs
FORCe /I$909/reg_config(1) 1 0.0 -Abs
FORCe /I$909/reg_config(2) 1 0.0 -Abs
FORCe /I$909/reg_config(3) 1 0.0 -Abs

FORCe /I$433/reg_config(0) 0 0.0 -Abs
FORCe /I$433/reg_config(1) 0 0.0 -Abs
FORCe /I$433/reg_config(2) 0 0.0 -Abs
FORCe /I$433/reg_config(3) 1 0.0 -Abs


FORCe /I$656/t_init 0.0 0.0 -Abs
FORCe /I$656/t_step 0.0005 0.0 -Abs
FORCe /I$656/t_end 0.1 0.0 -Abs
```

Bibliografia

[Arm89] J. R. Armstrong, "Chip-level Modeling with VHDL", Prentice-Hall, 1989.

[Bol93] C. Bolchini, et alii., "A Design Methodology for the Correct Specification of VLSI Systems", Microprocessing and Microprogramming, Vol.38, N.1-5, p.536-70, North-Holland 1993.

[Bou92] D. Boussebha, N. Giambiasi and J. Magnier, "Temporal Verification of Behavioral Descriptions in VHDL", EURO-DAC '92 European Design Automation Conference EURO-VHDL'92, p.692-7, 1992.

[Cam92] R. Camposano, "High-Level Synthesis", II Brazilian MIcroelectronics School, p.93-128, Gramado-RS, Brazil, March 1992.

[Car92] S. Carlson e E. Girczyc, "Understanding Synthesis begins with Knowing the Terminology", EDN, p.125-31, September 3, 1992.

[Cox91] P.F. Cox, et alii., "Direct circuit simulation algorithms for parallel processing", IEEE Transactions on Computer-Aided Design, Vol.10, N.6, p.714-25, June 1991.

[Dar90] J.A. Darringer e F.J. Ramming (editores), "Using VHDL as a Language for Synthesis of CMOS VLSI Circuits", Computer Hardware Description Languages and their Applications, IFIP 1990, p.331-43, North-Holland, 1990.

[Das89] S. Dasgupta, "Computer Architecture", J. Wiley & Sons Inc., vol.2, 1989.

[Dem78] B.P. Demidovich e I.A. Maron, "Computational Mathematics", Mir Publishers, 687pp., 1987.

[Deu84] J.T. Deustch and A.R. Newton, "A multiprocessor implementation of accurate electrical circuit simulation algorithm", Proc. 19th IEEE/ACM Design Automation Conference, Las Vegas, NV, June 1984.

[Dew92] A. Dewey e A.J. De Geus, "VHDL: Toward a Unified View of Design", IEEE Design & Test of Computers, Vol.9, Iss.2, p.8-17, June 1992.

[Eck92] W. Ecker e S. März, "Subtype Concept of VHDL for Synthesis Constraints", EURO-DAC '92 European Design Automation Conference EURO-VHDL '92, p.720-5, 1992.

[Fat83] E.T. Fathi e M. Krieger, "Multiple microprocessor systems: what, why, and when", Computer, p.23-32, March 1983.

[Gre86] B. Greer, "Converting SPICE to vector code", VLSI Systems Design, Vol.VII, p.30-35, Jan.1986.

[Hur92] S.L. Hurst, "Custom VLSI Microelectronics", Prentice Hall International Ltd., Hemel Hempstead, 1992.

[INM88] INMOS Inc., "The Transputer Databook", Bath Press Ltd., 477pp. 1988.

[Kho92] K.Y. Kho e J. Cong, "A fast multilayer general area router for MCM designs", EURO-DAC'92 European Design Automation Conference, p.292-7, 1992.

[Lel82] E. Lelarasmee, A.E. Ruelhi e A.L. Sangiovanni-Vincentelli, "The waveform relaxation method for time domain analysis of large scale integrated circuits", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol.1, N.3, p.131-45, July 1992.

[Lie92] J. Lienig, K. Thulasiraman e M.N.S. Swamy, "Routing algorithms for multi-chip modules", EURO-DAC'92 European Design Automation Conference, p.286-91, 1992.

[Lip89] R. Lipsett, C. Schaefer and C. Ussery, "VHDL: Hardware Description and Design", Kluwer Academic Publishers, 299pp., 1989.

[Mar90a] N. Marranghello e F. Damiani, "A methodology for circuit simulation", Proceedings of the 1990 Summer Computer Simulation Conference, p.111-15, Calgary-Canada, July 1990.

[Mar90b] N. Marranghello e F. Damiani, "ABACUS: a hardware-based circuit simulator" SID Digest of Technical Papers, p.177-179, 1990.

[Mar90c] N. Marranghello e F. Damiani, "Architecture and Algorithm of a circuit simulator", Proceeding of the Society for Photo-Optical Engineering (SPIE), vol. 1405, p.167-173, 1990.

[Mar92] N. Marranghello, "Uma Metodologia para a Simulação de Circuitos ULSI", Tese de Doutorado, FEE/UNICAMP, 105 pp., 1992.

[Mea88] L.G. Meares e C.E. Hymowitz, "Simulating With Spice", Intusoft, San Pedro-California, 1988.

[Mis92] C.A. Missio M. e A.A. Suzim, "Descrição e Simulação de Sistemas em Linguagem de Alto Nível", VII Congreso da Sociedade Brasileira de Microeletrônica, p.341-51, USP/São Paulo, Julho 1992.

[Nag92] V. Nagasamy, N. Berry e C. Dangelo, "Specification, Planning and Synthesis", IEEE Design & Test of Computers, Vol.9, Iss.2, p.58-68, June 1992.

[Nav91] Z. Navabi e M. Massoumi, "Investigating Simulation of Hardware at Various Levels of Abstraction and Timing Back-annotation of Data Flow Descriptions", Simulation, Vol.57, Iss.5, p.321-32, Nov. 1991.

[Per91] D. L. Perry, "VHDL", McGraw-Hill Inc., 459pp., 1991.

[IEE87] IEEE Standard VHDL Language Reference Manual, IEEE Standard 1076-1987.

[Sal83] R.A. Saleh, J.E. Kleckner e A.R.Newton, "Iterated timing analysis and SPLICE1", Proc. Int. Conf. on Computer-Aided Design, Santa Clara, CA, Sept 1983.

[Sal89] R.A. Saleh, et alii., "Parallel circuit simulation on supercomputers", Proc. of the IEEE, Vol.77, N.12, p.1915-31, Dec. 1989.

[Sal91] R.A. Saleh, T. Inoue e S. Ido, "Enhanced circuit simulation: expectations, problems, implementation, and integration", Electronics and Comunications in Japan, Part.3, Vol.74, N.11, 1991.

[Sav93] P. Saviz e O. Wing, "Circuit simulation by hierarchical waveform relaxation", Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol.12, N.6, p.845-60, June1993.

[Vis91] Ch. Visweswariah e R.A. Rohrer, "Piecewise aproximate circuit simulation", IEEE Transactions on Computer-Aided Design, Vol.10, N.7, p.861-70, July 1991.

[Vla82] A. Vladimirescu and D.O. Pederson, "Circuit simulation on vector processors", Proc. IEEE ICCC'82, p.172-175, 1982.

[Wee73] W.T. Weeks, et alii., "Algorithms for ASTAP - A Network Analysis Program", IEEE Transactions on Circuit Theory, Vol.CT-20, N.6, p.628-34, Nov. 1973.

[Whi86] J. White e A.S. Vincentelli, "Relaxation techniques for the simulation of VLSI circuits", Norwell, MA, Kluwer Academic Publishers, 1986.