UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

# PROPOSTA DE UMA ARQUITETURA ESPECIAL PARA SIMULAÇÃO LÓGICA

por

LEANDRO FORTES REY

Dissertação submetida como requisito parcial para
a obtenção do grau de Mestre em
Ciência da Computação

Prof. Flávio Rech Wagner
Orientador

Porto Alegre, Junho de 1988

À minha esposa Beatriz

CATALOGAÇÃO NA FONTE

Rey, Leandro Fortes

Proposta de uma arquitetura especial para simulação lógica. Porto Alegre, PGCC da UFRGS, 1988.

1 v.

Diss. (mestr. ci. comp.) UFRGS-PGCC, Porto Alegre, BR-RS, 1988.

Dissertação: Arquitetura de Computadores: Simulação lógica

## AGRADECIMENTOS

Ao Centro de Processamento de Dados da Universidade Federal do Rio Grande do Sul, pelas facilidades oferecidas ao aprimoramento profissional.

Ao meu orientador por seu interesse e colaboração no desenvolvimento do trabalho.

Aos meus colegas de CPD Jussara e Cirano pelo apoio e amizade.

# SUMÁRIO

# LISTA DE ABREVIATURAS

| | |
|---|---|
| AESL | Arquitetura Especial Para Simulação Lógica |
| ASIC | circuitos integrados de aplicação específica |
| CPD | Centro de Processamento de Dados |
| CPGCC | Curso de Pós-Graduação em Ciência da Computação |
| DMA | acesso direto à memória |
| ED | estrutura de dados |
| P. ex. | por exemplo |
| PLA | matriz lógica programável |
| RAM | memória de escrita e leitura |
| ROM | memória somente de leitura |
| UFRGS | Universidade Federal do Rio Grande do Sul |
| VLSI | integração em escala muito grande |

# LISTA DE FIGURAS

# LISTA DE TABELAS

## RESUMO

O objetivo deste trabalho é a proposta de uma arquitetura especial para simulação lógica ( AESL ).

As técnicas e modelos utilizados no processo de simulação lógica são brevemente revistos.

É definida uma taxonomia para AESL sob a qual são analisadas diversas propostas de AESL relatadas na literatura. Uma taxonomia já existente é comparada com a proposta.

A AESL definida é programável para diferentes algoritmos de simulação lógica. O detalhamento da AESL é, então, incrementado pela implementação de um algoritmo particular. Uma linguagem de simulação discreta é utilizada na construção de um modelo da arquitetura. Os resultados da simulação deste modelo permitem avaliar o desempenho da AESL e otimizar sua estrutura. Uma comparação com outras arquiteturas conclui a análise.

## ABSTRACT

This work proposes a dedicated architecture for logic simulation ( AESL ).

Several techniques and models used are briefly reviewed.

A taxonomy for AESL is defined, under wich various proposals for AESL reported in the literature are analysed. Also a comparison is made between the proposed taxonomy and one already published.

The proposed AESL is programmable for differing logic simulation algorithms. For the refinement of the proposal a particular algorithm is thus implemented. A discrete simulation language is used to build a model for the proposed architecture. The results of the simulation allow the evaluation of the AESL performance and the optimization of its structure. The analysis is completed with a comparison with other architectures.

# 1 INTRODUÇÃO

A tendência da indústria de computadores é a utilização crescente de circuitos VLSI, que apresentam uma grande vantagem na relação custo/desempenho. Tais circuitos trazem, contudo ,uma série de comprometimentos de projeto, tais como:

- ciclo de projeto mais longo;
- maior dificuldade na descoberta de erros;
- aumento da relação custo/erro nas fases finais de projeto;
- dificuldades nas modificações " em campo ".

Em vista destes fatores, a utilização de ferramentas de PAC torna-se a única maneira de projetar eficientemente sistemas digitais de grande complexidade, diminuindo drasticamente o ciclo de projeto e a ocorrência de erros.

Dentre as formas de validação de projeto, os simuladores são as ferramentas mais largamente utilizadas. Simulação pode ser realizada em diferentes níveis de abstração, dependendo do tipo de sistema a ser projetado ( p. ex. simulação funcional para sistemas construídos a partir de blocos já pré-definidos, simulação elétrica para circuitos a serem integrados ). A simulação lógica é uma das mais largamente utilizadas, por sua aplicação tanto ao projeto de sistemas construídos com componentes discretos como de circuitos integrados.

A simulação lógica de um sistema consiste na utilização de um modelo " em software " para o mesmo, onde o sistema é representado por uma interconexão de portas lógicas elementares, de forma que se obtenha o seu comportamento em função de um conjunto de estímulos de entrada.

O desempenho de um simulador é medido pelo tempo real necessário para simular um sistema, durante um intervalo de tempo de simulação. Simuladores convencionais não ultrapassam a taxa de avaliações de 20.000 portas lógicas/s [BAR80], apesar de já terem sido relatadas taxas de 90.000 portas lógicas/s [KRO81]. Aumentos na complexidade do modelo adotado (maior número de valores lógicos, vários tipos de atraso de propagação, etc) tendem a diminuir ainda mais este valor. Um simulador $10^8$ vezes mais lento que um sistema real pode levar 15 anos para executar um programa de teste que demoraria 5 segundos no sistema real [KOI85].

O processo de simulação lógica pode ser dividido em quatro etapas [ALL85]:

- definir o sistema;
- compilar o sistema;
- simular o sistema a partir dos estímulos de entrada
- processar os resultados da simulação.

A definição do sistema é realizada pelo projetista não caracterizando uma tarefa de alta carga computacional. A compilação do sistema consiste na tradução do mesmo para o formato utilizado pelo simulador. O tempo de compilação de um sistema, na maioria dos casos, é diretamente proporcional ao número de portas lógicas do mesmo [MEY85]. A simulação do sistema é, normalmente, a etapa mais demorada de todo o processo. O tempo de simulação cresce com o quadrado do número de portas lógicas de um sistema [MOT86].

A contribuição proporcional, em tempo, de cada etapa do processo depende, basicamente, do tamanho do sistema digital em análise. Para o final da década de 80 os sistemas digitais podem ser classificados de acordo com a tabela 1.1 [SMI86], considerando-se placas e circuitos

integrados de aplicação específica ( ASIC ).

Tabela 1.1 Classificação dos sistemas digitais

| TAMANHO DO SISTEMA | NÚMERO DE PORTAS LÓGICAS |
|---|---|
| pequeno | 2 000 |
| médio | 10 000 |
| grande | 50 000 |
| muito grande | 500 000 |

O efeito do tamanho do sistema na distribuição do tempo gasto em cada etapa do processo foi observada através da simulação de três sistemas diversos conforme tabela 1.2 [MOT86].

Tabela 1.2 Distribuição do tempo no processo de simulação

| TAMANHO DO SISTEMA | PERCENTAGEM DE TEMPO DISPENDIDA | | |
|---|---|---|---|
| | COMPILAÇÃO | SIMULAÇÃO | PROCESSAMENTO |
| pequeno | 30 | 60 | 10 |
| médio | 10 | 87 | 3 |
| grande | < 1 | 99 | < 1 |

Para pequenos sistemas a compilação e o processamento de saída ocupam uma grande parcela do tempo total. Para grandes sistemas, todavia, o tempo de simulação torna-se dominante constituindo o gargalo do processo. Várias soluções podem ser utilizadas para resolver este problema. Entre elas:

- hardware mais potente;
- software mais eficiente;

- arquiteturas especiais.

A abordagem tradicional para acelerar a simulação é o uso de computadores de grande porte. Estes apresentam uma elevada velocidade porém o alto custo de sua utilização torna-os, por vezes, proibitivos [WYA83].

A solução de aumentar a eficiência do software pode levar a resultados surpreendentes [BLO87]. A ordem de grandeza do ganho de velocidade, entretanto, não é suficiente para resolução do problema. Programar o algoritmo de simulação em assembler, por exemplo, aumenta três vezes a velocidade de simulação [PAS85].

O uso de arquiteturas especiais na resolução de problemas de PAC tem-se mostrado uma solução viável e popular [BLA84]. O desenvolvimento de uma Arquitetura Especial para Simulação Lógica, doravante denominada AESL, é vantajosa graças a duas condições presentes nos algoritmos de simulação [MOT85]:

- o núcleo do algoritmo é estável;
- 99 % da carga computacional está concentrada numa pequena parte do algoritmo.

O segundo item possibilita que 99 unidades de tempo, em cada 100, tendam a zero restando somente uma. O ganho de velocidade, neste caso, é de 100 vezes.

Neste trabalho pretendemos definir uma AESL flexível e de fácil implementação, considerando a tecnologia disponível no país. Esta AESL deve simular sistemas digitais cujas dimensões estejam entre grande e muito grande conforme apresentado na tabela 1.1. Não é nosso objetivo desenvolver esta arquitetura buscando o maior desempenho tecnicamente possível visto que entraríamos em conflito com nossas

premissas básicas.

Inicialmente faremos uma breve revisão sobre as técnicas e os modelos utilizados em simulação lógica. Uma taxonomia para AESL será definida e utilizada como base na análise das diversas propostas de AESL relatadas na literatura. Apresentatemos, então, nossa proposta que será analisada e reformulada a partir dos dados obtidos na simulação de um modelo criado para a mesma. Por fim, traçaremos um quadro comparativo entre a nossa proposta e outras arquiteturas utilizadas em simulação lógica.

# 2 TÉCNICAS DE SIMULAÇÃO

As técnicas de simulação podem ser divididas em dois grupos principais: a simulação baseada em código compilado, o mais antigo dos métodos, e a simulação baseada em tabelas, o mais utilizado atualmente.

## 2.1 SIMULAÇÃO BASEADA EM CÓDIGO COMPILADO

Nesta técnica a descrição do sistema digital é traduzida para um código executável por um computador. A estrutura do simulador é a da figura 2.1 [WAG84a]

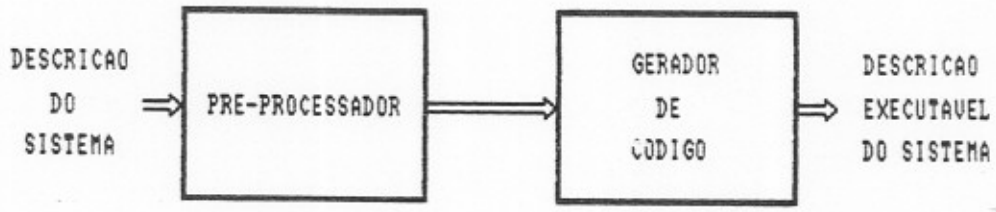



Figura 2.1 Estrutura de um simulador de código compilado

O pré-processador identifica os laços de realimentação do sistema dividindo-o em níveis a partir das entradas primárias. Cada porta recebe o número do nível a que pertence. Os elementos são ordenados de acordo com o número de seu nível.

O gerador de código converte a descrição ordenada do sistema em instruções do computador hospedeiro. O sistema pode ser executado como um programa qualquer do hospedeiro. Esta técnica de simulação é utilizada com o modelo de atraso de transporte zero ou unitário.

Na simulação com atraso unitário os valores lógicos das saídas dos elementos do sistema são armazenados,

redundantemente, em dois bancos de memória distintos: A e B. Num instante de tempo t todos os valores de entrada dos elementos são lidos no banco A enquanto as transições de saída são escritas no banco B. No tempo t + 1 os bancos são intercambiados atualizando os valores de saída de todos os elementos avaliados no instante anterior.

## 2.2 SIMULAÇÃO BASEADA EM TABELAS E ORIENTADA A EVENTOS

Esta técnica baseia-se numa propriedade apresentada pelos sistemas digitais: o seu baixo grau de atividade, ou seja, o número de elementos ativos, a cada instante de tempo, é pequeno [WAG84a] [WON86]. Um elemento é dito ativo quando ocorrer uma troca de valor lógico em pelo menos uma de suas entradas. Pode-se melhorar o desempenho da simulação avaliando-se somente estes elementos. É o conceito de simulação orientada a eventos, denominado avaliação seletiva ('selective trace' [SZY75]), segundo o qual um elemento somente necessita ser simulado quando ocorreu um evento em uma de suas entradas. Entende-se por evento a troca de valor de um sinal.

### 2.2.1 Descrição do sistema

A aplicação de técnicas de avaliação seletiva requer uma estrutura de dados especial, baseada em tabelas e suas inter-relações, que contém a descrição dos elementos e sua interconexões [D'A85]. Para um elemento i, com uma única saída, a descrição estrutural de um sistema deve conter:

- lista de conexões de saída de i (lista de 'fan-out' ou lista dos elementos cujas entradas estão conectadas na saída de i);

- lista de conexões de entrada de i (lista de "fan-in" ou lista dos elementos cujas saídas estão conectadas nas entradas de i).

Ao contrário da simulação baseada em código compilado, existe aqui uma separação entre a descrição estrutural do sistema e os procedimentos de avaliação dos elementos [WAG84a].

### 2.2.2 Fluxo do tempo

O mecanismo de fluxo de tempo é baseado na programação e efetivação de eventos [WAG84a]. A estrutura deste mecanismo depende do modelo de atrasos utilizado pelo simulador.

#### 2.2.2.1 Atraso zero

A utilização de atraso zero faz com que, a cada conjunto de valores lógicos de entrada, o sistema forneça, imediatamente, um único conjunto de valores de saída. A figura 2.2 apresenta um algoritmo utilizado com atraso zero, considerando que não há realimentações no sistema e aplicando-se a técnica de avaliação seletiva.

```
Inicializar fila de propagação
Para cada sinal de entrada que variou
Faça Para cada elemento i da lista de conexões
                de saída do sinal
     Faça Insira i na fila de propagação

Enquanto a fila de propagação não estiver vazia
Faça Para cada elemento i da fila de propagação
     Faça Avalie o elemento i
          Se saída variou
          Então Para cada elemento j da lista de conexões
                          de saída de i
                Faça Insira j na fila de propagação
```

Figura 2.2 Simulação com atraso zero

Uma desvantagem deste algoritmo é a possibilidade de um elemento ser avaliado diversas vezes. Este fato acontece sempre que um elemento é ativado por caminhos múltiplos a partir dos sinais de entrada. A solução para este problema é associar níveis aos elementos da mesma forma que numa simulação baseada em código compilado. A avaliação de um elemento só é efetuada quando o nível associado ao mesmo é igual ao nível corrente, conforme mostrado na figura 2.3. Desta forma cada elemento é avaliado, no máximo, uma vez.

```
Inicializar fila de propagação
Inicializar nível corrente
Para cada sinal de entrada que variou
Faça Para cada elemento i da lista de conexões
                de saída do sinal
     Faça Insira i na fila de propagação

Enquanto a fila de propagação não estiver vazia
Faça Para cada elemento i da fila de propagação
     Faça Se nível associado ao elemento é o corrente
          Então Avalie o elemento i
                Se saída variou
                Então Para cada elemento j da lista de
                                conexões de saída de i
                Faça Insira j na fila de propagação
     Incremente nível corrente
```

Figura 2.3 Simulação com atraso zero e associação de níveis

Conceitos de simulação a nível de transferência entre registradores podem ser utilizados, juntamente com o atraso zero, na simulação de sistema digitais síncronos. É uma simulação mista onde as portas lógicas possuem atraso zero e os registradores, carregados por um relógio, atraso unitário. Em cada ciclo de relógio os eventos gerados nas saídas dos registradores propagam-se, através dos elementos combinacionais, até as entradas dos registradores. No final do ciclo de relógio as saídas dos registradores são atualizadas, de acordo com suas entradas, gerando-se os eventos para o próximo ciclo.

### 2.2.2.2 Atraso unitário

Neste modelo todas os elementos possuem atraso unitário. O gerenciamento dos eventos é efetuado através de duas listas: La e Lb [BRE76]. Cada lista é composta por pares (i,s(i)) onde i é o identificador do elemento e s(i) é o valor da nova saída calculada para o elemento. A manipulação das duas listas implementa o mecanismo de fluxo de tempo conforme mostrado na figura 2.4 onde é utilizada a técnica de avaliação seletiva.

```
Se La não está vazia
Então Para cada par (i,s(i)) de La
      Faça Atualizar o valor da saída do elemento com s(i)
           Para cada elemento j da lista de conexões
                    de saída de i
           Faça Avaliar elemento j
                Se saída de j variou
                Então Inserir em Lb o par (j,s(j))
      La <- Lb
      Lb <- ∅
      Incrementar tempo de simulação
```

Figura 2.4 Simulação com atraso unitário

### 2.2.2.3 Atraso atribuível

Este modelo permite que sejam atribuídos atrasos distintos aos diversos elementos que constituem o sistema. Quando um elemento i é avaliado em um tempo t e sua saída muda de valor este novo valor somente deve ser atribuído à saída no tempo t+dt(i) onde dt(i) é o atraso associado ao elemento i [D'A85]. Este processo é implementado através da programação de um evento para o elemento i que deverá ser efetivado no tempo t+dt(i). A figura 2.5 exemplifica este mecanismo utilizando, também, a técnica de avaliação seletiva.

```
Avançar tempo t até que haja algum evento programado
Para cada evento programado
Faça Atualizar saída do elemento i associado ao evento
     Para cada elemento j da lista de conexões
                de saída de i
     Faça Avaliar elemento j
          Se saída de j variou
          Então Programar evento para j no tempo t+dt(j)
```

Figura 2.5 Simulação com atraso atribuível

O algoritmo da figura é chamado de passo único, visto que após a efetivação de cada evento são avaliados os elementos ativos decorrentes do mesmo. A ativação múltipla de um elemento, devido à existência de vários eventos afetando suas entradas, pode levar ao cancelamento de eventos. Um algoritmo de dois passos não possui este inconveniente. No primeiro passo são efetivados todos os eventos e uma fila de propagação recebe os elementos ativos. No segundo passo são avaliados os elementos da fila e programados os eventos necessários. Os algoritmos de passo único são mais eficientes, desde que sejam respeitadas duas condições:

- o número de reavaliações é pequeno;
- a sobrecarga causada pelas reavaliações é desprezível.

No caso de elementos lógicos simples, com poucas entradas e de rápida avaliação, estas premissas são verdadeiras.

Durante o processo de simulação uma média de aproximadamente 23 % do tempo total é dispendido em manipulações na lista de eventos [WON86]. É importante que exista um mecanismo eficiente para que este percentual não cresça em demasia, comprometendo o desempenho de todo o simulador.

As estruturas de dados utilizadas com este fim são as listas encadeadas de eventos e as chamadas filas de tempo ("time-queues" [SZY75]). Os eventos são armazenados em listas encadeadas. A cada valor de tempo corresponde uma lista. As listas podem ser novamente encadeadas formando uma lista de eventos única conforme figura 2.6.

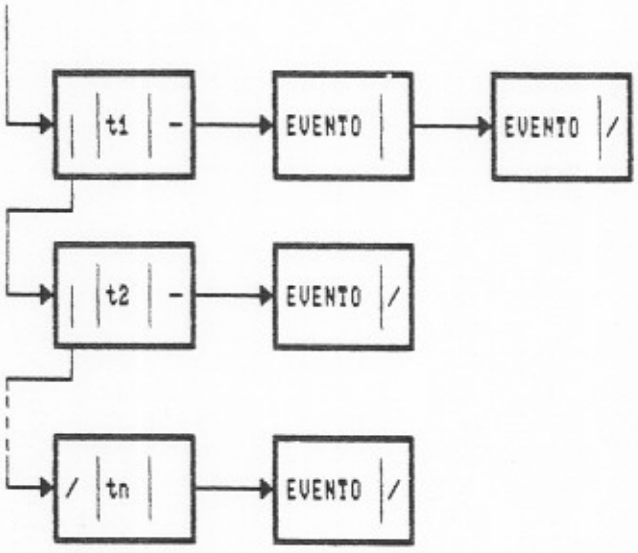



Figura 2.6 Estrutura de uma lista encadeada de eventos

O método mais utilizado é acessar as listas a partir de uma fila de tempo, estrutura circular implementada como um vetor de ponteiros, em que cada posição corresponde a uma fração igual de tempo conforme mostrado na figura 2.7.

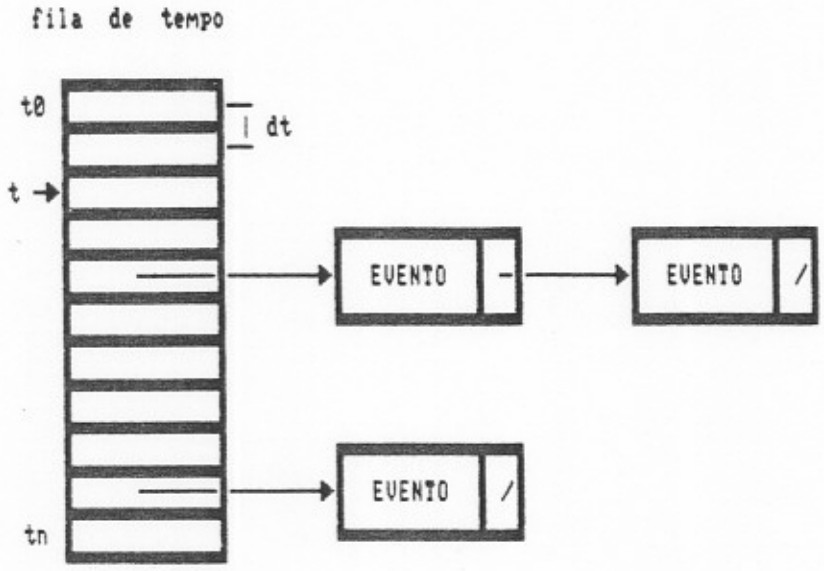



Figura 2.7 Estrutura de uma fila de tempo

O acesso à fila de tempo é feito através de um mecanismo de mapeamento de tempo ("time mapping" [ULR69]). A relação entre o tempo corrente de simulação e uma determinada posição na fila de tempo é dada por $T = C * n * dt + t$ onde:

T = tempo corrente de simulação;
C = número de vezes em que a fila de tempo completou um ciclo (t=tn);
n = número de posições na fila de tempo;
dt = incremento de tempo entre duas posições;
t = deslocamento dentro da fila de tempo.

Há duas abordagens quanto ao dimensionamento e controle de ciclos da fila de tempo [WAG84a].

Na primeira, a fila de tempo recebe a denominação de roda de tempo ("timing wheel" [BRE76]) existindo uma lista de excedentes onde são inseridos os eventos cujo tempo ultrapassa o abrangido pela roda de tempo. A cada novo ciclo a lista de excedentes é transposta para a roda de tempo e o deslocamento dentro da mesma reinicializado em t0.

Na segunda temos o chamado laço $\Delta T$ ('$\Delta T$-loop' [ULR69]) onde o tamanho da fila de tempo é determinado pela relação $n * dt > td$, sendo td o maior atraso existente entre os elementos do sistema. O valor ótimo para dt é determinado pelo máximo divisor comum do conjunto de atrasos dos elementos do sistema. Desta forma não é necessária uma lista de excedentes, visto que nenhum evento programado possuirá tempo superior ao abrangido pela lista.

### 2.2.3 Avaliação de elementos

A avaliação de um elemento lógico pode ser feita tendo por base dois métodos: subrotinas ou tabelas [D'A85].

A avaliação por subrotinas associa a cada tipo de elemento lógico uma subrotina cujos parâmetros de entrada são os valores das entradas do elemento e cuja saída é o valor calculado para a saída do elemento. Várias técnicas podem ser empregadas para melhorar o desempenho destas subrotinas [WAG84a].

A avaliação por tabelas utiliza apenas uma rotina cujos parâmetros de entrada (tipo do elemento e estado das entradas) formam um vetor que endereça a tabela verdade do elemento fornecendo o novo valor da saída. É um método rápido e bastante prático quando é utilizada lógica de múltiplos valores. As tabelas podem ser expandidas de forma a incluir funções mais complexas como o gerenciamento dos eventos [ULR72].

Em ambos os métodos é necessário o estado das entradas do elemento. Estes valores são obtidos através da lista de conexões de entrada do elemento ou armazenando o estado das entradas juntamente com o elemento.

# 3 MODELOS PARA SIMULAÇÃO

A precisão dos resultados obtidos numa simulação está diretamente relacionada com os modelos utilizados para os elementos lógicos. O fator determinante na escolha de um modelo é o algoritmo de simulação. A complexidade do modelo repercute negativamente no tempo de desenvolvimento e na eficiência obtida para um algoritmo.

## 3.1 ELEMENTOS LÓGICOS

Os elementos lógicos de um simulador são as primitivas utilizadas na modelagem de um circuito.

Os elementos básicos, presentes em todos os simuladores, são as portas lógicas tradicionais: AND; NAND; OR; NOR; NOT; BUFFER. Alguns simuladores incluem portas mais especializadas como XOR e NXOR [WAG87] e elementos tristate [WAG87] [CIR83] de forma a permitir a modelagem de barramentos.

A crescente utilização de tecnologias MOS aproximou o simulador lógico dos simuladores de chaves [BRY80] [BRY84] com a introdução, nas primitivas, dos transistores (NMOS, PMOS e CMOS) [WAG87] [CIR83] [SAN83]. Os resistores (ou transistores) de pull também são elementos utilizados como primitivas.

Vários simuladores incluem entre suas primitivas elementos de maior complexidade como ROMs [CAS78], elementos de memória RTL [SAN83], elementos funcionais definidos pelo usuário, etc.

Os flip-flops normalmente são modelados através de portas lógicas. Alguns autores [MIC87] consideram mais eficiente sua utilização como primitiva.

## 3.2 ATRASOS

O atributo principal na modelagem de um elemento lógico é seu atraso. O algoritmo de simulação é fruto das características adotadas para os atrasos dos elementos. Dentre estas características selecionamos as mais importantes que aparecem de forma combinada na maioria dos simuladores.

### 3.2.1 Atraso de transporte

Neste modelo cada elemento é formado por uma função lógica pura acrescida de um atraso na efetivação de seus valores de saída. Este atraso, válido para todos os elementos do sistema, pode ser de três tipos: zero; unitário ou atribuível.

O atraso zero não permite a verificação do "timing" do sistema e consequentemente a detecção de "hazards" e "races". É o modelo mais simples sendo de grande valia na verificação lógica de sistemas combinacionais e sistemas sequenciais síncronos. É utilizado, normalmente, em simuladores de código compilado [WAG84a].

O atraso unitário permite uma verificação grosseira do "timing" do sistema.

O atraso atribuível facilita a análise de "timing" devido à possibilidade de se atribuir livremente os atrasos dos elementos.

O modelo de atraso de transporte exige (exceto para atraso zero ou unitário) um simulador baseado em tabelas. Para cada futura transição na saída é programado um evento. Um elemento pode possuir vários eventos futuros programados. Não é necessário um mecanismo de cancelamento de eventos [WAG85].

### 3.2.2 Atraso de transição

Este modelo é mais realista que o de atraso de transporte. Neste caso cada porta possui dois atrasos de transporte que são utilizados de acordo com a transição do sinal de saída. São consideradas transições de subida (0 -> 1) e descida (1 -> 0). A combinação de diferentes atrasos de tranporte com variações na entrada pode levar à necessidade de cancelamento de eventos [WAG85].

### 3.2.3 Atraso com ambiguidade

O uso de ambiguidade incrementa a precisão do modelo de um elemento. A necessidade do uso desta característica advém do fato que portas de um mesmo tipo podem possuir distintos atrasos de propagação. Consequentemente, o atraso de uma porta deve ser representado por um valor mínimo e uma região de ambiguidade [SZY70]. Esta região representa uma incerteza sobre o comportamento do elemento e geralmente tem associado o valor indeterminado (X) [CHA71]. A análise do timing do sistema torna-se mais refinada apesar dos resultados pessimistas que pode apresentar [WAG84a]. Uma porta pode possuir vários eventos programados e o cancelamento de eventos pode ocorrer atingindo sempre o evento de maior tempo associado [WAG85].

### 3.2.4 Atraso inercial

O atraso inercial modela a característica de resposta em frequência de um elemento. A largura mínima necessária para que um pulso na entrada de um elemento se propague até sua saída é o chamado atraso inercial do elemento [D'A85]. Atrasos inerciais também podem estar associados às saídas dos elementos de forma a considerar as capacitâncias ali presentes [WAG85]. Num simulador que utilize somente dois valores lógicos há, no máximo, um evento pendente para cada elemento existindo a possibilidade de cancelamento [WAG85].

## 3.3 VALORES LÓGICOS

Os valores lógicos são utilizados para representar, no sistema em análise, as diversas condições de tensão e corrente dos sinais. O número de valores adotado, sua interpretação e seu inter-relacionamento varia enormemente de um simulador para outro.

Os primeiros simuladores utilizavam uma lógica de dois valores (0 e 1). Valores adicionais foram surgindo de forma a representar condições de erro, estados desconhecidos, transientes,etc. A simulação de lógica tristate levou à criação de um estado de alta impedância gerando um conjunto de valores (0,1,X,Z) largamente utilizado na implementação de simuladores lógicos [CIR83] [CAS78].

Com a utilização de tecnologias MOS na implementação de sistemas digitais a variável corrente passou a ser um parâmetro importante no modelo para simulação. O sinais digitais devem ser representados por pares (Vi,Sj) onde Vi indica o nível de tensão e Sj a carga elétrica disponível [HAY86]. O nível de tensão é geralmente representado por

três valores: 0 (nível baixo); 1 (nível alto) e X (indeterminado). O parâmetro carga elétrica representa os valores discretos de corrente (fornecida/absorvida por uma saída) ou carga (armazenada em uma capacitância) utilizadas pelo simulador. Quanto maior o número de valores de carga maior a precisão com que o comportamento do sistema é modelado. Um modelo com cinco valores de carga é mostrado na figura 3.1 [STE83].

| CARGA | SIGNIFICADO |
|---|---|
| E (External) | Entradas externas, VCC e GND |
| D (Driven) | Saída de porta (fonte de corrente) |
| R (Resistive) | Saída de porta (resistiva) |
| L (Large high impedance) | Informação armazenada em capacitância grande |
| Z (high impedance) | Informação armazenada em capacitância pequena |

Figura 3.1 Valores de carga de um simulador lógico

A simulação com múltiplos valores de carga implica em resultados pessimistas [FLA83]. Este fato pode ser contornado com a introdução de vários níveis de ambiguidade no conjunto de valores lógicos. Uma lógica de cinco estados ( 0 forte, 0 fraco, 1 forte, 1 fraco e alta impedância ) origina quinze valores lógicos conforme mostrado na figura 3.2 [FLA83].

Lógicas de múltiplos valores também são utilizadas na determinação de "hazards". A lógica de seis valores mostrada na figura 3.3 permite a análise de "hazards" estáticos [HAY86].

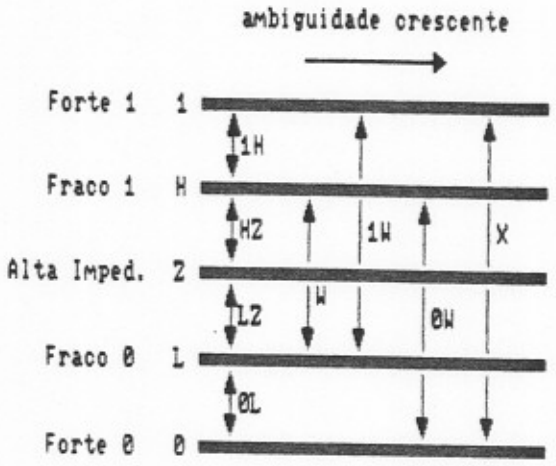



Figura 3.2 Valores lógicos para uma lógica de cinco estados

| VALOR | SIGNIFICADO |
|---|---|
| (0,0,0) | sinal estático em 0 |
| (1,1,1) | sinal estático em 1 |
| (0,X,1) | transição 0 -) 1 |
| (1,X,0) | transição 1 -) 0 |
| (0,X,0) | "hazard" estático em 0 |
| (1,X,1) | "hazard" estático em 1 |

Figura 3.3 Lógica de seis valores para análise de "hazards"

A análise de "hazards" estáticos e dinâmicos é feita utilizando-se uma lógica de oito valores conforme figura 3.4 [WAG84b].

| VALOR | SIGNIFICADO |
|---|---|
| 1 | sinal estático em 1 |
| 0 | sinal estático em 0 |
| ↑ | transição 0 -) 1 |
| ↓ | transição 1 -) 0 |
| 0* | "hazard" estático em 0 |
| 1* | "hazard" estático em 1 |
| ↑* | transição 0 -) 1 com "hazard" dinâmico |
| ↓* | transição 1 -) 0 com "hazard" dinâmico |

Figura 3.4 Lógica de oito valores para análise de "hazards"

O conjunto de valores lógicos adotados e o conjunto das operações sobre estes valores (definidas pelos elementos do sistema) definem uma álgebra. A definição de

valores lógicos e a construção de suas álgebras associadas foi formalizada em [HAY86]. A figura 3.5 apresenta a tabela do operador AND para a lógica definida na figura 3.3.

| AND | (0,0,0) | (1,1,1) | (0,X,1) | (1,X,0) | (0,X,0) | (1,X,1) |
|---|---|---|---|---|---|---|
| (0,0,0) | (0,0,0) | (0,0,0) | (0,0,0) | (0,0,0) | (0,0,0) | (0,0,0) |
| (1,1,1) | (0,0,0) | (1,1,1) | (0,X,1) | (1,X,0) | (0,X,0) | (1,X,1) |
| (0,X,1) | (0,0,0) | (0,X,1) | (0,X,1) | (0,X,0) | (0,X,0) | (0,X,1) |
| (1,X,0) | (0,0,0) | (1,X,0) | (0,X,0) | (1,X,0) | (0,X,0) | (1,X,0) |
| (0,X,0) | (0,0,0) | (0,X,0) | (0,X,0) | (0,X,0) | (0,X,0) | (0,X,0) |
| (1,X,1) | (0,0,0) | (1,X,1) | (0,X,1) | (1,X,0) | (0,X,0) | (1,X,1) |

Figura 3.5 Operador AND para uma lógica de seis valores

# 4 ARQUITETURAS ESPECIAIS PARA SIMULAÇÃO

O estudo das diversas propostas de AESL e a observação de suas características comuns permitiram-nos desenvolver uma taxonomia adequada para este tipo de máquina.

## 4.1 PROPOSTA DE UMA TAXONOMIA PARA AESL

O princípio básico de uma AESL é a exploração da concorrência existente no processo de simulação. A análise dos mecanismos de software e hardware que implementam esta concorrência é a base de nossa taxonomia apresentada na tabela 4.1.

Tabela 4.1 Taxonomia para AESL

| EXPLORAÇÃO DA CONCORRÊNCIA | PARTICIONAMENTO DO ALGORITMO | PROCESSAMENTO PIPELINE |
|---|---|---|
| | | PROCESSAMENTO DATA FLOW |
| | PARTICIONAMENTO DO SISTEMA | |

O particionamento do sistema pode ser caracterizado através de três aspectos conforme tabela 4.2. A taxonomia é complementada por alguns critérios auxiliares relativos às técnicas e modelos para simulação utilizados nas AESL.

Tabela 4.2 Características do particionamento do sistema

| | |
|---|---|
| SINCRONISMO | RELÓGIO GLOBAL |
| | RELÓGIO LOCAL |
| COMUNICAÇÃO ENTRE PROCESSADORES | REDE DE ROTEAMENTO |
| | BARRAMENTO COMPARTILHADO |
| PROCESSAMENTO DE SUBSISTEMA | UNIPROCESSADOR |
| | EXPLORAÇÃO DA CONCORRÊNCIA |

### 4.1.1 Particionamento do algoritmo

Particionar o algoritmo significa dividi-lo em etapas que são passíveis de execução concorrente. A atividade lógica em um sistema real envolve a propagação simultânea de sinais por diferentes caminhos. Este fato reflete-se na simulação pela existência de vários eventos programados para o mesmo tempo. O número de eventos simultâneos pode ser bastante elevado o que torna interessante a exploração desta concorrência potencial.

A figura 4.1 [ABR83] mostra o diagrama de precedência das diversas partes do algoritmo apresentado na figura 2.3 do capítulo 2. Neste diagrama todos os processos representados na mesma linha horizontal podem ser executados em paralelo. O particionamento do algoritmo pode ser implementado com arquiteturas que utilizam processamento pipeline ou data flow.

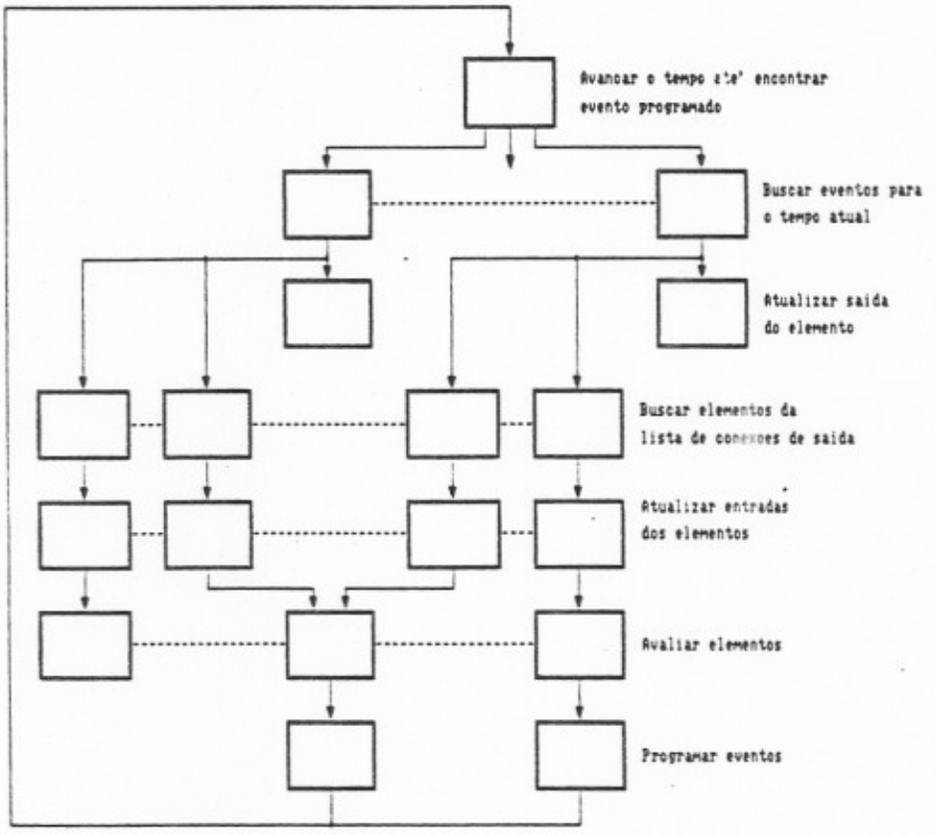



Figura 4.1 Concorrência num algoritmo para simulação

#### 4.1.1.1 Processamento pipeline

O processamento pipeline utiliza a técnica de decompor um processo sequencial em vários subprocessos. Cada subprocesso é executado num módulo autônomo e de forma concorrente com os demais. É o mesmo conceito utilizado em linhas de montagem industriais [RAM77]. Uma AESL típica com processamento pipeline é a da figura 4.2.

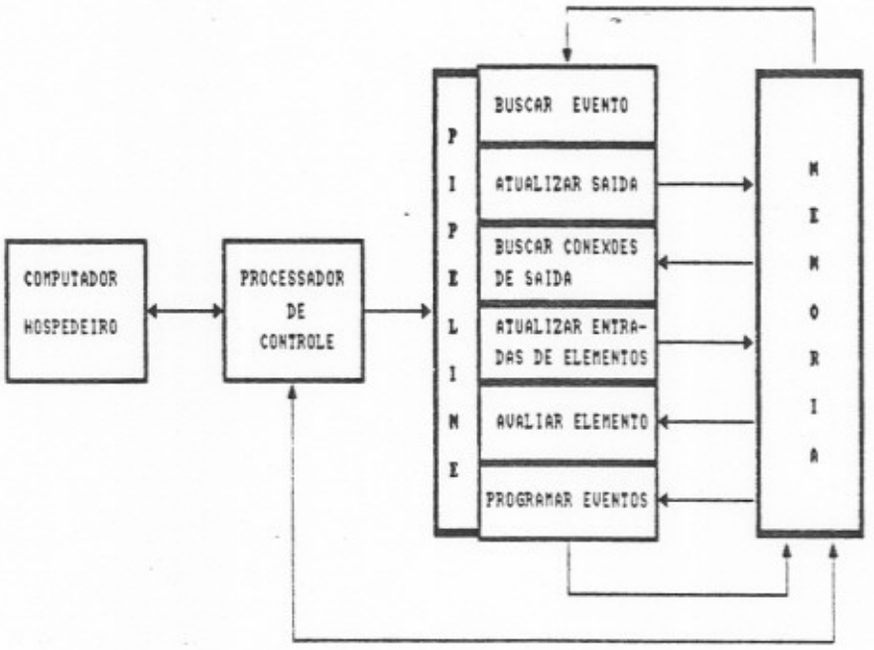



Figura 4.2 AESL com processamento pipeline

O pipeline é composto de seis estágios, correspondentes às distintas etapas do algoritmo. O relógio do pipeline é gerado pelo processador de controle, tendo em vista os diferentes tempos de processamento de cada estágio. O processador de controle tem por função:

- gerenciar o avanço do tempo de simulação;
- realizar a comunicação com o computador hospedeiro;
- controlar a transferência de dados entre estágios do pipeline;
- aplicar os vetores de entrada do sistema;
- monitorar os sinais de saída.

A memória contém a lista de eventos e a estrutura de dados que descreve o sistema. O acesso simultâneo pelos estágios do pipeline torna obrigatório o uso de uma memória multiporta ou a partição da memória de acordo com o tipo de dado (lista de conexões de saída, valores lógicos dos

nodos, etc).

O processamento pipeline também pode ser utilizado com simuladores de código compilado. Neste caso, o pipeline é alimentado por uma memória de instruções que contém a descrição do sistema.

4.1.1.2 Processamento data flow

O processamento data flow está baseado nas similaridades entre os algoritmos de simulação orientados a eventos e o funcionamento de uma máquina data flow.

Uma máquina data flow é um computador de programa armazenado em que o programa é a representação de um grafo data flow [AGE82]. Neste grafo dirigido os nodos representam as funções (instruções) e os arcos as dependências de dados entre funções conforme exemplo da figura 4.3.

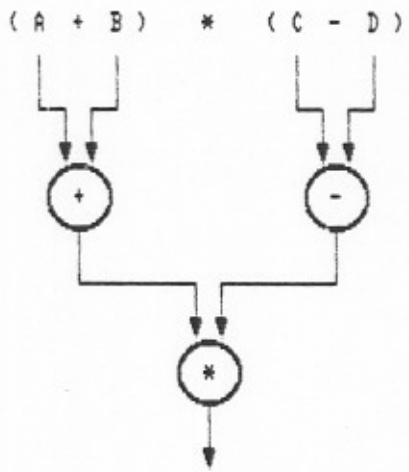



Figura 4.3 Grafo data flow para a expressão (A+B) * (C+D)

Um nodo só realiza sua função quando todos os seus operandos estão disponíveis. Uma máquina data flow real possui um número de processadores menor que o número de nodos

de um grafo típico. O mapeamento entre os dois é feito por compiladores, carregadores e software de operação. Há dois tipos de máquinas data flow: estáticas e dinâmicas. As estáticas só permitem que cada arco possua um operando. As dinâmicas associam um campo de 'tag' aos operandos contornando esta limitação e permitindo uma maior utilização de cada nodo [HWA84].

Um algoritmo de simulação pode produzir os mesmos resultados que uma máquina data flow estática desde que sejam observadas três condições [PAS85]:

- todas as entradas chegam ao nodo (ou elemento lógico) simultaneamente;
- o atraso nos nodos pode ser ajustado para satisfazer a condição anterior;
- sejam utilizados nodos nulos (função identidade) com atraso, de forma a ajustar o simulador para equações desbalanceadas

A figura 4.4 [PAS85] mostra a execução da equação desbalanceada ((A + A) * B) + 1) por uma máquina data flow e por um algoritmo de simulação. A inclusão de nodos nulos (NOP) evita que o algoritmo avalie os nodos, prematuramente, garantindo a chegada simultânea dos operandos.

As similaridades entre o modelo data flow e os algoritmos de simulação orientados a eventos estendem-se também às estruturas de dados e aos algoritmos básicos. As funções lógicas dos elementos correspondem às instruções que são codificadas nos pacotes data flow. Os níveis lógicos de entrada aos operandos e os elementos da lista de conexões de saída ao destino dos resultados. O tempo de simulação é análogo ao campo de 'tag' de um pacote data flow.

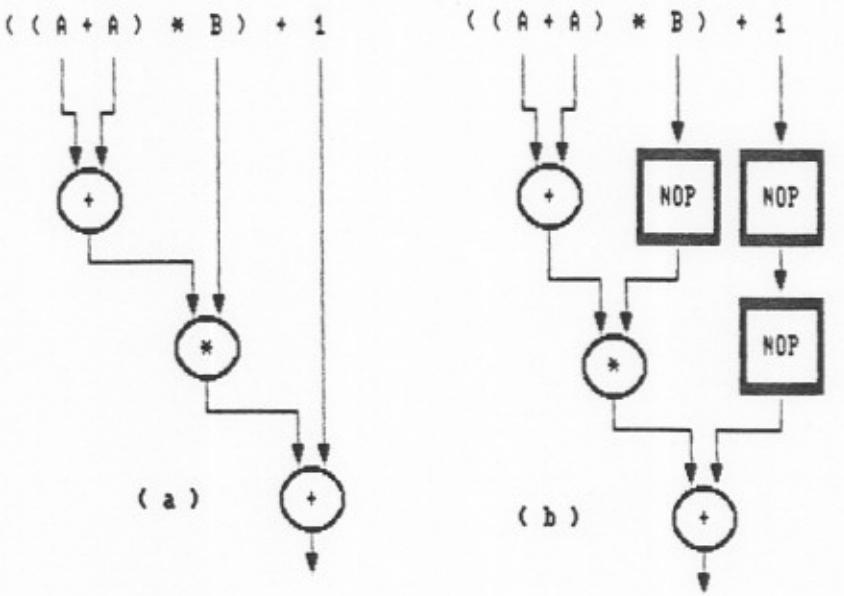



Figura 4.4 Equivalência entre (a) uma máquina data flow estática e (b) um algoritmo de simulação

Uma AESL típica com processamento data flow é mostrada na figura 4.5. Três unidades dispostas como um pipeline em anel implementam um data flow estático dedicado à simulação orientada a eventos.

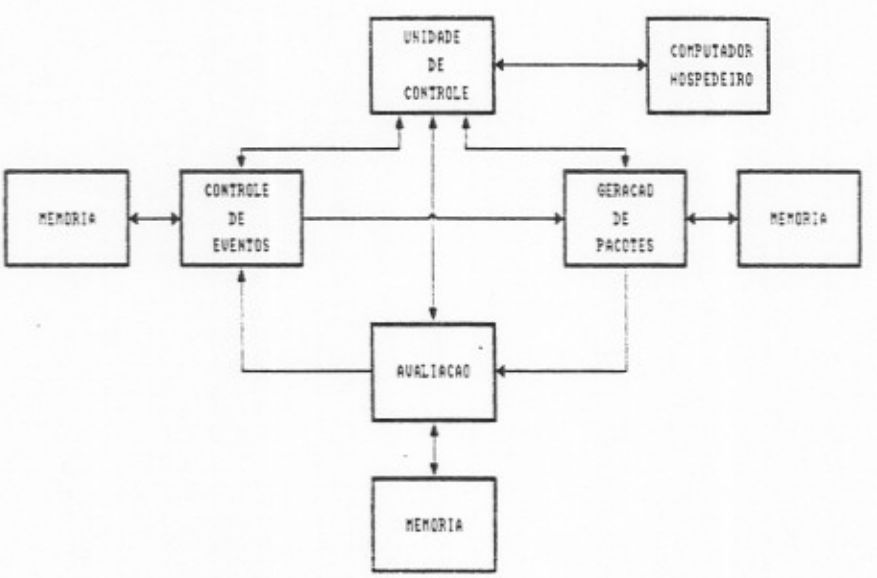



Figura 4.5 AESL com processamento data flow

A unidade de controle de eventos programa os eventos necessários relativos aos novos valores de saída e envia para a unidade de geração de pacotes os eventos para o tempo atual.

A unidade de geração de pacotes envia para a unidade de avaliação cada elemento afetado pelo evento ( elementos da lista de conexões de saída do elemento associado ao evento), sua função lógica e o valor de suas entradas.

A unidade de avaliação, a partir das funções lógicas e seus valores de entrada, envia para a unidade de controle de eventos os valores de saída calculados para os elementos.

O gerenciamento da simulação e da comunicação com o computador hospedeiro é feito pela unidade de controle.

### 4.1.2 Particionamento do sistema

O particionamento do sistema é uma forma de explorar o paralelismo físico que existe num sistema digital, baseando-se na divisão do mesmo em subsistemas.

A ocorrência de eventos concorrentes durante a simulação deve-se à maneira com que os sinais elétricos se propagam num sistema real. Esta propagação ocorre simultaneamente por diversos caminhos fazendo com que vários elementos sejam ativados ao mesmo tempo. O tempo total de simulação pode ser reduzido associando-se um subsistema a cada processador. O limite desta operação é alcançado quando o subsistema possui um único elemento.

O particionamento do sistema deve ser feito de forma a distribuir a atividade de simulação o mais

equitativamente possível entre os processadores, mantendo mínimas as necessidades de comunicação entre os mesmos [AGR86]. Esta tarefa pode ser executada manualmente ou por algoritmos heurísticos [LEV82] [AGR86]. Estes algoritmos utilizam, basicamente, a mesma filosofia. O sistema é percorrido, das entradas primárias para as saídas, formando-se vetores de elementos lógicos, atribuídos às diversas partições. Em cada vetor o elemento i+1 é obtido na lista de conexões de saída do elemento i. Os elementos que não foram associados a nenhum vetor são colocados no vetor que possui um elemento pertencente a sua lista de conexões de entrada. As partições são equalizadas tranferindo-se vetores entre elas. O desbalanceamento entre as partições gera uma diferença no tempo de simulação necessário para cada subsistema. Ao analisarmos o sincronismo no particionamento do sistema consideraremos os problemas que podem advir desta diferença de tempo.

Uma AESL que utilize esta forma de exploração de concorrência pode ser analisada sob três aspectos:

- sincronismo;
- comunicação entre processadores;
- processamento de subsistema.

#### 4.1.2.1 Sincronismo

A simulação de cada subsistema numa AESL corresponde a um processo num ambiente multiprocessador. A execução do mesmo gera eventos que devem ser enviados a outros processos. O sincronismo entre processos é a forma de ordenamento destes eventos que garante a execução correta de todo o sistema. Numa AESL o controle do tempo de simulação pode implementar o sincronismo através de duas abordagens: relógio local e relógio global.

A utilização do relógio global obriga todos os processadores a operarem sincronamente. O controle do tempo é feito de forma centralizada. Cada processador só inicia a execução das atividades do próximo instante de tempo após receber a habilitação de um controlador de tempo. É a forma mais simples de obtenção de sincronismo apresentando, porém, alguns problemas de desempenho,conforme mostrado na figura 4.6.

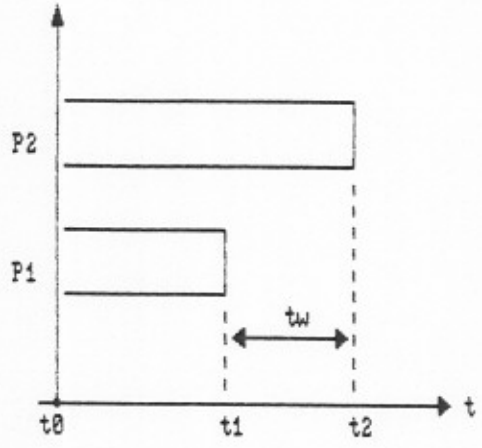



Figura 4.6 Sincronismo com relógio global

Para uma AESL com dois processadores (P1 e P2) num instante de tempo t0 há um certo número de eventos que devem ser efetivados. Suponha que os processadores P1 e P2 finalizam suas atividades nos tempos t1 e t2, respectivamente, onde t2 ) t1. Podemos, então, afirmar que:

- o tempo t2 é um ponto de sincronismo do sistema, isto é, o controlador de tempo pode incrementar o tempo de simulação;
- no intervalo de tempo tw o processador P1 nada executa, diminuindo o desempenho global do sistema.

O aproveitamento dos processadores pode ser melhorado com uma política de relógio local. Nesta abordagem os processadores operam de forma assíncrona. Os pontos de

sincronismo do sistema são implementados por mecanismos especiais que funcionam, normalmente, com características distribuídas.

Cada processador possui um relógio local. Diferentes processadores podem estar em instantes diversos no tempo de simulação. Este é o conceito de tempo virtual para um sistema distribuído [JEF85] segundo o qual cada processo é executado livremente até a ocorrência de um conflito com outro processo. A chegada de uma mensagem de evento cujo valor de tempo associado é menor que o tempo virtual local caracteriza um conflito. Todas as mensagens enviadas para outros processos são armazenadas localmente a partir do chamado tempo virtual global que é o menor valor de tempo virtual local entre todos os processos. Um conflito é resolvido retornando-se o tempo virtual local até seu valor anterior mais próximo ao conflito. Todas as mensagens enviadas a partir deste tempo são canceladas com o envio das chamadas "antimensagens". O processo pode, então, ser retomado.

#### 4.1.2.2 Comunicação entre processadores

A principal característica de um sistema multiprocessador clássico é a possibilidade de cada processador compartilhar módulos de memória principal e, possivelmente, dispositivos de entrada/saída [HWA84]. Esta capacidade de compartilhamento é possível graças à existência de uma estrutura de comunicação entre os diversos componentes do sistema. Uma AESL utiliza esta estrutura para transferir dados entre os processadores à medida que ocorrem variações nos sinais de interface entre subsistemas. A estrutura de comunicação também pode ser utilizada para informar aos processadores as variações dos sinais de entrada primários do sistema e enviar para o usuário os valores dos sinais de saída primários definidos para o sistema em simulação. Dois tipos

de estrutura são comumente empregados com esta finalidade: o barramento compartilhado e a rede de roteamento.

O barramento compartilhado interliga todos os processadores conforme apresentado na figura 4.7. Constitui-se no sistema mais simples e fácil de reconfigurar utilizado na interconexão de processadores.

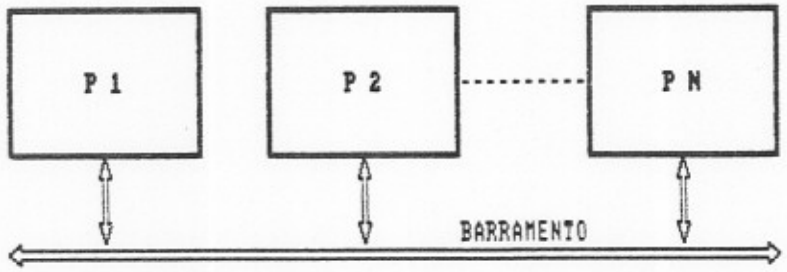



Figura 4.7 Estrutura de comunicação com barramento compartilhado

O barramento é , geralmente, um elemento passivo e as operações de tranferência são controladas pelas interfaces de cada processador com o barramento. Um mecanismo de contenção no acesso deve existir tendo em vista ser este um recurso compartilhado. O desempenho global do sistema fica limitado pela máxima taxa de transferência do barramento.

A simulação de um subsistema num processador conectado ao barramento envolve dois tempos distintos:

- tempo de processamento, tp;
- tempo de comunicação entre processadores, tc.

Admitindo que as atividades de processamento e comunicação são realizadas de forma concorrente podemos dizer que o desempenho máximo do sistema ( medido pela razão taxa de avaliações do multiprocessador / taxa de avaliações de um processador ) é obtido quando, para todo o conjunto de processadores, tp = tc [LEV82].

Aumentar o número de processadores além deste ponto pode diminuir tp mas, em contrapartida, eleva tc degradando o sistema como um todo.

Numa estrutura do tipo rede de roteamento vários processadores podem comunicar-se simultaneamente. A figura 4.8 apresenta a topologia desta estrutura.

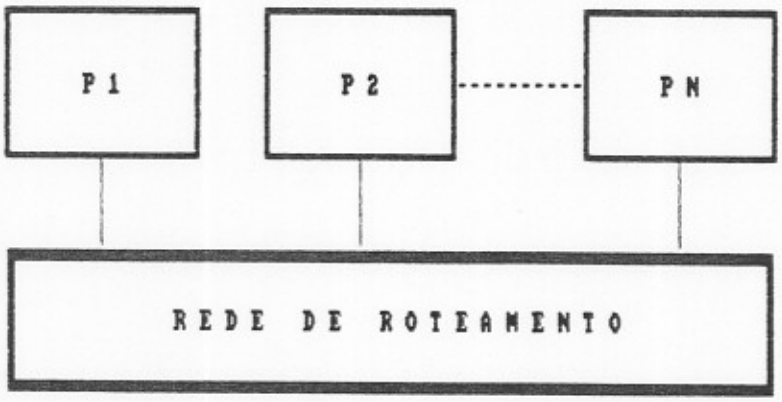



Figura 4.8 Estrutura de comunicação com rede de roteamento

A rede de roteamento gerencia as comunicações paralelas e resolve os conflitos que ocorrem quando há múltiplas requisições de acesso a um mesmo processador. A implementação de uma rede pode utilizar dois métodos: a chave "crossbar" e a rede multiestágios.

A chave "crossbar" é a estrutura de comunicação que apresenta maior banda passante. Entende-se por banda passante o número de requisições aceitas por unidade de tempo. Para um sistema com quatro processadores teríamos uma chave "crossbar" como a da figura 4.9. A principal desvantagem da chave "crossbar" é seu alto custo, visto que o número de elementos da chave cresce quadraticamente com o número de processadores.

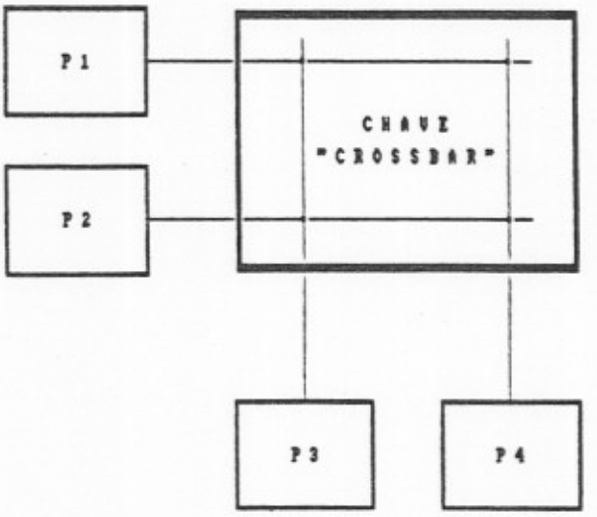



Figura 4.9 Topologia de uma chave "crossbar"

A rede multiestágio é uma estrutura modular que implementa uma rede de roteamento com um custo bem menor que a chave "crossbar". Este tipo de rede pode apresentar diversas topologias, adotadas de acordo com a aplicação específica do sistema. Uma rede multiestágio denominada "banyan" é apresentada na figura 4.10 [HWA84].

Esta rede conecta oito processadores baseando-se em chaves "crossbar" 2 X 2 e apresentando um custo proporcional a n log n. Uma desvantagem das redes multiestágio em relação à chave "crossbar" é sua característica de bloqueio [FRA81]. Dois processadores livres podem ser impedidos de interligação devido à utilização de algum estágio do caminho em outra conexão. As células de uma rede multiestágio podem possuir "buffers" fazendo com que a comunicação entre processadores seja feita concorrentemente e de forma pipeline.

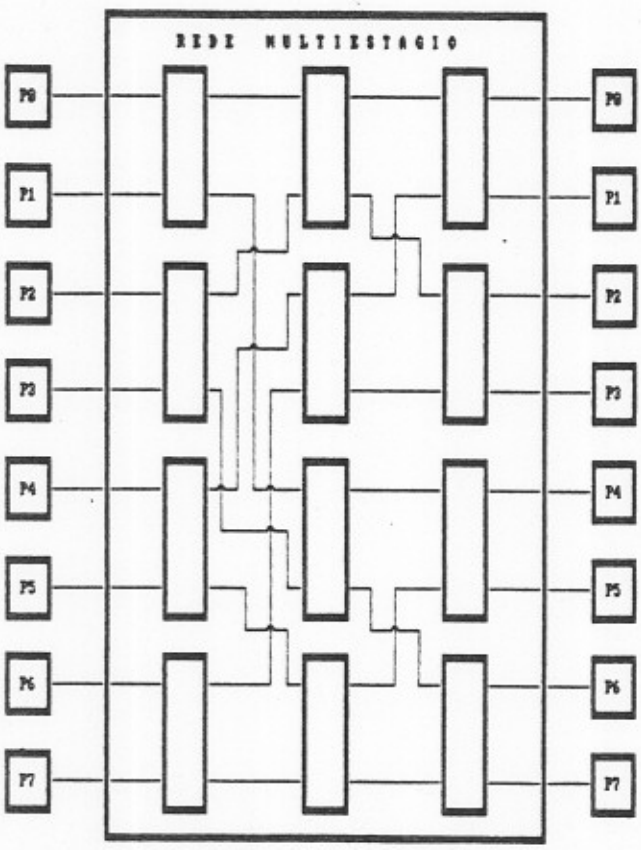



Figura 4.10 Topologia de uma rede multiestágio "banyan"

4.1.2.3 Processamento de subsistema

A simulação de cada subsistema, neste tipo de AESL, é atribuída a um processador. O modo pelo qual este processador realiza a simulação é chamado forma de processamento.

O emprego de um único processador executando um algoritmo sequencial de simulação, idêntico em todos os seus pares, caracteriza o uso de um uniprocessador. Este pode ser implementado de três formas distintas: em hardware, microprogramado ou por um microprocessador.

A maior velocidade é obtida através da implementação em hardware. O alto custo e a baixa flexibilidade são as desvantagens desta técnica. A microprogramação torna o

processamento de subsistema mais independente dos modelos e técnicas utilizadas na simulação, sendo uma solução intermediária relativamente à velocidade e custo. Os microprocessadores, por suas características de uso geral, apresentam baixo custo e alta flexibilidade. Seu desempenho, entretanto, é o menor dos três métodos.

Outra abordagem possível é a aplicação, num subsistema, das mesmas técnicas de exploração de concorrência utilizadas a nível de sistema. Este ponto de recursividade da taxonomia permite a definição de AESL's cujo limite é o próprio sistema físico. Na prática, a recursividade fica limitada a um nível.

### 4.1.3 Critérios auxiliares

A concepção de uma AESL, assim como a criação de um simulador "em software", é grandemente influenciada pelas técnicas e modelos adotados na simulação ( ver capítulos 2 e 3 ).

#### 4.1.3.1 Descrição do sistema

A descrição do sistema digital pode ser feita de duas formas: código compilado e tabelas.

O uso de código compilado permite a aplicação direta de técnicas pipeline, largamente utilizadas em computadores comerciais [RAM77]. O particionamento do sistema também é uma abordagem que pode ser utilizada neste caso.

A descrição por tabelas, graças a sua grande flexibilidade, permite que seja aplicada qualquer método de exploração de concorrência.

4.1.3.2 Fluxo do tempo

O mecanismo de avanço do tempo e a organização da lista de eventos determinam a forma utilizada pela AESL na modelagem do fluxo do tempo.

O avanço do tempo pode ser feito de forma incremental ou por evento. No avanço incremental o tempo flui em passos de tamanho fixo, independentemente da existência de eventos associados aos diversos instantes do mesmo. Este método é utilizado, obrigatóriamente, nas AESL cuja descrição da rede é feita em código compilado. No avanço por evento o novo valor do tempo é o associado ao próximo evento a ser processado. É um método mais eficiente, graças ao melhor aproveitamento da reduzida taxa de atividade do sistema.

A lista de eventos, quando existente, está condicionada à forma de exploração da concorrência na AESL em questão. O particionamento do algoritmo, normalmente, implica numa lista de eventos única. O particionamento do sistema induz à criação de uma lista de eventos por processador o que corresponde a uma lista por subsistema. A utilização de lista única, neste caso, é possível mas desaconselhável visto que aumenta a carga da estrutura de comunicação. Uma AESL sem mecanismos de avaliação seletiva não necessita de uma lista de eventos.

4.1.3.3 Avaliação dos elementos

A avaliação dos elementos lógicos pode ser feita através de tabelas específicas de forma sequencial ou concorrente. As rotinas são utilizadas, principalmente, para elementos mais complexos como módulos funcionais. As tabelas são, geralmente, carregadas em memória RAM. O índice de acesso a cada tabela é formado a partir da função lógica do

elemento.

A avaliação sequencial utiliza somente uma unidade com este fim para todos os elementos do sistema.

A avaliação concorrente possui múltiplas unidades que podem estar associadas a processadores ( particionamento do circuito ) ou integradas numa unidade de avaliação de alto desempenho ( particionamento do algoritmo ).

#### 4.1.3.4 Modelo adotado

O modelo que a AESL emprega para descrever a estrutura e o comportamento do sistema pode ser caracterizado por:

- tipos de elementos lógicos disponíveis;
- atrasos associados aos elementos;
- valores lógicos empregados.

Uma análise destes itens é apresentada no capítulo 3.

## 4.2 ANÁLISE DAS PROPOSTAS DE AESL

A taxonomia apresentada anteriormente pode ser aplicada no estudo das diversas propostas de AESL relatadas na literatura. Um resumo das características de cada arquitetura é apresentado nas tabelas 4.4 e 4.5 do item 4.2.9. A proposição de uma nova AESL será baseada nas conclusões desta análise.

### 4.2.1 Uma arquitetura para simulação de lógica digital [BAR80]

Esta proposta de AESL implementa um simulador, baseado em tabelas e orientado a eventos, que utiliza técnicas de avaliação seletiva. A figura 4.11 apresenta o diagrama em blocos da mesma.

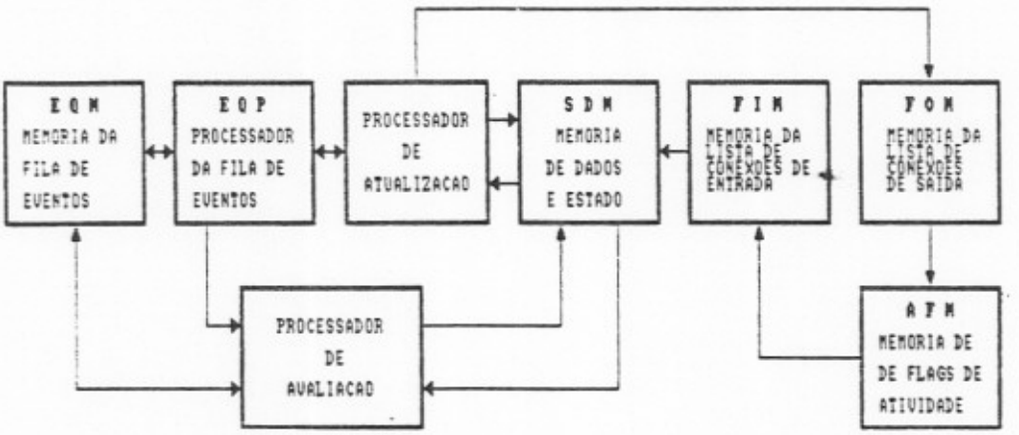



Figura 4.11 Diagrama em blocos da AESL

A descrição do sistema está dividida em três memórias ( SDM,FIM,FOM ) formadas por oito bancos. Uma fila para requisições de acesso está associada a cada banco. O mecanismo de avaliação seletiva é implementado pela AFM que associa um "flag" de atividade a cada elemento do sistema. Os eventos programados ficam na SDM ( 1º evento ) e na EQM ( eventos seguintes ). Cada elemento é identificado por um endereço, comum a todas as memórias, até um máximo de 32K elementos. Um algoritmo simplificado para descrever a operação desta AESL é apresentado na figura 4.12.

```
Carregue a descrição do sistema nas memórias
T <- 0
Enquanto T < Tmax
Faça  Avaliar
      Atualizar
      T <- T + 1
```

Figura 4.12 Algoritmo simplificado da AESL

Deste algoritmo podemos concluir que o avanço do tempo é feito de forma unitária e a simulação é realizada em dois passos: avaliação e atualização.

A exploração da concorrência é baseada no particionamento do algoritmo. Os dois passos utilizam processamento data flow caracterizado pelo fluxo assíncrono de dados e funções entre os diversos estágios da arquitetura.

O fluxo de dados e endereços na fase de avaliação é apresentado na figura 4.13.

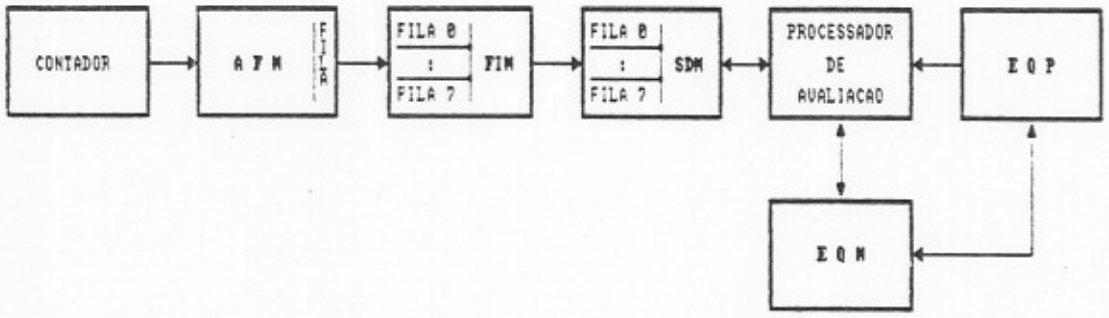



Figura 4.13 Fase de avaliação

Várias atividades são realizadas, de forma concorrente, durante esta fase. Um contador percorre toda AFM enviando para as filas da FIM o endereço dos elementos ativos. A FIM transfere para as filas da SDM os endereços correspondentes à lista de conexões de entrada de cada elemento. O processador de avaliação recebe da SDM os dados de cada elemento, juntamente com o valor de suas entradas, a fim de executar sua rotina específica de avaliação. Caso haja mudança em algum valor de saída é programado um evento na SDM ou, através do EQP, na EQM. O processo recomeça com a busca de um novo elemento na SDM. A fase de avaliação termina quando:

- o contador da AFM chegou ao fim;
- todas as filas estão vazias;

- o processador de avaliação está livre.

A fase de atualização possui um fluxo de dados e endereços conforme mostrado na figura 4.14.

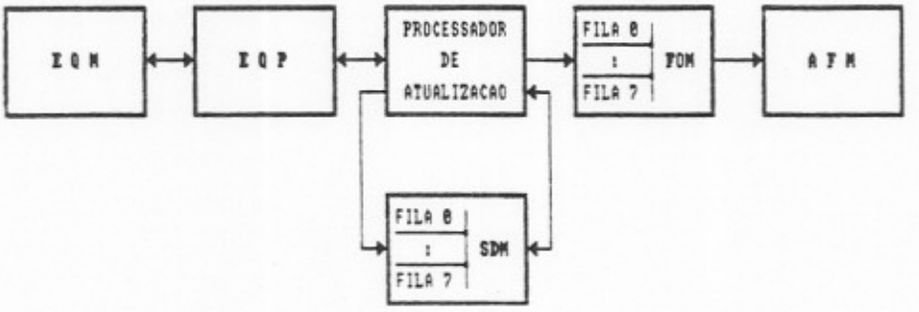



Figura 4.14 Fase de atualização

Nesta fase são efetivados os eventos e ativados os 'flags' de atividade dos elementos que sofreram variações em algum sinal de entrada. O processador de atualização é implementado por um pipeline de sete estágios. Todos os elementos da SDM passam pelo pipeline e tem seu estado atualizado. A efetivação de eventos presentes na SDM pode ativar o EQP para uma reorganização na lista associada ao elemento. O endereço de cada elemento i, cuja saída variou, é enviado para as filas da FOM. A FOM seta os 'flags' de atividade de todos elementos j da lista de conexões de saída de i.

A lista de eventos é única sendo que o primeiro evento de cada elemento fica armazenado juntamente com seus dados e estado. Os eventos seguintes estão encadeados no primeiro, residindo na EQM, e sendo manipulados pelo EQP. Os sinais de interface do sistema são tratados no início da fase de avaliação. Uma entrada que troca de valor é modelada setando os 'flags' na AFM dos elementos atingidos pela mudança.

Considerando que esta AESL não chegou a ser implementada o seu desempenho é apenas estimado a partir das seguintes premissas:

- o processador de avaliação é um microprocessador microprogramável, "bit slice" e de tecnologia ECL;
- o processador de atualização é implementado diretamente em hardware.

Para um tempo de avaliação médio de 5 uS por elemento o desempenho chega a aproximadamente 200 K elementos por segundo.

### 4.2.2 O YSE ( Yorktown Simulation Engine ) [DEN82]

O YSE é uma AESL concebida e implementada pela IBM para uso interno da companhia. A forma de exploração da concorrência adotada foi o particionamento do sistema. A figura 4.15 mostra a arquitetura básica do YSE.

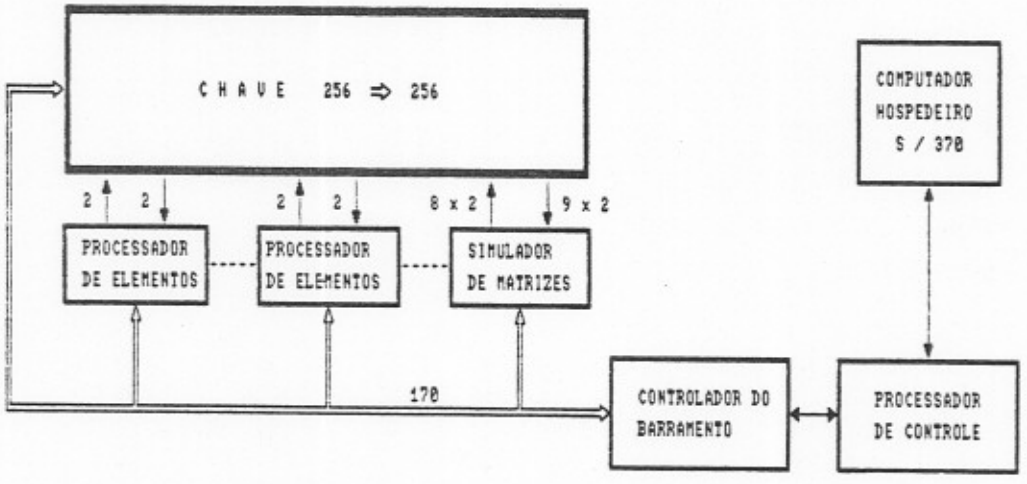



Figura 4.15 Topologia do YSE

O processador de controle gerencia a comunicação do YSE com o computador hospedeiro sendo baseado no microprocessador de 16 bits Z8000. A simulação pode ser feita de forma interativa. O barramento e seu controlador são utilizados para ler/escrever valores de sinais e controlar a operação dos processadores e da chave. O número máximo de processadores é 256. A simulação de elementos como memórias e bancos de registradores é realizada, de forma mais eficiente, pelo simulador de matrizes.

A descrição do sistema é feita em código compilado. Um elemento lógico corresponde a uma instrução do processador de elementos, gerada em tempo de compilaçao do sistema. Não há mecanismos de avaliação seletiva. A simulação pode ser feita com atraso zero, unitário ou no modo misto. São utilizados quatro valores lógicos: 0, 1, X, Z.

O particionamento do sistema é efetuado por um algoritmo que, na medida do possível, associa a cada processador os elementos e todos os componentes de sua lista de conexões de entrada [KR082].

A comunicação entre os processadores é efetuada através de uma rede de roteamento programável do tipo chave "crossbar",com 256 entradas e 256 saídas. A configuração da chave é determinada em tempo de compilação sendo armazenada numa memória endereçada por um contador.

O sincronismo entre os processadores é implementado por um relógio global. Cada processador possui uma memória de instruções e um CP (Contador de Programa) associado à mesma conforme mostra figura 4.16.

Os CP de todos os processadores são incrementados simultaneamente e apontam para o mesmo endereço das respectivas memórias de instruções.O contador da memória de

configuração da chave também utiliza o relógio global.

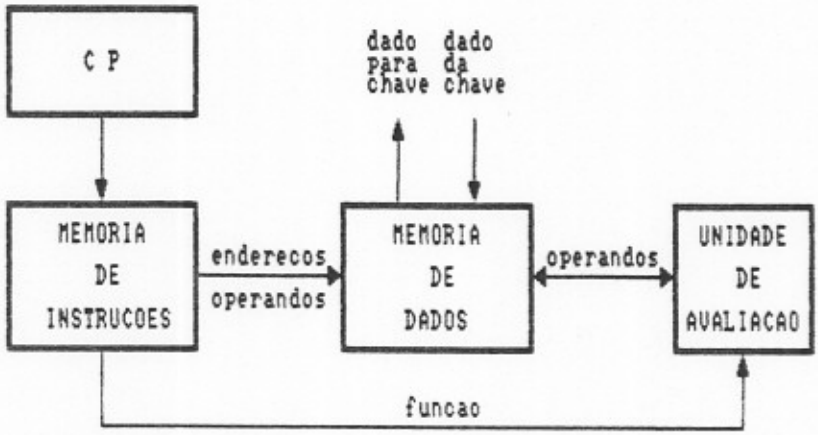



Figura 4.16 Processador de elementos

A memória de instruções comporta 8K elementos de quatro entradas. A memória de dados contém os valores lógicos dos elementos sendo do tipo multiporta. Há duas áreas distintas: memória local de dados e memória de dados para a chave. A simulação com atraso unitário obriga uma divisão ao meio destas duas áreas. A unidade de avaliação é implementada por um conjunto de memórias RAM. Uma RAM de função comporta 32 tipos de elementos. As quatro entradas e saída de um elemento podem ser condicionadas através de memórias RAM denominadas GDM (Generalized DeMorgan Memories). Elementos com mais de quatro entradas são avaliados iterativamente. O processador de elementos também utiliza técnicas de exploração da concorrência. Um pipeline de oito estágios executa as instruções conforme mostrado na figura 4.17.

1. Incrementar CP
2.3. Buscar instrução
4. Buscar operando
5. Avaliar elemento
6. Armazenar resultado
7. Buscar dado para a chave
8. Enviar dado para a chave

Figura 4.17 Pipeline do processador de elementos

O tempo em cada estágio do pipeline é de 80 ns o que dá uma taxa de avaliação de 12,5 milhões elementos/s em cada processador. Com os 256 processadores chega-se a 3,2 bilhões elementos/s. Esta fantástica velocidade, quando analisada mais a fundo, apresenta algumas restrições. A atividade de um sistema digital geralmente é baixa. Supondo um nível de atividade de 1% podemos considerar que a taxa de avaliação efetiva é de 32 milhões elementos/s.

Durante a compilação e o particionamento do sistema instruções nulas (NOP) são inseridas a fim de gerar um ordenamento entre as instruções de subsistemas que se comunicam [KRO82]. A consequência destas inserções é a queda no desempenho do sistema. Um exemplo apresentado em [KRO82] mostra que esta sobrecarga pode chegar a 300%. Neste caso um aumento do número de processadores de 4 para 256 causou um ganho de tempo de apenas 10 vezes.

### 4.2.3 Um computador dedicado à simulação lógica baseado em processamento distribuído [LEV82]

Esta AESL, de cuja implementação não se tem notícia, explora a concorrência através do particionamento do sistema. A figura 4.18 apresenta esta proposta de AESL com multiprocessamento que realiza simulação lógica baseada em tabelas e orientada a eventos.

O computador hospedeiro comunica-se com a AESL através do processador mestre. Entre as tarefas do hospedeiro estão a interface com o usuário e o particionamento do sistema com posterior carga nos processadores escravos. O particionamento é feito por um algoritmo que gera um conjunto de elementos partindo de uma entrada primária e colecionando elementos até chegar numa saída primária.

Os conjuntos são equalizados em tamanho e distribuídos entre os diversos processadores.

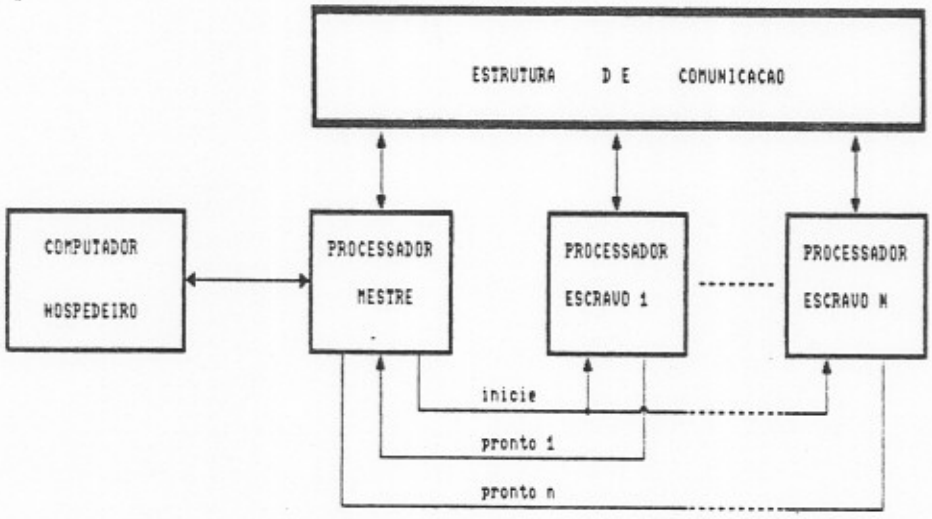



Figura 4.18 Multiprocessador para simulação lógica

Há dois tipos de processadores escravos: os avaliadores simples e os avaliadores funcionais. Os avaliadores simples recebem os conjuntos de elementos simples como portas lógicas, flip flops, etc, passíveis de avaliação por tabelas. Os avaliadores funcionais dedicam-se a conjuntos de elementos cujas funções lógicas são descritas por rotinas específicas.

A estrutura de comunicação entre os processadores, mostrada na figura 4.19, foi construída de forma a aproveitar as diferenças entre os dois tipos de avaliadores. Nos avaliadores simples o tempo de processamento por elemento é pequeno e, consequentemente, a banda passante da estrutura de comunicação deve ser elevada, a fim de não prejudicar o desempenho do processo. Este requisito é preenchido plenamente por uma rede de roteamento do tipo chave "crossbar". A avaliação de elementos funcionais é um processo mais lento. Um barramento paralelo compartilhado fornece a banda passante suficiente para os avaliadores funcionais, a um custo bem menor que uma rede de roteamento.

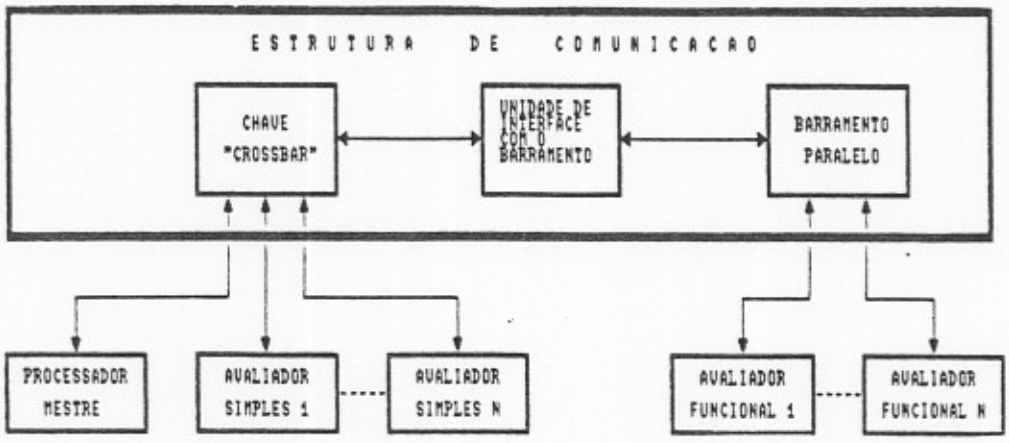



Figura 4.19 Estrutura de comunicação entre os processadores

O sincronismo entre processadores é baseado numa política de relógio global. O processador mestre incrementa o tempo, de forma unitária, somente quando todos os escravos estão livres. Cada escravo possui um sinal de PRONTO ( ver figura 4.18 ) que indica esta condição. Após incrementar o tempo o mestre ativa os escravos, para que procedam a simulação, através de um sinal de INICIE. Os escravos que não possuem evento programado para o tempo atual ativam seu sinal de PRONTO implementando um avanço do tempo por evento.

O processamento de subsistema, em cada escravo, é realizada por um processador único. O subsistema é descrito por tabelas. Um algoritmo de dois passos implementa o mecanismo de avaliação seletiva. A lista de eventos utiliza uma estrutura do tipo roda de tempo. Para reduzir o tráfego na estrutura de comunicação é adotada a política de somente enviar para outros processadores os eventos que serão efetivados no tempo seguinte.

O desempenho desta AESL depende, principalmente, do processador escolhido para os escravos e da banda passante da estrutura de comunicação. Assumindo que o tempo de comunicação é sempre menor ou igual ao tempo de processamento podemos estimar o desempenho desta AESL, para dois microprocessadores de 16 bits. A tabela 4.3 fornece a taxa

de avaliação, por processador, estimado para esta AESL.

Tabela 4.3 Desempenho da AESL

| PROCESSADOR ESCRAVO | AVALIAÇÕES POR SEGUNDO | |
|---|---|---|
| | ELEMENTOS SIMPLES | ELEMENTOS FUNCIONAIS |
| 8086 | 5009 | 125 |
| Am 29116 | 34965 | 874 |

O microprocessador Am29116 apresenta uma elevada taxa de avaliações, visto que foi considerada a microprogramação do algoritmo de simulação. As variações do desempenho desta AESL também foram analisadas em [LEV82] considerando as diversas opções de implementação.

### 4.2.4 Uma máquina para simulação lógica [ABR83]

A concorrência inerente ao processo de simulação é explorada nesta AESL através do particionamento do algoritmo utilizando-se processamento data flow. A figura 4.20 apresenta o diagrama em blocos desta AESL.

A descrição do sistema é feita por tabelas que estão divididas em três memórias de forma a compatibilizar seu tempo de acesso com as unidades funcionais. O algoritmo implementado por esta AESL utiliza avaliação seletiva sendo realizado em passo único.

O processador de evento corrente busca na memória da lista de eventos os associados ao tempo atual, enviando-os para as filas das duas unidades seguintes no data flow. Este processador também controla os sinais de saída do sistema, os pontos de parada, as oscilações e os sinais em monitoração.

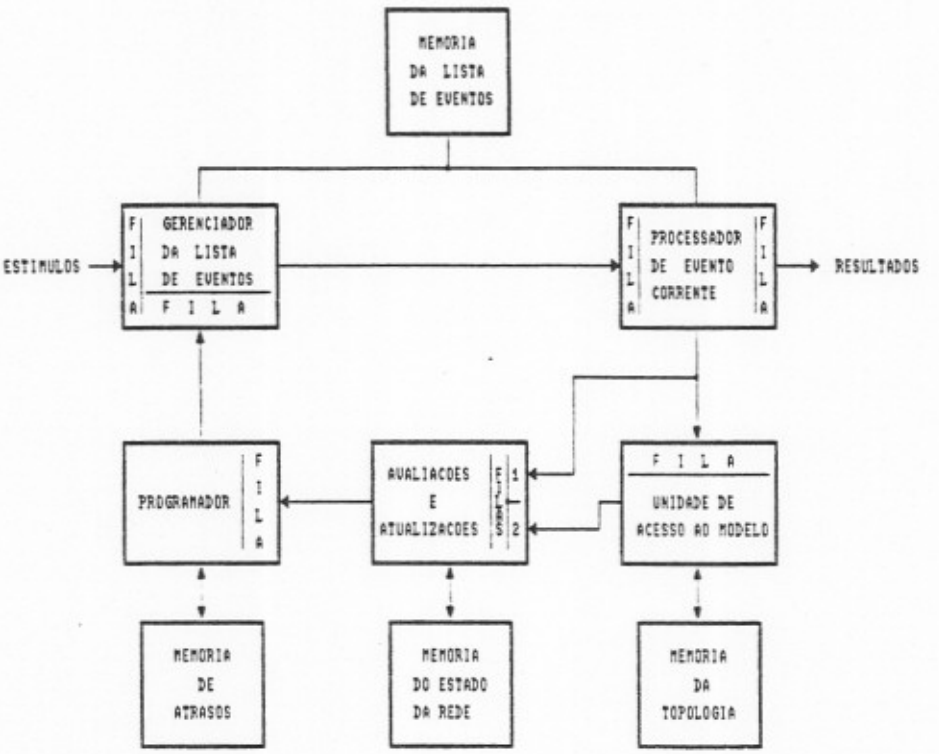



Figura 4.20 Estrutura da AESL

A unidade de acesso ao modelo, para cada evento recebido, envia para a fila da unidade de atualizações e avaliações a lista de conexões de saída e a função do elemento associado ao evento. Estas informações do sistema estão na memória da topologia.

A unidade de atualizações e avaliações possui uma estrutura para elementos simples ( portas lógicas, flip flops, etc ) e outra para elementos funcionais, conforme figura 4.21. Também são atualizadas as entradas dos elementos que compoem as listas de conexões de saída, pertencentes aos elementos cuja saída variou.

O processador de configuração simples atualiza a saída dos elementos em função dos eventos programados para o tempo atual. Os elementos ativos e o estado de suas entradas e saídas são enviados para o avaliador simples que utiliza tabelas para avaliar os elementos. Este, por sua vez, envia

para a fila do programador os eventos ou cancelamentos de eventos ( algoritmo de passo único ) necessários àqueles elementos.

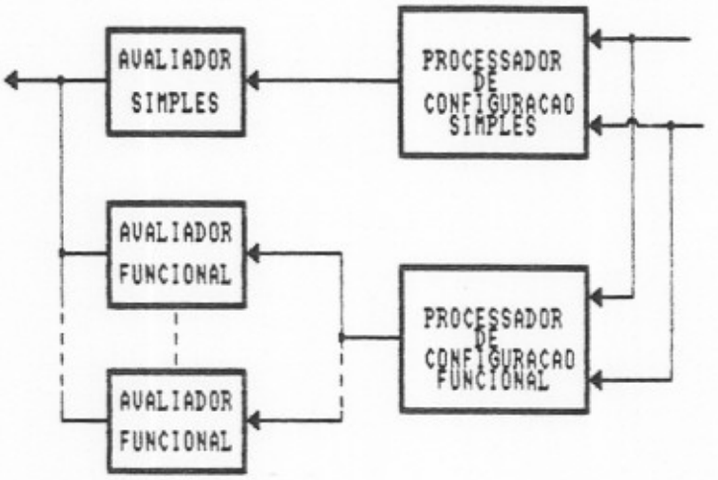



Figura 4.21 Unidade de avaliações e atualizações

Os elementos funcionais, em virtude de seu maior tempo de avaliação, são tratados de forma concorrente pelos avaliadores funcionais. O processador de configuração funcional distribui os blocos funcionais entre os avaliadores de forma a maximizar a concorrência do processo. As rotinas de avaliação correspondem a microprogramas acessados a partir do tipo do elemento.

O programador associa aos eventos recebidos os atrasos dos elementos retirados da memória de atrasos. Os eventos e os pedidos de cancelamento são enviados à fila do gerenciador da lista de eventos.

O gerenciador controla a lista de eventos que é uma estrutura do tipo roda de tempo. Quando o programador sinaliza que não há mais eventos a programar no tempo atual, o gerenciador avança o tempo até o valor associado com o evento mais próximo, na roda de tempo. Os estímulos de entrada também levam à programação de eventos.

O gerenciador ativa o processador de evento corrente enviando o novo tempo e o endereço do próximo evento, na memória da lista de eventos.

As cinco unidades funcionais implementam uma arquitetura data flow. Esta característica está bem presente pois cada unidade condiciona a realização de suas tarefas à chegada dos dados e funções. Caracteriza-se assim uma relação produtor-consumidor entre as diversas unidades funcionais.

A abordagem de passo único, apesar da necessidade de cancelamento de eventos, permite uma utilização mais intensa das unidades que o algoritmo de dois passos adotado em [BAR80].

Cada unidade funcional é microprogramada de forma a garantir velocidade e flexibilidade para futuras alterações no algoritmo. O desempenho esperado desta AESL, tendo em vista não ter sido relatada sua implementação, foi estimado entre 500 mil e um milhão avaliações/s.

### 4.2.5 ULTIMATE: simulação lógica em hardware [GLA84]

A exploração da concorrência, baseada no particionamento do algoritmo, e utilizando processamento pipeline, é claramente empregada nesta proposta de AESL denominada ULTIMATE. Um algoritmo de simulação é mapeado diretamente para um processador pipeline. A descrição do sistema é baseada em tabelas, sendo utilizado um mecanismo de avaliação seletiva. A figura 4.22 apresenta, em linhas gerais, a arquitetura do ULTIMATE.

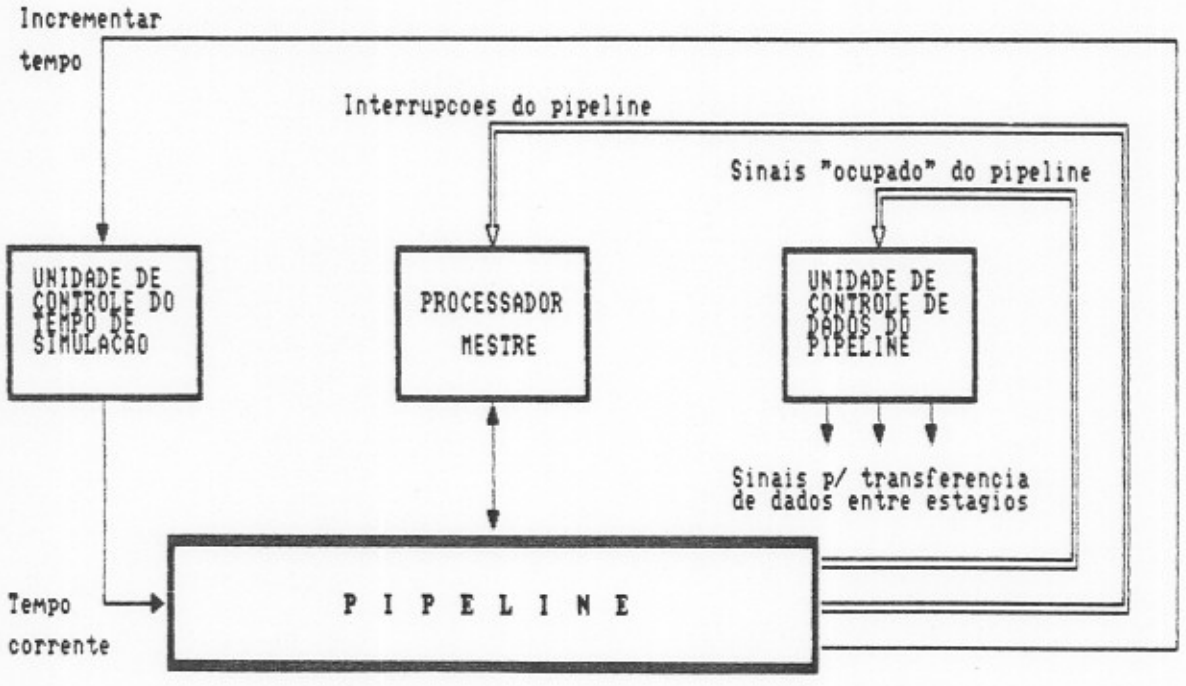



Figura 4.22 Arquitetura básica do ULTIMATE

O processador mestre controla a simulação gerando o relógio para o pipeline. Os procedimentos pouco utilizados não foram implementados no pipeline sendo executados pelo processador mestre. O pipeline ativa estes procedimentos através de um conjunto de interrupções. Os diversos estágios do pipeline podem apresentar tempos diferentes de execução.

A unidade de controle de dados do pipeline monitora os sinais de "ocupado" de cada estágio. A transferência dos dados de um estágio somente é habilitada quando todos os seus sucessores não se encontram "ocupados".

A unidade de controle do tempo de simulação incrementa o mesmo quando a lista de eventos para o tempo atual está vazia e não há mais elementos para avaliar. O algoritmo utilizado pelo pipeline possui uma característica própria: simulação em passo único e em dois passos realizadas conjuntamente. Desta forma pode-se aproveitar as vantagens dos dois métodos. Os elementos classificados como primitivos são

avaliados em passo único enquanto os não primitivos em dois passos. Um elemento não primitivo é aquele que possui uma grande probabilidade de ser reavaliado ou cuja reavaliação prejudica o desempenho da simulação. Há critérios para classificar, automaticamente, os elementos durante a fase de compilação do sistema. Todavia, o melhor deles é a experiência do usuário. O pipeline e seus diversos módulos de memória associados estão na figura 4.23.

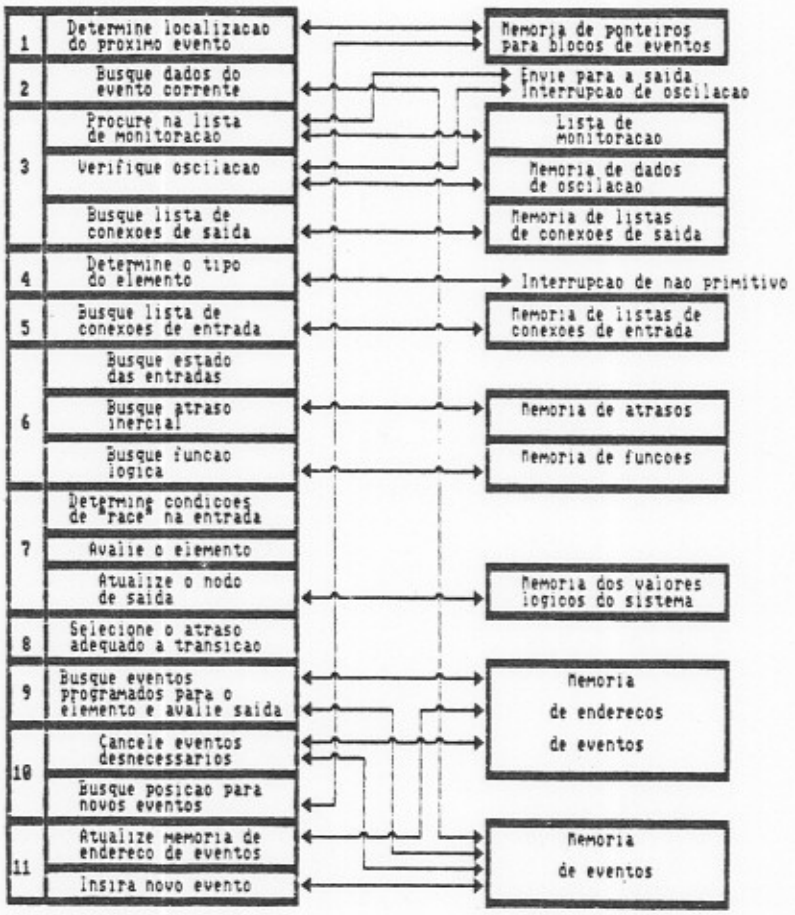



Figura 4.23 Processador pipeline do ULTIMATE

A memória de eventos contém a lista de eventos. Os endereços inicial e final de cada lista ficam na memória de ponteiros para blocos de eventos. Os elementos são associados a seus eventos pendentes através da memória de endereços de eventos. Elementos primitivos são avaliados por tabelas (estágio 7 do pipeline). Após o processamento de todos os

eventos pelo pipeline o mestre avalia os elementos não primitivos. Elementos não previstos para o pipeline podem ser avaliados por rotinas no mestre. A análise do algoritmo apresentado em [GLA84] mostra que o avanço do tempo é por evento.

O desempenho previsto para o ULTIMATE depende, principalmente, do número de entradas médio de cada elemento e do tempo de acesso das memórias do pipeline. Para um tempo de acesso de 100 ns e um número médio de entradas igual a um a taxa de avaliação do pipeline chega ao seu máximo: 5 milhões avaliações/s. Se o número médio de entradas for três a taxa de avaliações cai para 2,2 milhões avaliações/s. Esta sensibilidade ao número de entradas deve-se ao acesso sequencial à lista de conexões de entrada de cada elemento, constituindo um gargalo do pipeline.

### 4.2.6 HAL: uma máquina de alta velocidade para simulação lógica [KOI85]

Esta AESL, denominada HAL, foi concebida pela Nippon Electric Corporation com a finalidade de simular grandes sistemas a velocidades que tornem possível um teste efetivo dos mesmos, antes de sua implementação.

O algoritmo de simulação adotado é o mais simples para implementação em hardware, ou seja, o de atraso zero com dois valores lógicos: 0 e 1. A técnica adotada é a da simulação mista. Uma análise detalhada de "timing" além de reduzir a velocidade de simulação, implicaria num acréscimo de três vezes na capacidade de memória do simulador [TAK86].

A descrição do sistema é baseada em blocos. Um bloco é definido como um conjunto de elementos lógicos. Um conjunto de elementos lógicos simples ( portas lógicas e

registradores ) é chamado bloco lógico. Da mesma forma, um conjunto de memórias é chamado bloco de memória. A simulação, a nível de blocos, utiliza mecanismos de avaliação seletiva. Os blocos são classificados em níveis, das entradas primárias para as saídas, de forma a permitir sua avaliação na ordem correta. Dentro de cada bloco os elementos também são classificados em níveis.

A exploração da concorrência é baseada no particionamento do sistema. Um conjunto de 32 processadores implementa um simulador de atraso zero conforme mostrado na figura 4.24.

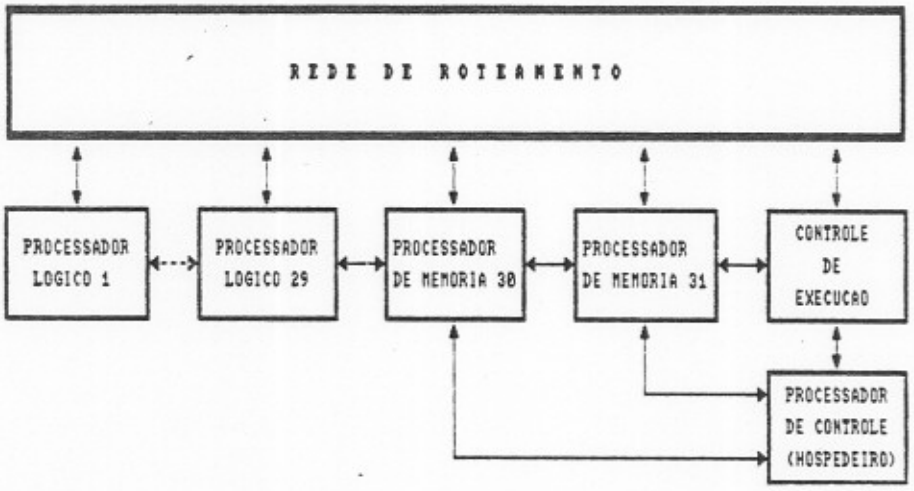



Figura 4.24 Diagrama em blocos do HAL

O processamento de subsistema é efetuado pelos processadores lógicos e pelos processadores de memória.

Os 29 processadores lógicos realizam a simulação a nível de blocos. Cada um é composto por um processador de nodo e um "gate array" dinâmico, conforme figura 4.25, responsáveis pela simulação dos blocos lógicos.

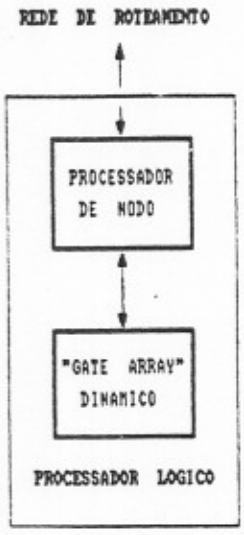



Figura 4.25 Estrutura de um processador lógico

A concorrência é novamente explorada a nível de subsistema. O processador de nodo é um processador data flow com dez estágios que controla a comunicação com a rede de roteamento, mantém a lista de eventos, a lista de conexões e os valores lógicos do subsistema [BLA84]. A capacidade máxima por processador é de 1024 blocos, com 32 entradas e saídas em cada bloco.

O "gate array" dinâmico realiza a avaliação dos blocos lógicos. Os elementos, correspondentes aos diversos níveis dentro de cada bloco, são avaliados através de tabelas em memória RAM. Em cada nível do bloco podem ser avaliados, simultaneamente, 16 elementos de 16 entradas. O processo de avaliação pode ser iterativo.

Dois processadores de memória foram criados a fim de minimizar o gasto de tempo e espaço decorrente da representação de memórias ( RAM e ROM ) baseada em portas lógicas e registradores. A figura 4.26 mostra o processador de memória, responsável pela simulação dos blocos de memória.

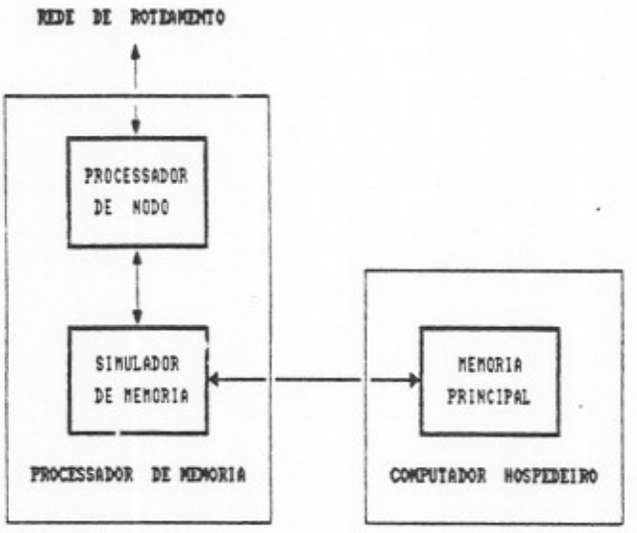



Figura 4.26 Estrutura de um processador de memória

A memória utilizada na simulação destes blocos é a do computador hospedeiro, acessada através de um mecanismo de acesso direto à memória ( DMA ).

A comunicação entre os processadores é feita utilizando-se uma rede de roteamento. Esta rede, com 32 entradas e 32 saídas, é implementada por uma rede multiestágio que possui 80 células de roteamento. As células são chaves "crossbar" 2 X 2 com registradores, permitindo a transmissão paralela de eventos de forma pipeline.

O sincronismo entre os processadores é baseado numa política de relógio global. O processador de controle só avança a simulação para o próximo nível quando todos os processadores de nodo concluíram a avaliação do nível atual.

A capacidade máxima do HAL, com todos os processadores instalados, é de 20 mil blocos lógicos e mil blocos de memória ( até 2 Mbytes ). Estima-se que o HAL pode simular um ciclo de relógio com 20 mil blocos em 5,7 ms. Num simulador lógico esta taxa corresponde a uma capacidade de 1,5 milhões de elementos e uma taxa de avaliação de 260 milhões elementos/s.

A simulação de um subsistema de memória de um grande computador foi realizada para análise de desempenho. O sistema possui 4.500 blocos ( 100K portas lógicas ), 2 Mbyte de memória, 52 tipos de blocos e 34 níveis. Foram efetuadas várias simulações, variando-se o número de processadores. O tempo de simulação, para 10 mil ciclos de relógio, está representado na figura 4.27.

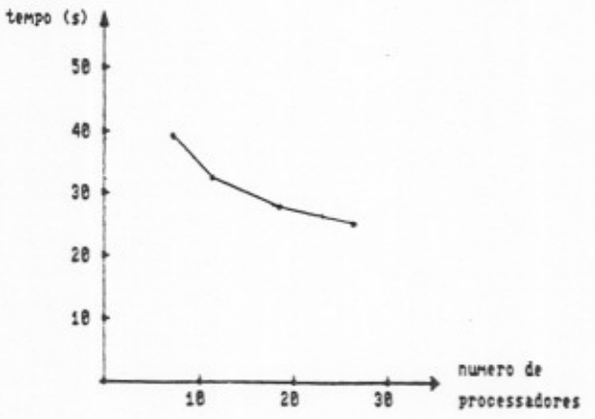



Figura 4.27 Simulação de um subsistema de memória no HAL

A figura mostra que o tempo de simulação decresce com o aumento do número de processadores. O ganho em velocidade, entretanto, tende a estabilizar-se com um grande número de processadores. Isto ocorre devido ao pequeno tamanho do sistema em análise e ao desbalanceamento de carga entre os processadores.

### 4.2.7 Atacando o gargalo da simulação [ALL85]

O LE ( Logic Evaluator ) foi a primeira AESL fabricada comercialmente. A exploração da concorrência foi efetuada pelo particionamento do circuito. A estrutura adotada é semelhante a [LEV82] conforme mostrado na figura 4.28.

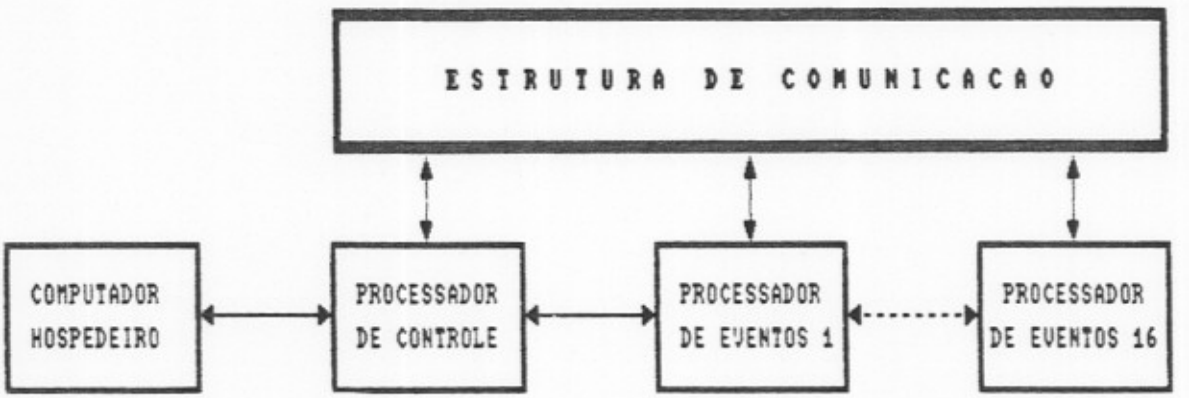



Figura 4.28 Arquitetura do LE

A descrição do sistema é baseada em tabelas. Os elementos lógicos que formam um sistema são: elementos de memória ( RAM, ROM, PLA, etc ) e elementos lógicos com três entradas e uma saída cuja função é descrita por uma tabela [BLA84]. Uma lógica com três estados e quatro intensidades permite modelar barramentos e transistores de passagem. Os elementos podem ter atrasos associados às entradas e saídas.

O processador de controle tem por função:

- gerenciar a comunicação com o hospedeiro;
- controlar o tempo de simulação;
- simular os elementos de memória.

A comunicação com o hospedeiro é feita por uma interface paralela com 16 bits e taxa de transferência de 1 a 2 Mbytes/s permitindo carregar o sistema rapidamente. A figura 4.29 [IVE82] apresenta a estrutura interna do processador de controle e de um processador de eventos.

Cada processador de eventos tem capacidade para 64K elementos de três entradas o que equivale a 100K elementos de 2 entradas. Um pipeline de cinco estágios explora a concorrência do algoritmo alcançando uma taxa de avaliação de um milhão eventos/s, no modelo LE1002 com um processador.

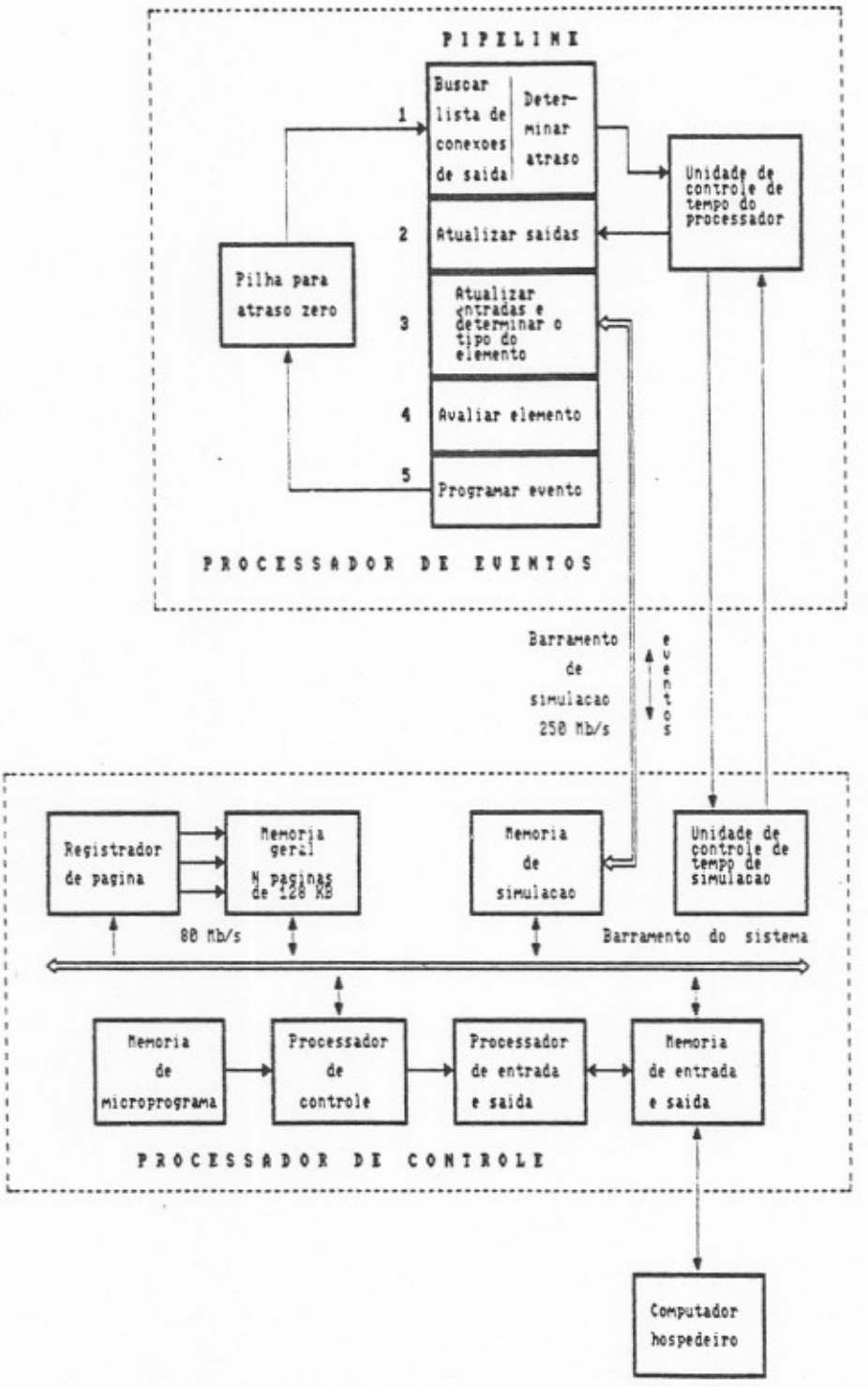



Figura 4.29 Estrutura interna dos processadores do LE

A unidade de controle de tempo do processador implementa uma roda de tempo. O sincronismo entre processadores é feito por um relógio global implementado pela unidade

de controle de tempo de simulação.

A comunicação entre os processadores é feita por um barramento paralelo compartilhado com uma elevada banda passante ( 250 Mb/s ).

O desempenho do LE com 16 processadores (LE 1032) pode chegar a 16 milhões eventos/s. Considerando uma média de 2,4 avaliações por evento [BLO87] obtemos 38,4 M avaliações/s.

### 4.2.8 Conceitos data flow aceleram a simulação num sistema de PAC [PAS85]

A Daisy Systems Corporation desenvolveu esta AESL para uma estação de trabalho Megalogician. A arquitetura adotada permite um ganho de velocidade de até cem vezes. Sua concepção visou um baixo custo, a utilização de tecnologia convencional e a flexibilidade para novas aplicações e algoritmos.

O modelo e as técnicas de simulação permanecem os mesmos da estação de trabalho, a saber:

- simulação baseada em tabelas e orientada a eventos;
- lógica de três estados e 4 intensidades;
- elementos podem ser portas lógicas, transistores de passagem, modelos funcionais ,modelos comportamentais, etc;
- utilização de modelos em hardware.

A presença do acelerador é transparente ao usuário da estação de trabalho, exceto pelo ganho de velocidade.

O particionamento do algoritmo foi a abordagem escolhida conforme mostrado na figura 4.30.

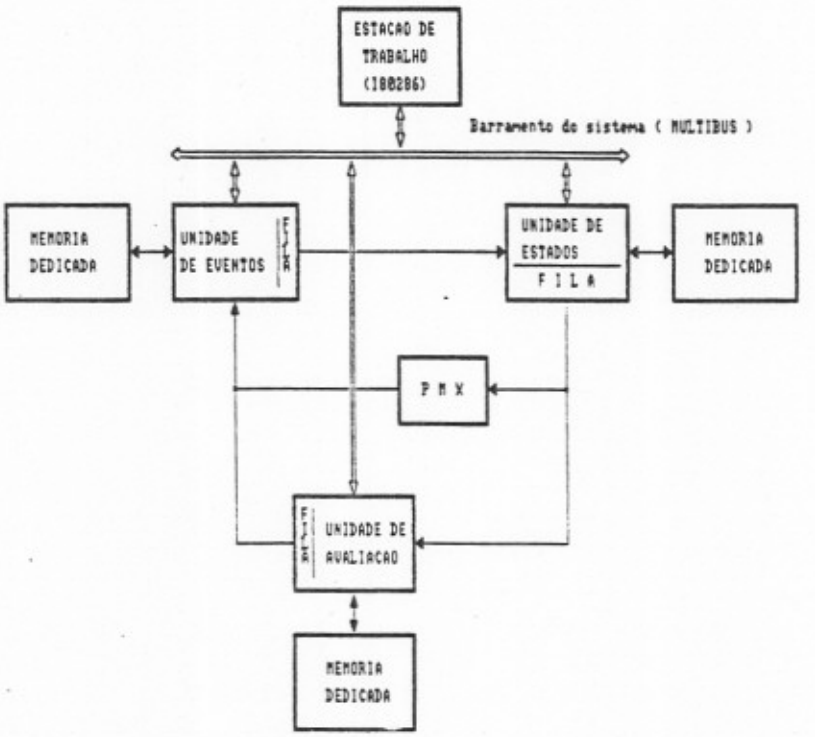



Figura 4.30 AESL para estação de trabalho Megalogician

Três unidades, dispostas em anel, formam uma arquitetura data flow. Cada unidade implementa uma parte do algoritmo através de um processador "bit-slice", uma memória local e uma fila.

A unidade de avaliação recebe as funções lógicas e seus valores de entrada, calculando os valores de saída. Os módulos PMX atuam como coprocessadores desta unidade avaliando modelos em hardware.

A unidade de eventos recebe os novos valores de saída dos elementos e programa, numa lista encadeada, os eventos necessários para sua efetivação. A memória desta unidade armazena, além da lista de eventos, as informações sobre os atrasos dos elementos. Os eventos programados para

o tempo atual são enviados para a unidade de estados.

A unidade de estados combina funções lógicas e valores de entrada enviando-os para a unidade de avaliação. Sua memória contém as listas de conexões de entrada e saída do elemento e o valor atual da saída do mesmo.

Os três módulos estão ligados a um barramento comum, pertencente à estação de trabalho, que controla a simulação. A taxa de avaliações chega a um máximo de 100 mil portas lógicas/s com uma capacidade de 64 K primitivas ou 1M portas lógicas [BLA84].

### 4.2.9 Quadro comparativo

As diversas AESL vistas podem ser comparadas à luz da taxonomia proposta, conforme apresentado nas tabelas 4.4 e 4.5. A distribuição das AESL na taxonomia proposta é apresentada na tabela 4.6.

Tabela 4.4 Comparação das AESL através da taxonomia proposta

| PROPOSTA | SITUAÇXO | EXPLORAÇXO DA CONCORRÊNCIA | CARACTERÍSTICAS DO PARTICIONAMENTO | DESCRIÇXO DO SISTEMA |
|---|---|---|---|---|
| [BAR80] | proposta | particionamento do algoritmo | processamento data flow<br>fase de avaliação microprogramada<br>fase de atualização em hardware | tabelas |
| YORKTOWN SIMULATION ENGINE | protótipo | particionamento do sistema | relógio global<br>comunicação por chave "crossbar"<br>processamento pipeline de subsistema:<br>- em hardware<br>máximo de 256 processadores | código compilado |
| [LEV82] | proposta | particionamento do sistema | relógio global<br>comunicação por chave "crossbar"<br>e barramento compartilhado<br>uniprocessamento de subsistema:<br>- microprogramado | tabelas |
| [ABR83] | proposta | particionamento do algoritmo | processamento data flow:<br>- microprogramado | tabelas |
| ULTIMATE | proposta | particionamento do algoritmo | processamento pipeline:<br>- em hardware | tabelas |
| HAL | protótipo | particionamento do sistema | relógio global<br>comunicação por rede de roteamento<br>processamento data flow de subsistema<br>- em hardware<br>máximo de 32 processadores | tabelas |
| LOGIC EVALUATOR | comercial | particionamento do sistema | relógio global<br>comunicação por barramento compartilhado<br>processamento pipeline de subsistema:<br>- em hardware<br>máximo de 16 processadores | tabelas |
| DAISY MEGALOGICIAN | comercial | particionamento do algoritmo | processamento data flow:<br>- microprogramado | tabelas |

Tabela 4.5 Critérios auxiliares da taxonomia

| PROPOSTA | FLUXO DO TEMPO | AVALIAÇÃO DOS ELEMENTOS | MODELO ADOTADO | CAPACIDADE (elementos) | TAXA DE AVALIAÇÕES (elementos/s) |
|---|---|---|---|---|---|
| [BARB0] | avanço incremental do tempo<br>lista de eventos única<br>algoritmo de dois passos | sequencial<br>por rotinas | lógica de 3 valores:<br>- 0, 1 e X<br>atraso de transporte<br>atribuível:<br>-subida e descida | 32K | 200K |
| YORKTOWN SIMULATION ENGINE | avanço incremental do tempo<br>lista de eventos inexistente | em cada processador:<br>- sequencial<br>- por tabelas,<br>- em hardware | lógica de 4 valores<br>- 0, 1, X e Z<br>atrasos:<br>- 0, 1 e misto | 2M | 3,2G |
| [LEV82] | avanço do tempo por evento<br>lista de eventos:<br>- distribuída<br>- roda de tempo<br>algoritmo de dois passos | em cada processador:<br>- sequencial | -- | -- | 34.965<br>( por processador) |
| [ABRB3] | avanço do tempo por evento<br>lista de eventos:<br>- única<br>- roda de tempo<br>algoritmo de passo único | elementos simples:<br>- sequencial<br>- por tabelas<br>em software<br>elementos funcionais:<br>- concorrente<br>- por rotinas | -- | -- | 1K |
| ULTIMATE | avanço do tempo por evento<br>lista de eventos única<br>algoritmo de passo único e dois passos | elementos primitivos:<br>- sequencial<br>- por tabelas<br>em hardware<br>elementos não primitivos:<br>- sequenciais<br>- rotinas | -- | -- | 3,3K |
| HAL | avanço incremental do tempo<br>lista de eventos distribuída | em cada processador:<br>- concorrente<br>- por tabelas | atraso 0 e misto<br>lógica de 2 valores<br>portas lógicas e<br>registradores | 1,5K | 260K |
| LOGIC EVALUATOR | avanço do tempo por evento<br>lista de eventos:<br>- distribuída<br>- roda de tempo<br>algoritmo de passo único | em cada processador:<br>- por tabelas | lógica de 12 valores:<br>- 3 estados<br>- 4 intensidades | 1,6K | 38,4K |
| DAISY MEGALOGICIAN | avanço do tempo por evento<br>lista de eventos única<br>- encadeada | sequencial | lógica de 12 valores:<br>- 3 estados<br>- 4 intensidades | 1K | 100K |

Tabela 4.6 Distribuição das AESL na taxonomia

| PROPOSTA | EXPLORAÇÃO DA CONCORRÊNCIA | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | PARTICIONAMENTO DO ALGORITMO | | PARTICIONAMENTO DO SISTEMA | | | | | | |
| | | | SINCRONISMO | | COMUNICAÇÃO ENTRE PROCESSADORES | | | PROCESSAMENTO DE SUBSISTEMA | |
| | PROCESSAMENTO PIPELINE | PROCESSAMENTO DATA FLOW | RELÓGIO GLOBAL | RELÓGIO LOCAL | REDE DE ROTEAMENTO | BARRAMENTO COMPARTILHADO | UNIPROCESSADOR | EXPLORAÇÃO DA CONCORRÊNCIA | |
| [BARB0] | | * | | | | | | | |
| TSE | (1) | | * | | * | | | * | |
| [LEV82] | | | * | | * | * | * | | |
| [ABR83] | | * | | | | | | | |
| ULTIMATE | * | | | | | | | | |
| HAL | | (1) | * | | * | | | * | |
| LOG. EVA. | (1) | | * | | | * | | * | |
| DAIST | | * | | | | | | | |

(1) forma de exploração da concorrência no processamento de subsistema

## 4.3 OUTRAS TAXONOMIAS

Uma taxonomia para AESL foi apresentada em [FRA84]. A base para a mesma é o fato do algoritmo de simulação possuir três componentes básicos:

- controle do tempo;
- controle da lista de eventos;
- avaliação dos eventos ( funções ).

A tabela 4.7 [WON86] apresenta esta taxonomia.

Tabela 4.7 Componentes da taxonomia

| | | |
|---|---|---|
| CONTROLE DO TEMPO<br>Avanço do tempo<br>Sincronização | UNITÁRIO (UI)<br>RELÓGIO GLOBAL (GC) | POR EVENTO (EI)<br>RELÓGIO LOCAL (LC) |
| LISTA DE EVENTOS | ÚNICA (SL) | MÚLTIPLA (ML) |
| AVALIAÇÃO DOS EVENTOS/FUNÇÕES | SERIAL (SM) | PARALELA (MM) |

Os mecanismos de controle do tempo possuem dois aspectos principais: avanço do tempo e sincronização.

O avanço do tempo pode ser unitário (UI) ou por evento (EI) de forma semelhante ao apresentado no item 4.1.3.2 de nossa taxonomia.

A existência de vários processadores exige uma sincronização entre os tempos dos mesmos. O controle centralizado do tempo caracteriza um relógio global (GC). No relógio local (LC) cada processador possui seu próprio controle de tempo.

A lista de eventos, visto que podem existir vários processadores, pode ser única (SL) ou múltipla (ML)

A forma de avaliação dos eventos/funções finaliza a taxonomia. O emprego de uma única máquina, com este propósito, caracteriza a análise serial (SM). A avaliação paralela (MM) ocorre quando são utilizadas várias máquinas.

A combinação dos diversos itens desta taxonomia permite identificar 16 tipos diferentes de arquiteturas. Um monoprocessador executando um algoritmo de simulação com avanço unitário do tempo é classificado conforme figura 4.31.

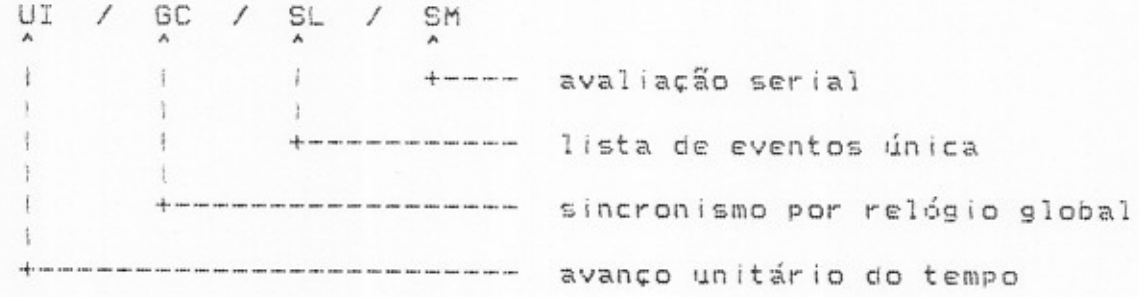



Figura 4.31 Classificação de um monoprocessador

A utilização de um algoritmo com avanço do tempo por evento leva à classificação EI/GC/SL/SM.

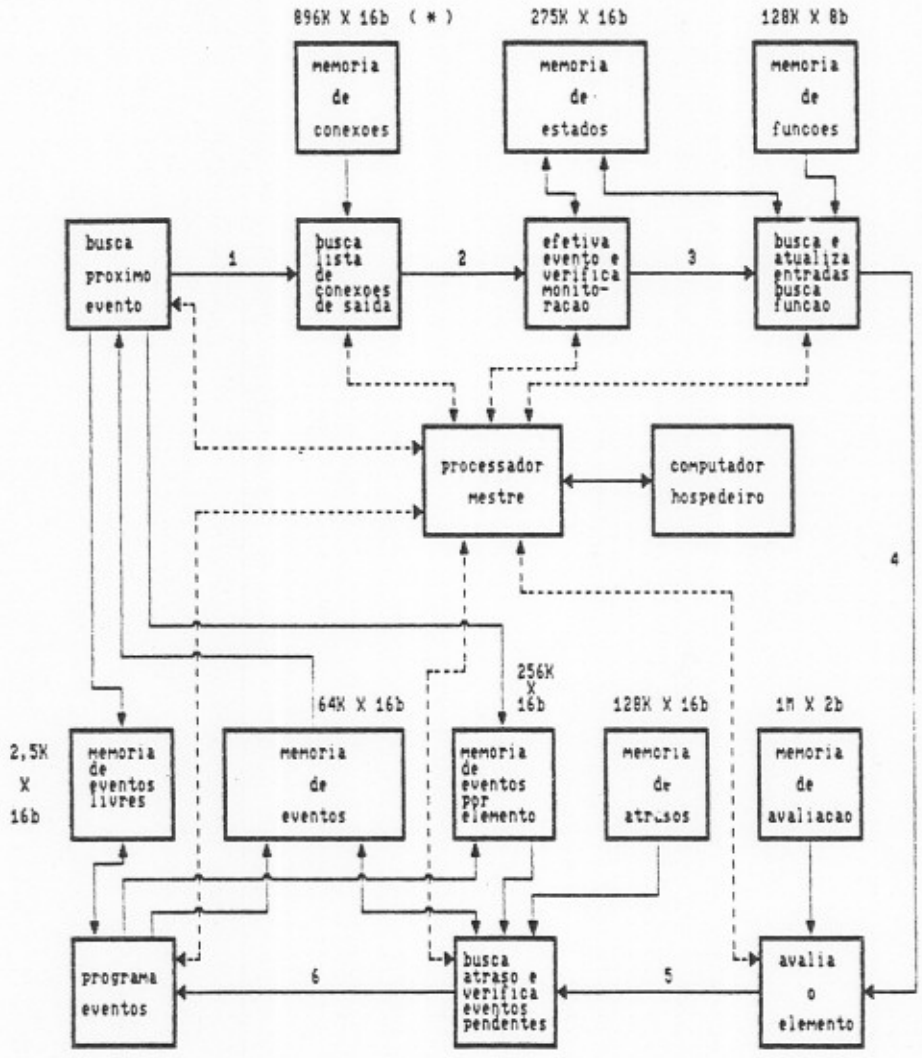
Das 14 classificações restantes nem todas são aproveitáveis. Combinações como ML/SM ou LC/SL/MM não possuem nenhum significado prático.

Esta taxonomia é semelhante, em alguns aspectos, à nossa proposta. Os mecanismos de controle do tempo e a lista de eventos também foram consideradas por nós. O critério da avaliação dos eventos/funções pode causar algumas dúvidas. Em algumas arquiteturas [ABR83] [PAS85] os eventos são avaliados serialmente (SM) enquanto as funções podem ser avaliadas de forma concorrente (MM). A forma de comunicação entre os processadores, um fator importante no desempenho de uma AESL que explora o particionamento do sistema, também não foi abrangida nesta taxonomia. A caracterização de uma arquitetura que utiliza simulação baseada em código compilado é feita em nossa taxonomia pelo critério da descrição do sistema que foi desconsiderado nesta análise.

# 5 PROPOSTA DE AESL

A proposição de uma nova AESL depende de vários fatores como: limitações tecnológicas, compromissos entre custo e desempenho, flexibilidade da arquitetura escolhida, etc.

A base de nossa proposta é a utilização de tecnologia já dominada visando a facilidade de implementação. A inexistência de estruturas complexas de comunicação como barramentos compartilhados de alta velocidade ou redes de roteamento é portanto aconselhável, apesar da redução na flexibilidade da arquitetura. A figura 5.1 apresenta a arquitetura proposta. Nosso objetivo é posicionar esta AESL num estágio intermediário entre as soluções genéricas como a utilização de um multiprocessador e soluções específicas como a implementação direta "em hardware" de um determinado algoritmo.




( * ) dimensoes validas para a implementacao descrita no item 5.2

BUFFERS ENTRE ESTAGIOS

1 EVENTO ( elemento , valor )

2 EVENTO , NRO FAN OUT , FAN OUT ( elemento , pos entrada )

3 VALOR , FAN OUT

4 ELEMENTO , FUNCAO , SAIDA , NRO ENT , ENT1 , ENT2 , .... , ENT8

5 ELEMENTO , SAIDA , NOVA SAIDA

6 EVENTO ( elemento , nova saida , tempo ) , END ULTIMO EV ELEMENTO

Figura 5.1 Proposta de AESL

## 5.1 DESCRIÇÃO DA ARQUITETURA

Utilizando a taxonomia definida no capítulo 4 podemos dizer que a exploração da concorrência, nesta AESL, é baseada no particionamento do algoritmo.

Sete estágios implementam as diferentes etapas de um algoritmo de simulação de passo único mantendo, entre si, uma relação do tipo produtor-consumidor. Cada estágio só inicia suas atividades quando recebe um pacote de dados de seu predecessor. A transferência de dados entre os diversos estágios é feita de forma assíncrona caracterizando, segundo nossa taxonomia, uma arquitetura data flow.

A descrição do sistema digital e a estrutura de dados do algoritmo de simulação estão distribuídas em oito memórias, acessadas pelos diversos estágios. A função de cada estágio e a consequente distribuição dos dados nas memórias depende do algoritmo de simulação adotado. Nossa descrição pressupõe um determinado algoritmo que será detalhado no item 5.2.

Inicialmente o processador mestre ativa a unidade busca_próximo_evento fornecendo o tempo atual. Esta unidade busca, na memória_de_eventos, aqueles que possuem tempo associado igual ao atual enviando-os para a unidade seguinte. A área ocupada por cada evento na memória_de_eventos é liberada atualizando-se a memória_de_eventos_livres. A memória_de_eventos_por_elemento também é atualizada caso os elementos associados aos eventos possuam, para algum tempo futuro, eventos programados.

A unidade busca_lista_de_conexões_de_saída recebe o evento e envia-o para a unidade seguinte juntamente com a lista de conexões de saída do elemento associado ao mesmo. A lista é acessada na memória_de_conexões.

A unidade efetiva_evento_e_verifica_monitoração efetiva o evento atualizando a memória_de_estados. Caso o elemento esteja sob monitoração o nome do elemento e o novo valor de sua saída são enviados para o mestre. O elemento, seu novo valor de entrada e a posição da entrada que deve receber este valor são enviados para a unidade seguinte.

A fim de eliminar as listas de conexões de entradas a memória_de_estados armazena, de forma redundante, o valor da saída e das entradas de cada elemento. Desta forma, para cada evento, deve-se atualizar a saída do elemento associado ao evento e as entradas de todos os elementos pertencentes à lista de conexões de saída do mesmo. A atualização das entradas é feita pela unidade busca_e_atualiza_entradas/busca_função que também acessa a memória_de_funções associando uma função ao elemento.

A avaliação dos elementos é efetuada na unidade avalia_o_elemento. Esta unidade recebe da anterior o elemento, função lógica, saída atual e valor de suas entradas. Com estes dados é possível, utilizando tabelas existentes na memória_de_avaliação, calcular o novo valor da saída, a partir da função do elemento e do valor de suas entradas. Elementos mais complexos podem ser avaliados por rotinas previamente carregadas na memória de programa desta unidade.

A unidade busca_atraso_e_verifica_eventos_pendentes é responsável pelo cancelamento de eventos e pela decisão de programar um novo evento. Para tanto ela percorre, na memória_de_eventos, uma lista formada por todos os eventos programados para um determinado elemento (eventos pendentes), ordenados de acordo com seu tempo de efetivação. O endereço do primeiro evento desta lista é obtido na memória_de_eventos_por_elemento. O cancelamento de eventos, quando necessário, é feito através da desativação de um "flag" de validade associado ao evento. A área ocupada na

memória_de_eventos será liberada posteriormente pela unidade busca_eventos. O atraso de tempo associado a um novo evento é buscado na memória_de_atrasos. A programação de um evento é efetuada enviando-se para a unidade programa_eventos os dados necessários à programação, ou seja: elemento, nova saída, tempo associado à transição e endereço do último evento pendente do elemento.

Programar um evento consiste em obter, na memória_de_eventos_livres, uma área disponível na memória_de_eventos e inserir nela o evento. O encadeamento deste evento com o anterior para o elemento também é efetuado na unidade programa_eventos. As alterações nos sinais de entrada do sistema são efetivadas pela programação de eventos fornecidos pelo mestre.

O processador de cada um dos sete estágios pode ser implementado por um microprocessador ou por um processador microprogramado. Esta escolha não é, certamente, a de melhor desempenho. Flexibilidade e menor custo devem compensar esta desvantagem. O processo de simulação não exige dados com elevado número de bits. Processadores de 8 bits, entretanto, tenderiam a fragmentar demais os dados na memória aumentando o número de acessos à mesma. A maior facilidade na obtenção de ferramentas de desenvolvimento de software e hardware ( emuladores, montadores, etc ) faz pender nossa decisão para processadores com palavra de 16 bits. O emprego de um processador com esta palavra justifica-se, principalmente, por implicar em menores custos de implementação.

A memória de programa, em cada estágio, é do tipo volátil, sendo carregada pelo processador mestre. Desta forma, há uma grande liberdade na escolha do algoritmo utilizado na simulação. Esta flexibilidade facilita o desenvolvimento de melhores algoritmos visando, por exemplo, sua futura implementação 'em hardware'.

O avanço do tempo de simulação somente é efetuado depois que todos os estágios concluíram suas atividades. Isto é necessário visto que eventos efetivados no tempo atual podem cancelar eventos no tempo seguinte. Utilizando-se buffers entre estágios com tamanho maior que um aceleramos a liberação dos estágios mais rápidos. Este fato, todavia, não altera significativamente o rendimento global da AESL, conforme será visto posteriormente. A solução adotada é o emprego de buffers unitários entre os estágios.

O processador mestre é o responsável pelo controle da simulação. Entre suas atribuições podemos citar:

- carga do programa dos estágios;
- carga da estrutura de dados do sistema;
- controle do tempo global de simulação;
- comunicação com o hospedeiro durante a simulação.

A comunicação com o hospedeiro deve ser feita por uma interface paralela de forma a reduzir os tempos de carga do sistema.

## 5.2 IMPLEMENTAÇÃO DE UM ALGORITMO

O uso de nossa AESL pode ser exemplificado através da implementação de um algoritmo específico de simulação lógica. O desempenho da AESL depende, em grande parte, da escolha do algoritmo. A fim de facilitar o processo de análise adotamos um algoritmo simples. O sistema digital é modelado a partir de portas lógicas tradicionais, com dois valores lógicos e atraso de transição. O algoritmo utilizado é baseado em tabelas e orientado a eventos, de passo único e com o fluxo de tempo efetuado por um laço $\Delta T$. Com este modelo todos os estágios da AESL são utilizados ocorrendo,

inclusive, o cancelamento de eventos.

### 5.2.1 Estrutura de dados

A base de um algoritmo de simulação é a ED ( estrutura de dados ) que descreve o sistema e o seu estado atual. As entidades que compoem a ED estão distribuídas nas oito memórias da AESL.

Inicialmente consideramos um máximo de 128 K portas lógicas ( 1 K = 1024 ). O identificador do elemento deve, portanto, ter 17 bits. Para os endereços de memória adotamos uma largura de 24 bits que corresponde a uma memória máxima de 16 MW ( 1 M = 1024 K e 1 W = 16 bits ). Veremos, posteriormente, que esta dimensão é mais que suficiente. O aumento do número de elementos está limitado à largura máxima do identificador do mesmo e não à memória máxima enderecável.

#### 5.2.1.1 Memória_de_eventos

Esta memória contém o laço $\Delta T$ e todos os eventos programados para o sistema em simulação. A fim de facilitar o gerenciamento de memória os eventos de um mesmo tempo são agrupados em blocos de tamanho fixo. O bloco constitui a área de memória mínima no caso de alocação/liberação. Os blocos de um mesmo tempo são encadeados entre si e acessados a partir do laço $\Delta T$. A figura 5.2 apresenta a composição de um bloco e seus eventos.

```
BLOCO   =>  BLOCO.TEMPO          tempo associado aos eventos do bloco
            BLOCO.NRO_EVENTOS    número n de eventos deste bloco
 (n vezes)  BLOCO.EVENTO         n eventos componentes do bloco
            BLOCO.PROX_BLOCO     endereço do próximo bloco para mesmo tempo

EVENTO  =>  EVENTO.VAL           flag de validade do evento
            EVENTO.ENCAD         flag de encadeamento do evento com outros do mesmo elemento
            EVENTO.VALOR         novo valor de saida para o elemento
            EVENTO.ELEMENTO      elemento que recebe novo valor
            EVENTO.PROX_EVENTO_ELEMENTO   endereço do próximo evento para o elemento
```

Figura 5.2 Estrutura de um bloco

Considerando a largura da palavra adotada ( 16 bits ), uma possível organização da memória_de_eventos seria a da figura 5.3.

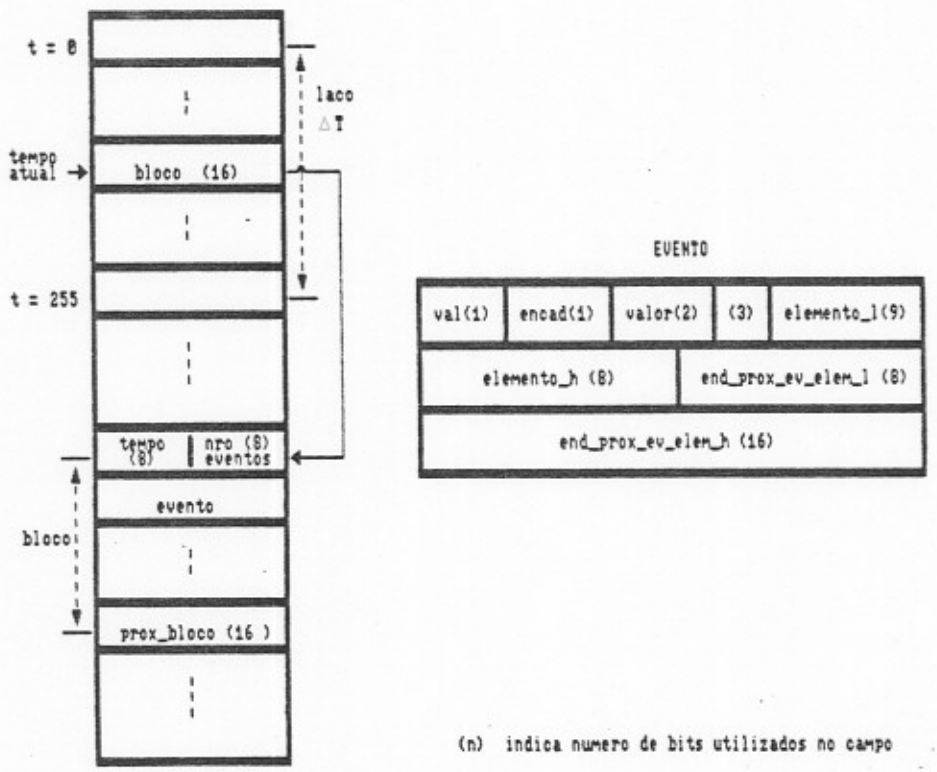



Figura 5.3 Organização da memória_de_eventos

Nesta memória identificam-se, claramente, duas áreas: o laço ΔT e os blocos de eventos.

O laço ΔT é formado por 256 palavras, correspondentes às 256 unidades de tempo. Cada palavra possui um ponteiro para um bloco, caso haja algum evento programado no tempo correspondente. O número máximo de blocos é 64 K ( 16 bits ).

Os blocos são constituidos por conjuntos de eventos para um mesmo tempo. O tamanho do bloco é fixo podendo ser escolhido nas potências de 2 entre 32 e 256 W. A limitação do tamanho é dada pelos 8 bits do número de eventos por bloco. Para facilitar o algoritmo cada bloco possui a informação de seu tempo associado ( 8 bits ) no laço ΔT. Os blocos de um mesmo tempo podem encadear-se através de um ponteiro residente na última palavra do bloco ( 16 bits ). Os eventos de um mesmo elemento encadeiam-se através de um ponteiro ( 24 bits ). Empregamos 2 bits para o valor lógico ( era necessário apenas um ) para facilitar a evolução para uma lógica de 3 ( 0, 1 e X ) ou 4 ( 0, 1, X e Z ) valores. Os três bits restantes na primeira palavra do evento podem ser utilizados para aumentar o número de elementos ou utilizar uma lógica com maior número de valores. Em qualquer das duas opções deveria ser redefinido o tamanho máximo das memórias da AESL.

O tamanho da memória_de_eventos pode ser estimado, partindo-se da atividade do sistema digital em simulação, através da expressão:

Nro de eventos = Nro de elementos * atividade

Assumindo uma atividade máxima de 15% ( posteriormente justificada ) chegamos a 19.661 eventos para 128 K portas lógicas. Considerando que cada evento utiliza 3 palavras o tamanho máximo da memória de eventos fica em 58.983 palavras. As duas palavras extras por bloco de eventos ( considerando o bloco mínimo de 32 palavras ) e as 256

palavras ao laço $\Delta T$ levam esta memória a 64 KW.

### 5.2.1.2 Memória_de_eventos_livres

As informações sobre a área disponível na memória_de_eventos ficam na memória_de_eventos_livres. Para cada tempo ao laço $\Delta T$ é armazenado o endereço do próximo evento livre, caso exista, no bloco associado àquele tempo. Uma lista de blocos livres é responsável pelo controle das liberações e alocações de blocos. A figura 5.4 mostra as duas estruturas que compoem a memória_de_eventos_livres.

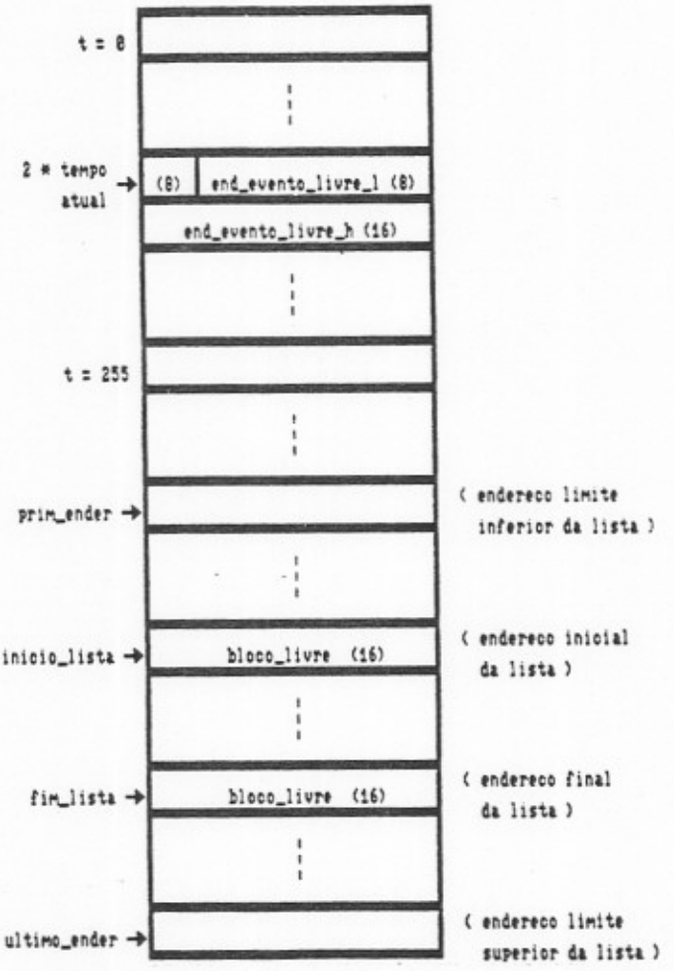



Figura 5.4 Organização da memória_de_eventos_livres

Conhecendo-se o número máximo de eventos e supondo um tamanho mínimo de bloco que acomode 10 eventos ( ou seja 1 bloco = 32 palavras ) o número máximo de blocos necessários fica em 1.966. Acrescentando-se as 512 palavras gastas nos endereços de eventos livres chegamos a 2.478 palavras. Consideramos, então, para esta memória 2,5 KW.

### 5.2.1.3 Memória_de_eventos_por_elemento

A programação de um novo evento para um elemento exige a verificação dos eventos pendentes para o mesmo. Uma forma eficiente de efetuar este acesso é encadear os eventos de um mesmo elemento [WAG84a]. O endereço do primeiro evento para o elemento é armazenado na memória_de_eventos_por_elemento conforme mostrado na figura 5.5. Um flag de validade indica se não há eventos válidos programados para o elemento. O número do elemento é utilizado para calcular o endereço na memória_de_eventos_por_elemento onde se encontram estas informações. Esta memória deve ter 256 KW visto que a cada elemento correspondem duas palavras.

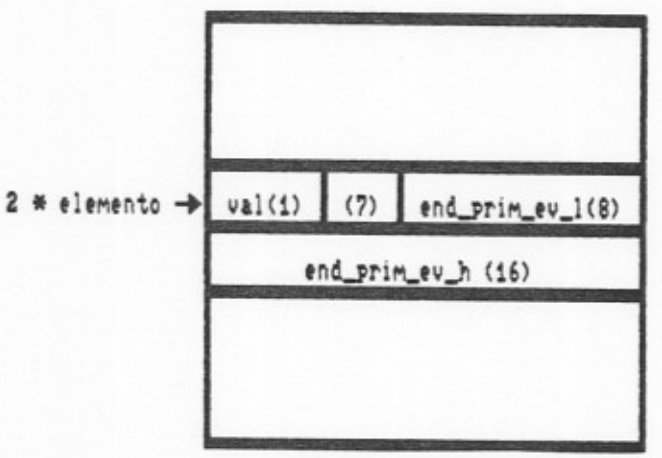



Figura 5.5 Organização da memória_de_eventos_por_elemento

5.2.1.4 Memória_de_conexões

Esta memória contém a lista de conexões de saída do elemento. A partir do número do elemento obtem-se o tamanho da lista e o seu endereço. Cada elemento da lista possui o índice da posição da entrada que sofreu alteração de valor. A figura 5.6 mostra esta estrutura.

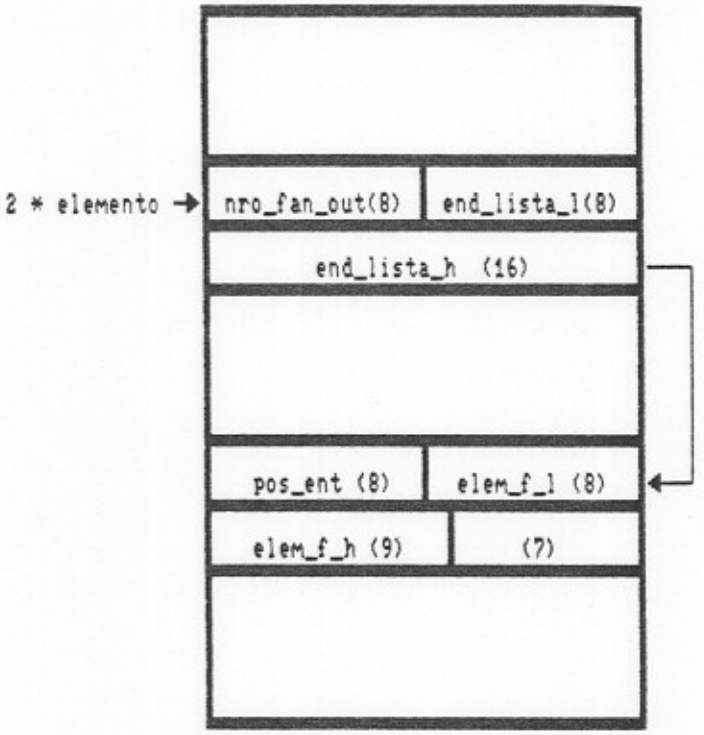



Figura 5.6 Organização da memória_de_conexões

O número máximo de elementos nesta lista ficou limitado em 256 ( 8 bits ). Para cada elemento são dispendidas duas palavras para o tamanho e o endereço da lista e duas para cada componente da lista. Considerando que o número médio de saída é aproximadamente 2,5 cada elemento ocupa, em média, 7 palavras de memória. Para 128 K elementos correspondem, então, 896 kW de memória.

### 5.2.1.5 Memória_de_estados

O estado atual do sistema em simulação está armazenado na memória_de_estados. A partir do número do elemento é possível obter o valor de sua saída, de suas entradas e verificar se a saída do elemento está sendo monitorada. O valor das entradas pode formar uma lista ( número de entradas ) 8 ) ou estar logo após o valor da saída do elemento conforme mostrado na figura 5.7.

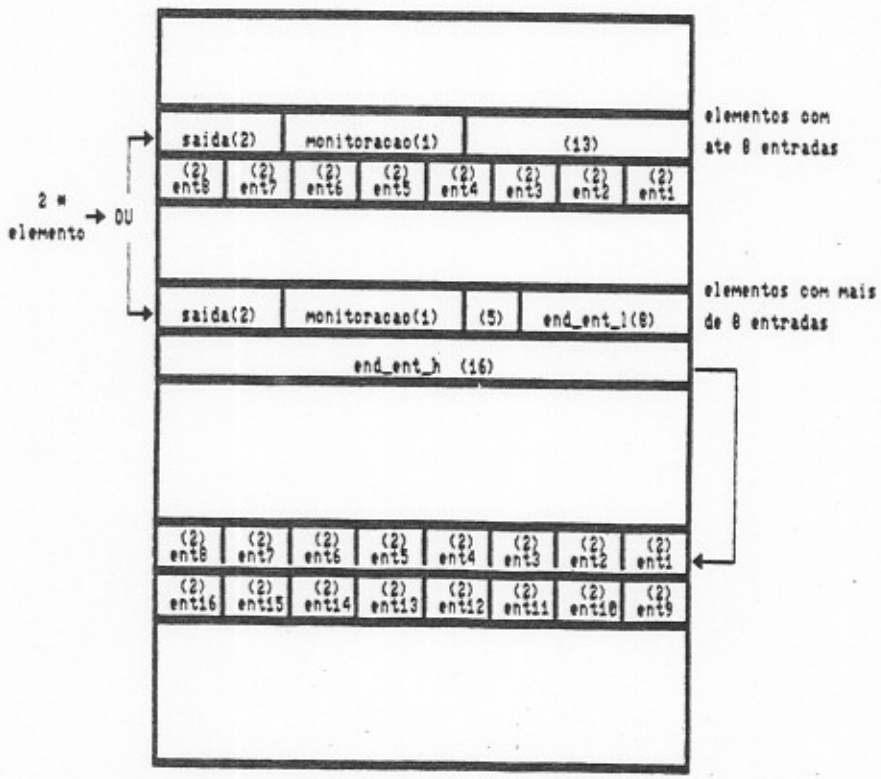



Figura 5.7 Organização da memória_de_estados

O número de entradas ( presente na memória de funções ) permite definir, antecipadamente, qual o formato das duas palavras iniciais de cada elemento, na memória_de_estados. O número médio de entradas por elemento determinará a dimensão necessária à esta memória. Considerando que o número de entradas seja sempre inferior a 8, 256 KW são suficientes para armazenar o estado do sistema. Todavia,

devemos possibilitar a existência de elementos com mais de oito entradas. Adotamos, então uma memória de 275 KW ( 5 % dos elementos com até 24 entradas ).

5.2.1.6 Memória_de_funções

Esta memória contém o código da função do elemento e o número de entradas que o mesmo utiliza, conforme apresentado na figura 5.8.

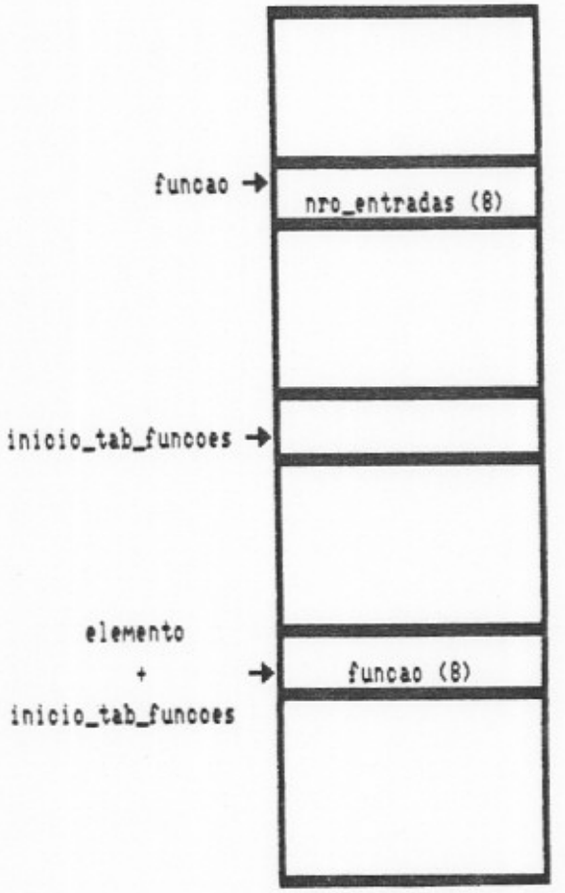



Figura 5.8 Organização da memória_de_funções

A palavra desta memória possui apenas 8 bits. O número de entradas associado à função ( máximo de 255 ) está na parte inferior da memória. A função é acessada a partir do número do elemento e do endereço inicial da tabela de

funções dentro da memória. O número de bytes necessários é igual à soma do número de elementos e do número de funções, ou seja: 128 KB + 256 B ( aproximadamente 64 KW ).

### 5.2.1.7 Memória_de_avaliação

As tabelas de avaliação dos elementos estão nesta memória. O índice de acesso às tabelas é calculado concatenando-se a função do elemento com o valor de até seis entradas do mesmo. Se o elemento possuir mais de seis entradas é utilizada uma nova tabela e a concatenação passa a ser de cinco entradas com o valor obtido na tabela anterior. O processo é repetido até se esgotarem as entradas do elemento. A largura da palavra é de apenas dois bits conforme mostrado na figura 5.9.

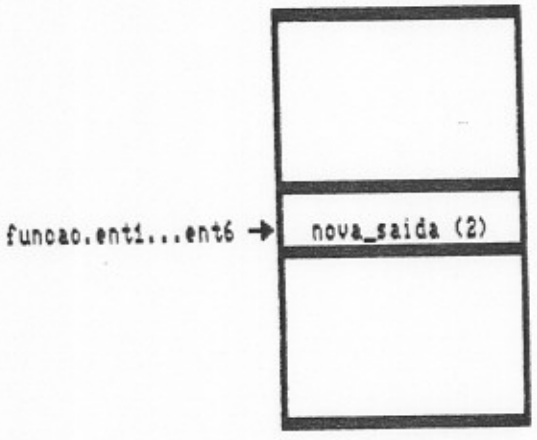



Figura 5.9 Organização da memória_de_avaliação

O número de 6 entradas por tabela foi escolhido de forma a limitar as dimensões desta memória. Para 256 funções e 6 entradas temos 1 M palavras de 2 bits que equivale à 128 KW. Caso tivéssemos adotado 8 entradas por tabela o número de palavras de 2 bits seria de 16 M equivalendo a 2 MW. O número médio de entradas por elemento não justifica este desperdício de memória com conseqüente aumento no tempo de carga das tabelas. Se um elemento possui mais de seis

entradas são utilizadas n tabelas distintas e consecutivas ( onde n é igual ao número de entradas restantes dividido por 5 ) . Desta forma o código das n funções seguintes não pode ser utilizado, reduzindo-se o número de funções disponíveis.

### 5.2.1.8 Memória_de_atrasos

Esta memória contém os atrasos associados às transições dos sinais de saída dos elementos. Como o tamanho do laço $\Delta T$ é de 256 unidades de tempo e não há uma lista de overflow, devido à característica circular do laço, o valor do atraso pode ter apenas oito bits. O modelo de atraso de transição e o uso de uma lógica de dois valores associam a cada elemento dois atrasos. O acesso ao valor do atraso é feito a partir do número do elemento conforme mostrado na figura 5.10. Uma memória de 128 KW ( 1 W = 2 atrasos ) é suficiente para o modelo adotado.

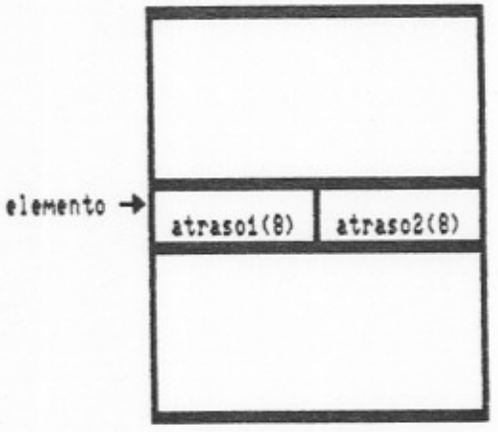



Figura 5.10 Organização da memória_de_atrasos

### 5.2.1.9 Ocupação das Memórias

A disposição da ED nas diversas memórias da AESL considerando a área ocupada está resumida na tabela 5.1.

Esta tabela pressupõe todas as memórias com sua capacidade máxima sendo utilizada.

Tabela 5.1 Distribuição da ED nas memórias da AESL

| MEMÓRIA | PALAVRA | | NÚMERO TOTAL DE BITS | |
|---|---|---|---|---|
| | QUANTIDADE | LARGURA | UTILIZADOS | SEM UTILIZAÇÃO |
| Eventos | 64 K | 16 bits | 951.708 | 96.868 |
| Eventos livres | 2,5 K | 16 bits | 37.600 | 3.360 |
| Eventos por elemento | 256 K | 16 bits | 3.276.800 | 917.504 |
| Conexões | 896 K | 16 bits | 12.386.304 | 2.293.760 |
| Estados | 275 K | 16 bits | 2.853.528 | 1.652.062 |
| Funções | 128,25 K | 8 bits | 1.050.624 | 0 |
| Avaliação | 1 M | 2 bits | 2.097.152 | 0 |
| Atrasos | 128 K | 16 bits | 2.097.152 | 0 |
| T O T A L | | | 24.750.868 | 4.963.554 |

O número médio de bits necessários ( utilizados + sem utilização ) para cada uma das 128 K portas lógicas é de 226 que corresponde a cerca de 28 bytes. Algumas das memórias possuem um considerável número de bits sem utilização o que se justifica por duas razões: acesso direto às informações ( isto é, sem envolver complexas manipulações de bits ) e previsão para o emprego de algoritmos mais complexos.

### 5.2.2 Processo padrão

A fim de padronizar a descrição do algoritmo nos diversos estágios criou-se um processo chamado "processo

padrão". Este processo busca as informações do estágio anterior num buffer de entrada e escreve os resultados para o estágio seguinte num buffer de saída. O término dos dados de entrada é marcado por um sinal de entrada chamado fim_eventos. Após encerrar o processamento do último dado o estágio seguinte é avisado através de um sinal de saída também chamado fim_eventos. O processo padrão aplica-se a todos os estágios intermediários da AESL. O primeiro e último estágio possuem processos derivados do padrão. A figura 5.11 descreve o processo padrão.

```
Repita  Repita  Repita  Lê estado do buffer de entrada
                        Lê entrada fim_eventos
                Até que estado buffer entrada <> vazio  ou
                        entrada fim_eventos = verdadeiro
                Se estado buffer entrada <> vazio
                Então   saída fim_eventos <- falso
                        Lê buffer de entrada
                        Processa dados
                        Repita Lê estado do buffer de saída
                        Até que estado buffer saída <> cheio
                        Escreve no buffer de saída
        Até que entrada fim_eventos = verdadeiro
        saída fim_eventos <- verdadeiro
Até que haja uma interrupção do processador mestre
```

Figura 5.11 Processo padrão

Os buffers de entrada e saída de alguns estágios intermediários foram divididos em dois para facilitar o algoritmo. Seus processos, todavia, sofrem pequenas alterações em relação ao padrão.

### 5.2.3 Processo por estágio

A fim de fornecer maiores subsídios para a compreensão das etapas seguintes deste trabalho, descreveremos o algoritmo do estágio busca_e_atualiza_entradas/busca_função. Para tanto é necessário definir o formato dos dados nas interfaces do estágio. O buffer de entrada foi definido de acordo com a figura 5.12. O campo VALOR contém o valor lógico associado ao evento. Este será atribuído à entrada

indicada por POS_ENT no elemento apontado pela concatenação de ELEM_F_H e ELEM_F_L ( um dos elementos da lista de conexões de saída do elemento associado ao evento ).

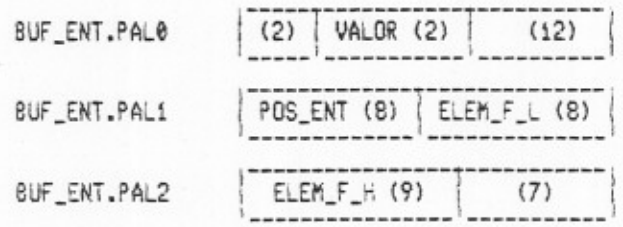



Figura 5.12 Buffer de entrada

O buffer de saída foi dividido em dois grupos conforme mostrado na figura 5.13. O campo SAÍDA contém o valor lógico atual da saída do elemento identificado por ELEM_F_H e ELEM_F_L. A função lógica do elemento e o seu número de entradas estão nos campos FUNÇÃO e NRO_ENT, respectivamente.

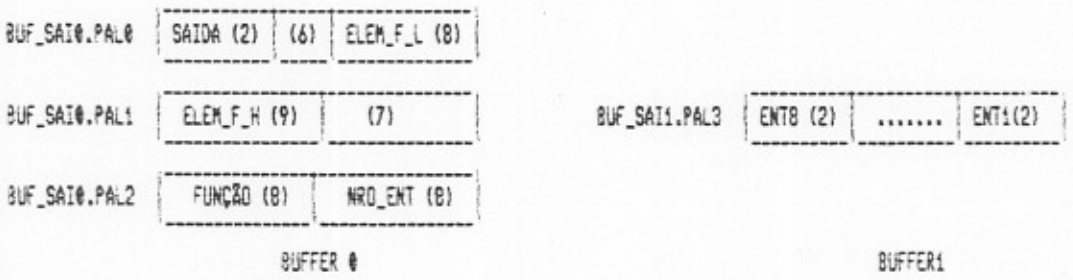



Figura 5.13 Buffers de saída

Os campos ENT8...ENT1 do BUFFER1 contém o valor das primeiras oito entradas do elemento. Caso o número de entradas seja maior que oito este buffer será utilizado quantas vezes for necessário. O algoritmo do estágio, em português estruturado está na figura 5.14.

```
Repita  Repita Repita  Ler ESTADO_BUF_ENT
                       Ler ENT_FIM_EVENTOS
               Até que ESTADO_BUF_ENT () VAZIO   ou
                       ENT_FIM_EVENTOS = VERDADEIRO
               Se  ESTADO_BUF_ENT () VAZIO
               Então  Início
                      SAI_FIM_EVENTOS (- FALSO
                      PAL0_ENT (- BUF_ENT.PAL0                          lê valor associado ao evento
                      PAL1_ENT (- BUF_ENT.PAL1                          lê posição de entrada e nro
                      PAL2_ENT (- BUF_ENT.PAL2                           do elemento fan out
                      POS_ENT (- PAL1_ENT.POS_ENT
                      END_ELEM (- CONCAT(PAL2_ENT.ELEM_F_H,PAL1_ENT.ELEM_F_L)  gera endereço do elemento
                      END_FUNC (- END_ELEM + INICIO_TAB_FUNÇÕES             calcula endereco da função
                      FUNÇÃO (- MEM_FUNÇÕES ( END_FUNC )                busca função
                      NRO_ENT (- MEM_FUNÇÕES ( FUNÇÃO )                 busca nro de entradas
                      PAL2_SAI (- CONCAT(FUNÇÃO,NRO_ENT)                concatena função,nro de entradas
                      END_ELEM (- 2 * END_ELEM
                      PALAVRA1 (- MEM_ESTADOS ( END_ELEM )              busca saída atual
                      PAL0_SAI (- CONCAT(PALAVRA1.SAÍDA,PAL1_ENT.ELEM_F_L) concatena saida atual e nro do
                      Repita  Ler ESTADO_BUF_SAI0                       elemento (byte - significativo)
                      Até que ESTADO_BUF_SAI0 () CHEIO
                      BUF_SAI0.PAL0 (- PAL0_SAI                         envia nro do elemento
                      BUF_SAI0.PAL1 (- PAL2_ENT
                      BUF_SAI0.PAL2 (- PAL2_SAI                         envia função e nro de entradas
                      Se NRO_ENT ) 8
                      Então Início
                            END_ELEM (- END_ELEM + 1                    busca endereço da lista
                            PALAVRA2 (- MEM_ESTADOS ( END_ELEM )         de entradas
                            END_ENT (- CONCAT(PALAVRA2,PALAVRA1.END_ENT_L)
                            Fim
                      Senão END_ENT (- END_ELEM + 1                     não há lista de entradas
                      CONT_ENT (- 0
                      ATUALIZAÇÃO (- FALSO
                      Enquanto CONT_ENT ( NRO_ENT                       verifica se acabaram as entradas
                      Faça Início
                           PAL3_SAI (- MEM_ESTADOS(END_ENT)
                           CONT_ENT (- CONT_ENT + 8
                           Se ATUALIZAÇÃO = FALSO e
                              POS_ENT (= CONT_ENT
                           Então Início
                                 PAL3_SAI.(POS_ENT) (- PAL0_ENT.VALOR   atualiza entrada de PAL3_SAI
                                 ATUALIZAÇÃO (- VERDADEIRO               apontada por POS_ENT
                                 MEM_ESTADOS(END_ENT) (- PAL3_SAI
                                 Fim
                           Repita Ler ESTADO_BUF_SAI1
                           Até que ESTADO_BUF_SAI1 () CHEIO
                           BUF_SAI1.PAL3 (- PAL3_SAI
                           END_ENT (- END_ENT + 1                       envia entradas de 8 em 8
                           Fim
                      Fim
      Até que ENT_FIM_EVENTOS = VERDADEIRO
      SAI_FIM_EVENTOS (- VERDADEIRO
Até que haja uma interrupção do mestre
```

Figura 5.14 Procedimento busca_e_atualiza_entradas/busca_função

## 5.3 SIMULAÇÃO DA AESL

Simular é construir um modelo e exercitá-lo de forma a estudar seu comportamento. O modelo de simulação de nossa AESL é orientado a processos baseando-se na implementação do algoritmo apresentado no item 5.2. Cada estágio da AESL corresponde a um processo com determinada duração. O estudo dos tempos envolvidos em cada processo é importante para uma análise quantitativa do modelo. Para este estudo é necessário definir o processador que executará o algoritmo em cada estágio.

### 5.3.1 Processador Ideal

A escolha de um determinado processador influencia enormemente o desempenho de uma arquitetura. A evolução dos processadores disponíveis é constante e a adoção de um específico traria alguns inconvenientes para o modelo como:

- demasiado aumento na granularidade;
- especificidade dos resultados obtidos.

Adotamos então um "processador ideal" que incorpora características de vários processadores existentes, além daquelas que o levam a denominar-se ideal. Numa futura implementação, o modelo de simulação seria facilmente adaptado a um processador específico bastando estabelecer suas diferenças em relação ao processador ideal e proceder às alterações no modelo.

O processador ideal apresenta as seguintes características:

- palavra de 16 bits;

- infinitos registradores de dados de 16 bits;
- infinitos registradores de endereço de 32 bits;
- endereçamento indireto à memória de dados via registradores de endereço;
- código de instrução sempre com uma palavra;
- tempo idêntico na execução de qualquer instrução.

Denominamos T o tempo que o processador leva para executar uma instrução de transferência entre a memória de dados e um de seus registradores. Segundo nossa definição este tempo T é válido para todas as demais instruções do processador ideal.

### 5.3.2 Construção do modelo

A linguagem utilizada na descrição do modelo foi GPSS (General Purpose Simulation System). Os fundamentos desta linguagem extrapolam os objetivos deste trabalho podendo ser encontrados em [GRE72] [ODO79]. O emprego de uma linguagem convencional de programação ( Pascal concorrente, por exemplo ) dificultaria a obtenção de estatísticas. Acrescente-se a isto o fato de não haver disponível no CPGCC, durante a elaboração deste trabalho, nenhuma outra linguagem de simulação.

A figura 5.15 mostra a estrutura do modelo GPSS. O processador, em cada estágio, é representado por uma transação. Um valor de tempo é atribuído a cada instrução do algoritmo. Instruções que não interagem com outros estágios nem influenciam no fluxo do programa são mapeadas como um bloco ADVANCE do GPSS.

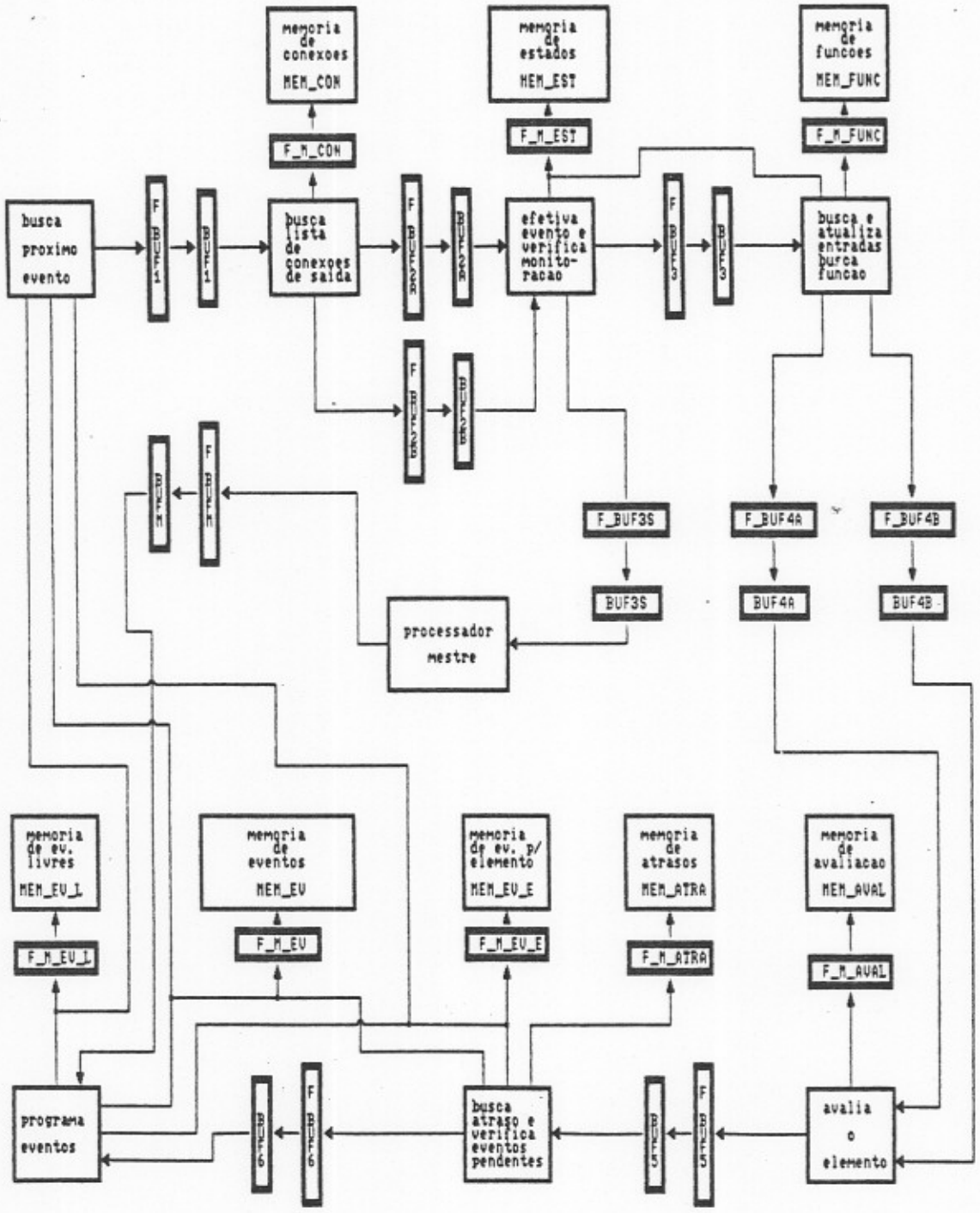



Figura 5.15 Modelo em GPSS da AESL proposta

Todas as variáveis do algoritmo correspondem a registradores do processador padrão. Instruções mais complexas são desmembradas em suas equivalentes no processador padrão.

Para efeito de regularidade, as memórias, com ou sem concorrência no acesso, foram mapeadas como FACILITIES. A cada memória existe uma fila associada para coleta de estatísticas.

Os buffers entre estágios são representados por STORAGES. O tamanho dos buffers normalmente é unitário. Dimensionar os buffers como infinitos e verificar os tamanhos máximos alcançados é uma forma de identificar gargalos na AESL. Para cada buffer existe uma fila associada. A figura 5.15 mostra a estrutura do modelo GPSS.

O estágio busca_e_atualiza_entradas/busca_função, por exemplo, acessa duas memórias e utiliza três buffers. O acesso à memória_de_estados ( MEM_EST ) é compartilhado com o estágio efetiva_evento_e_verifica_monitoração existindo uma fila ( F_M_EST ) onde um estágio aguarda enquanto a memória está sendo utilizada pelo outro. Os dados de entrada são lidos num buffer ( BUF3 ) cujo estado ( vazio / não vazio ) pode ser testado. Os dados de saída são escritos em dois buffers ( BUF4A e BUF4B ) correspondentes aos apresentados na figura 5.13. Cada buffer possui uma fila associada ( F_BUF4A e F_BUF4B ) para coleta de estatísticas. Um novo dado só é escrito no buffer quando ele não está cheio, caso contrário o estágio fica aguardando a liberação de área no buffer.

A passagem algoritmo -> programa GPSS é feita em três etapas. Primeiramente atribui-se atrasos às instruções do algoritmo. A figura 5.16 exemplifica esta etapa utilizando o algoritmo do estágio busca_e_atualiza_entradas/busca_função.

```
Repita  Repita Repita  Ler ESTADO_BUF_ENT                1T *
                       Ler ENT_FIM_EVENTOS               1T
               Até que ESTADO_BUF_ENT () VAZIO   ou      2T
                       ENT_FIM_EVENTOS = VERDADEIRO      2T
               Se  ESTADO_BUF_ENT () VAZIO               2T
               Então  Início
               1T     SAI_FIM_EVENTOS (- FALSO
               1T     PAL0_ENT (- BUF_ENT.PAL0                          lê valor associado ao evento
               1T     PAL1_ENT (- BUF_ENT.PAL1                          lê posição de entrada e nro
               1T     PAL2_ENT (- BUF_ENT.PAL2                           do elemento fan out
               2T     POS_ENT (- PAL1_ENT.POS_ENT
               5T     END_ELEM (- CONCAT1(PAL2_ENT.ELEM_F_H,PAL1_ENT.ELEM_F_L)  gera endereço do elemento
               2T     END_FUNC (- END_ELEM + INICIO_TAB_FUNÇÕES                  calcula endereço da função
               1T     FUNÇÃO (- MEM_FUNÇÕES ( END_FUNC )                busca função
               1T     NRO_ENT (- MEM_FUNÇÕES ( FUNÇÃO )                 busca nro de entradas
               5T     PAL2_SAI (- CONCAT(FUNÇÃO,NRO_ENT)                concatena função,nro de entradas
               1T     END_ELEM (- 2 * END_ELEM
               1T     PALAVRA1 (- MEM_ESTADOS ( END_ELEM )              busca saída atual
               5T     PAL0_SAI (- CONCAT(PALAVRA1.SAÍDA,PAL1_ENT.ELEM_F_L)  concatena saída atual e nro do
               1T     Repita  Ler ESTADO_BUF_SAI0                       elemento (byte - significativo)
               2T     Até que ESTADO_BUF_SAI0 () CHEIO
               1T     BUF_SAI0.PAL0 (- PAL0_SAI                         envia nro do elemento
               1T     BUF_SAI0.PAL1 (- PAL2_ENT
               1T     BUF_SAI0.PAL2 (- PAL2_SAI                         envia função e nro de entradas
               2T     Se NRO_ENT ) 8
                      Então Início
               1T           END_ELEM (- END_ELEM + 1                    busca endereço da lista
               1T           PALAVRA2 (- MEM_ESTADOS ( END_ELEM )         de entradas
               5T           END_ENT (- CONCAT(PALAVRA2,PALAVRA1.END_ENT_L)
               1T           Fim
               2T     Senão END_ENT (- END_ELEM + 1                     não há lista de entradas
               1T     CONT_ENT (- 0
               1T     ATUALIZAÇÃO (- FALSO
               2T     Enquanto CONT_ENT ( NRO_ENT                       verifica se acabaram as entradas
                      Faça Início
               1T          PAL3_SAI (- MEM_ESTADOS(END_ENT)
               2T          CONT_ENT (- CONT_ENT + 8
               2T          Se ATUALIZAÇÃO = FALSO e
               2T             POS_ENT (= CONT_ENT
                           Então Início
               5T               PAL3_SAI.(POS_ENT) (- PAL0_ENT.VALOR    atualiza entrada de PAL3_SAI
               1T               ATUALIZAÇÃO (- VERDADEIRO                apontada por POS_ENT
               1T               MEM_ESTADOS(END_ENT) (- PAL3_SAI
                                Fim
               1T          Repita Ler ESTADO_BUF_SAI1
               2T          Até que ESTADO_BUF_SAI1 () CHEIO
               1T          BUF_SAI1.PAL3 (- PAL3_SAI
               1T          END_ENT (- END_ENT + 1                       envia entradas de 8 em 8
               1T          Fim
                     Fim
          Até que ENT_FIM_EVENTOS = VERDADEIRO   2T
          SAI_FIM_EVENTOS (- VERDADEIRO          1T
Até que haja uma interrupção do mestre                                  (*) tempo gasto na execução da instrução
```

Figura 5.16 Tempos no procedimento busca_e_atualiza_entradas/busca_função

Logo após monta-se um fluxograma, o mais próximo possível do algoritmo, considerando-se as características do GPSS e os parâmetros do sistema em simulação. A figura 5.17 apresenta o fluxograma gerado para o algoritmo visto anteriormente.

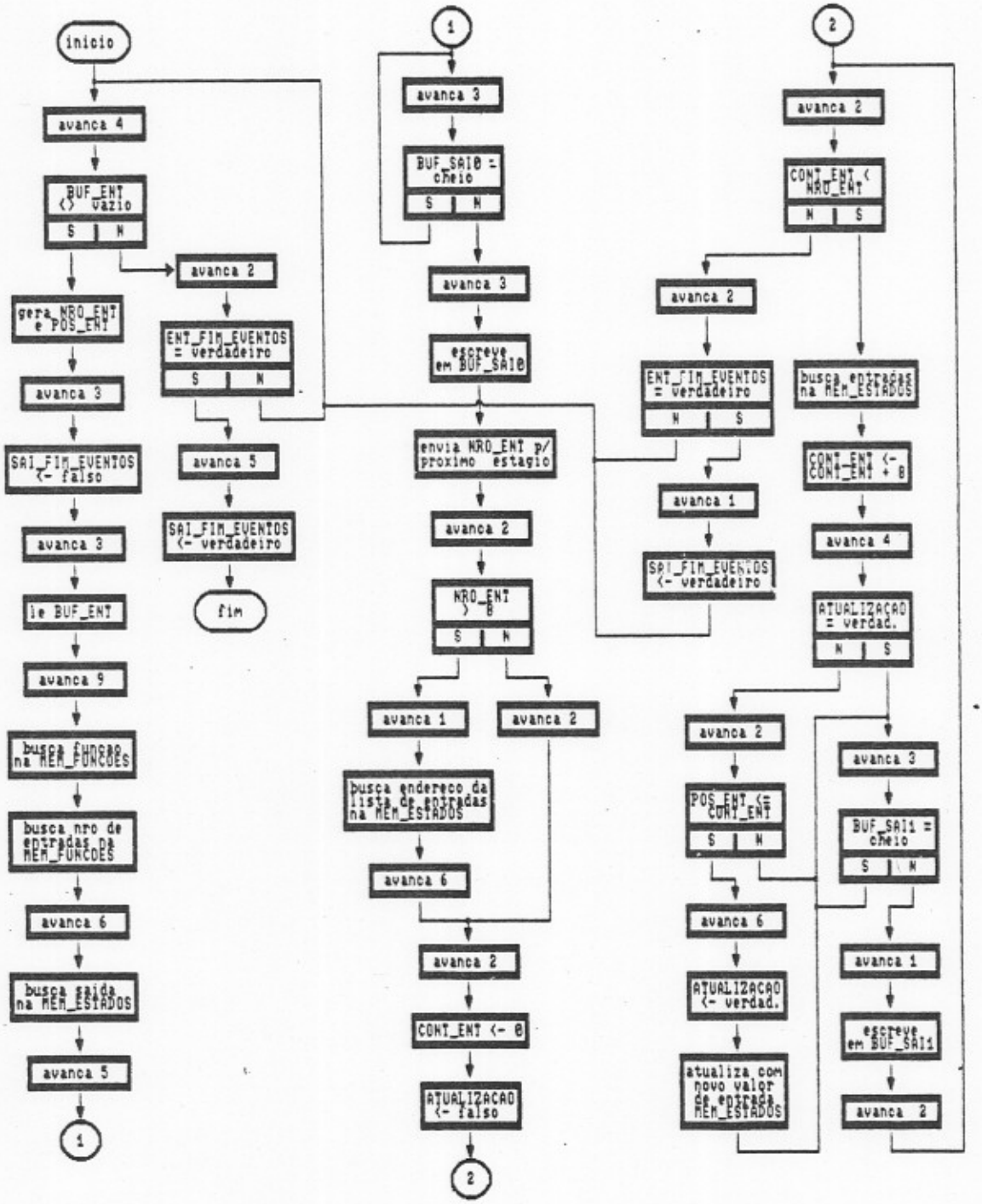



Figura 5.17 Fluxograma para implementação do algoritmo

Os parâmetros de simulação são classificados em dois grupos: estáticos e dinâmicos [WAG83].

Os parâmetros estáticos refletem a topologia do circuito, propriedades das portas e decisões na implementação do algoritmo. São eles:

FAN_MIN - número mínimo de portas conectadas a uma saída;
FAN_MAX - número máximo de portas conectadas a uma saída;
ENT_MIN - número mínimo de entradas de cada porta;
ENT_MAX - número máximo de entradas de cada porta;
TAM_BLO - tamanho do bloco de eventos utilizado pelo algoritmo.

Os parâmetros dinâmicos refletem o número de portas avaliadas durante a simulação ou para as quais há eventos programados. São eles:

EV_VAL - porcentagem de eventos válidos em relação a todos os eventos programados;
EV_PEND - porcentagem de elementos que, ao serem avaliados, já têm eventos programados;
EV_ENCAD - porcentagem de eventos que possuem um evento encadeado;
MONIT - porcentagem de portas que tem sua saída monitorada.

Os parâmetros mantém relações que refletem a dinâmica da simulação. O número de eventos a programar em cada tempo , por exemplo, é calculado através de uma equação. Seja:

EV - número de eventos válidos programados a cada tempo ( que não serão posteriormente cancelados );

EI - número de eventos programados a cada tempo que serão posteriormente cancelados;

FAN_MED - ( FAN_MIN + FAN_MAX ) / 2

EV_PROG - número de eventos a programar em cada tempo.

Considerando que EV se conserva no decorrer da simulação:

( EV / nro de portas lógicas avaliadas a cada tempo) = ( 1 / FAN_MED )

como EV = EV_VAL * ( EV + EI ) e EV_PROG = EV + EI

concluímos que

( EV_PROG / nro de portas lógicas avaliadas a cada tempo ) = ( 1 / ( EV_VAL * FAN_MED ))

Em [WON86] foram apresentadas várias estatísticas sobre o processo de simulação. Na ausência de outras referências utilizaremos várias de suas conclusões na construção de nosso modelo. Durante uma média de 86 % do tempo de simulação não há eventos a serem efetivados. Considerando somente os tempos em que há pelo menos um evento programado a média de eventos simultâneos é de 18,6 para cinco circuitos com uma média de 4300 transistores ( aproximadamente 1.100 portas lógicas ). A distribuição destes eventos, segundo nossas conclusões ao analisar os gráficos apresentados, possui uma característica exponencial. A atividade máxima do sistema pode ser calculada dividindo-se o tamanho máximo alcançado pela lista de eventos pelo número total de portas lógicas do mesmo. Os parâmetros de simulação foram adotados de acordo com a tabela 5.2.

Tabela 5.2 Parâmetros de simulação

| PARÂMETRO | VALOR | OBSERVAÇÕES |
|---|---|---|
| FAN_MIN | 2 | número médio de portas |
| FAN_MAX | 3 | conectadas a uma saída é 2,5 |
| ENT_MIN | 2 | 5 % dos elementos possuem |
| ENT_MAX | 4 | entre 10 e 12 entradas |
| TAM_BLO | 32 | capacidade para 10 eventos |
| EV_VAL | 95% | o sistema não está operando |
| EV_ENCAD | 5% | em seu limite de frequência |
| EV_PEND | 15% | nível de atividade do sistema é elevado |
| MONIT | 2% | o sistema é grande o suficiente para ter poucas saídas monitoradas em relação às existentes |

Para alguns parâmetros como EV_VAL, EV_ENCAD, EV_PEND e MONIT não foram encontradas referências na literatura. Seus valores foram estimados a partir das informações disponíveis. O parâmetro EV_PEND é função da atividade do circuito e da porcentagem de eventos encadeados ( EV_ENCAD ). Assumimos, pessimistamente, o valor da atividade para EV_PEND, na falta de maiores dados.

A última etapa na construção do modelo consiste em escrever o programa GPSS. Cada estágio utiliza um gerador de números aleatórios que, juntamente com os parâmetros da simulação, determina o caminho a percorrer no algoritmo. Os geradores de números aleatórios fornecem valores entre 0 e 999. As decisões, em cada estágio, são tomadas comparando-se um valor constante, calculado a partir de um ou mais parâmetros de simulação, com o valor de um gerador de números aleatórios. A programação de um evento, por exemplo, deve

ser efetuada de forma a satisfazer a equação vista anteriormente. A expressão 2000/(EV_VAL*(FAN_MAX+FAN_MIN)/1000) gera um número constante entre 0 e 999 que comparado com um número aleatório permite decidir pela programação ou não de um evento, a cada porta lógica avaliada. Parâmetros que necessitam ser enviados ao estágio seguinte ( número de entradas, por exemplo ) utilizam como veículo uma cópia da transação processador. A figura 5.18 apresenta o programa GPSS para o estágio busca_e_atualiza_entradas/busca_função.

```
AUX3     VARIABLE  ENT_MIN + ( (ENT_MAX - ENT_MIN + 1) * RN$5 ) / 1000
*
AUX4     VARIABLE  1 + ( P$NRO_ENT  * RN$5  / 1000 )
*
MAIS_ENT FUNCTION RN$5,D2          ; 5% dos elementos com mais de 8 entradas
         .95,0/.999,8
*
*
*************                                                  *************
*************   Estagio BUSCA E ATUALIZA ENTRADAS / BUSCA FUNCAO  *************
*************                                                  *************
*
BUF4A    STORAGE  1
BUF4B    STORAGE  1
*
         GENERATE ,,,1
         LOGIC_S FIM_EV4
L830     ADVANCE 4
         GATE_SE BUF3,L831                 ; trata evento se BUF_ENT não vazio
         ADVANCE 2
         GATE_LS FIM_EV3,L830              ; se BUF_ENT não vazio e fim eventos
         ADVANCE 5                         ; está setado finaliza
         LOGIC_S FIM_EV4
         TRANSFER ,L830
L831     ASSIGN NRO_ENT,V$AUX3             ; nro de entradas médio
         INCREMENT P$NRO_ENT,FN$MAIS_ENT   ; 5% dos elementos c/ + de 8 entradas
         ASSIGN POS_ENT,V$AUX4             ; posição da entrada
         ADVANCE 3
         LOGIC_R FIM_EV4
         ADVANCE 3
         LEAVE BUF3                        ; lê BUF_ENT
         ADVANCE 9
         TRANSFER SBR,MEM_FUNC,12
         TRANSFER SBR,MEM_FUNC,12
         ADVANCE 6
         TRANSFER SBR,MEM_EST,12
         ADVANCE 5
```

Figura 5.18 Programa GPSS para um estágio da AESL cont.

continuação

```
        QUEUE F_BUF4A
LB32    ADVANCE 3
        GATE_SNF BUF4A,LB32                ; aguarda, se BUF_SAI0  está cheio
        ADVANCE 3
        SPLIT 1,LB4A                       ; envia nro de ent. ao próx estágio
        DEPART F_BUF4A
        ENTER BUF4A                        ; escreve em BUF_SAI0
        ADVANCE 2
        TEST_LE P$NRO_ENT,8,LB38
        ADVANCE 1
        TRANSFER SBR,MEM_EST,12
        ADVANCE 6
        TRANSFER ,LB33
LB38    ADVANCE 2
LB33    ADVANCE 2
        ASSIGN CONT_ENT,0
        LOGIC_R ATUALIZ
LB34    ADVANCE 2
        TEST_GE P$CONT_ENT,P$NRO_ENT,LB37  ; verifica fim das entradas
        ADVANCE 2
        GATE_LS FIM_EV3,LB30
        ADVANCE 1
        LOGIC_S FIM_EV4
        TRANSFER ,LB30
LB37    TRANSFER SBR,MEM_EST,12
        INCREMENT CONT_ENT,8
        ADVANCE 4
        GATE_LR ATUALIZ,LB35
        ADVANCE 2
        TEST_LE P$POS_ENT,P$CONT_ENT,LB35  ; verifica atualizacao da
        ADVANCE 6                          ; entrada que variou
        LOGIC_S ATUALIZ
        TRANSFER SBR,MEM_EST,12
LB35    QUEUE F_BUF4B
LB36    ADVANCE 3
        GATE_SNF BUF4B,LB36
        ADVANCE 1
        DEPART F_BUF4B
        ENTER BUF4B                        ; envia entradas ao próx estágio
        ADVANCE 2                          ; escrevendo em BUF_SAI1
        TRANSFER ,LB34
```

Figura 5.18 Programa GPSS para um estágio da AESL

A variável AUX3 é utilizada, juntamente com um gerador de números aleatórios, para criar o número de entradas do elemento. A posição da entrada é gerada por AUX4 de forma similar. A função MAIS_ENT soma 8 entradas em 5% dos elementos de modo que seu número de entradas fica entre ENT_MIN + 8 e ENT_MAX + 8. O acesso às memórias é feito por subrotinas cuja estrutura é vista na figura 5.19.

```
MEM_EV   QUEUE  F_M_EV                    ; acesso a memoria_de_eventos
         SEIZE  MEM5
         DEPART F_M_EV
         ADVANCE 1
         RELEASE MEM5
         TRANSFER P,12,1
```

Figura 5.19 Subrotina de acesso à memória_de_eventos

### 5.3.3 Análise dos resultados

A análise dos resultados será baseada em várias simulações das quais utilizaremos somente os dados relevantes. A fim de simplificar o texto numeraremos os estágios de 1 a 7 a partir do estágio busca_próximo_evento de acordo com o fluxo dos dados.

O tamanho máximo das filas de acesso e a utilização média das memórias ( fração do tempo total em que a memória é acessada ) mostra que é mínimo o conflito na utilização deste recurso. Este fato decorre da pequena parcela que o acesso à memória representa na execução do algoritmo. No caso da implementação de um algoritmo " em hardware " teríamos situação oposta, ou seja, o gargalo do sistema seria o tempo de acesso à memória. A tabela 5.3 apresenta alguns dados obtidos na simulação.

Tabela 5.3 Estatística de acesso à memória

| MEMÓRIA | UTILIZAÇÃO MÉDIA | TAMANHO MÁXIMO DA FILA DE ACESSO | PERCENTUAL DE ACESSOS SEM CONTENÇÃO |
|---|---|---|---|
| Eventos | 0,05 | 2 | 97,48 |
| Eventos livres | 0,02 | 1 | 100,00 |
| Eventos por elemento | 0,03 | 1 | 98,41 |
| Conexões | 0,03 | 1 | 100,00 |
| Estados | 0,06 | 1 | 97,86 |
| Funções | 0,02 | 1 | 100,00 |
| Avaliação | 0,01 | 1 | 100,00 |
| Atrasos | 0,01 | 1 | 100,00 |

O tempo médio de permanência dos dados nos buffers entre estágios e o tempo médio de espera do processador na fila associada aos mesmos no modelo GPSS indicam as diferenças de velocidade entre estágios conforme tabela 5.4.

Tabela 5.4 Estatísticas de utilização dos buffers

| BUFFER | ESTÁGIOS CONECTADOS | UTILIZAÇÃO MÉDIA | TEMPO MÉDIO DE PERMANÊNCIA | TEMPO MÉDIO DE ESPERA DO PROC. |
|---|---|---|---|---|
| 1 | 1 e 2 | 0,87 | 182,36 | 153,58 |
| 2A | 2 e 3 | 0,55 | 114,06 | 0 |
| 2B | 2 e 3 | 0,86 | 71,92 | 49,15 |
| 3 | 3 e 4 | 0,89 | 74,25 | 50,68 |
| 3S | 3 e MESTRE | 0 | 13,00 | 0 |
| 4A | 4 e 5 | 0,17 | 14,31 | 1,57 |
| 4B | 4 e 5 | 0,10 | 7,79 | 0 |
| 5 | 5 e 6 | 0,17 | 13,91 | 0,03 |
| 6 | 6 e 7 | 0,05 | 9,28 | 0 |

Os valores elevados de tempo médio de permanência e tempo médio de espera na fila apontam para uma diferença de velocidade entre os estágios 1 e 2. O quadro se repete para os estágios 2-3 e 3-4 porém de forma menos acentuada.

É possível refinar o estudo de velocidades através de um artifício. Dimensionando os buffers com tamanho 1000 retiramos os gargalos de saída dos estágios e obtivemos os dados da tabela 5.5.

Tabela 5.5 Tamanho máximo dos buffers

| BUFFER | ESTÁGIOS CONECTADOS | TAMANHO MÁXIMO ALCANÇADO |
|---|---|---|
| 1 | 1 e 2 | 34 |
| 2A | 2 e 3 | 1 |
| 2B | 2 e 3 | 2 |
| 3 | 3 e 4 | 96 |
| 3S | 3 e MESTRE | 1 |
| 4A | 4 e 5 | 2 |
| 4B | 4 e 5 | 2 |
| 5 | 5 e 6 | 2 |
| 6 | 6 e 7 | 1 |

Confirmamos, então, a maior velocidade do estágio 1 em relação ao 2. Quanto aos estágios 3 e 4 há duas hipóteses prováveis. O estágio 3 é rápido demais em relação aos seguintes ou o estágio 4 é lento demais e retarda o fluxo de dados no restante da máquina. Verificamos, também, que o tempo final de simulação sofreu uma redução de apenas 1% , fato que justifica o uso de buffers unitários entre os estágios.

Para melhorar nossa análise acrescentamos ao programa de simulação alguns contadores de tempo. Em cada estágio, para cada unidade de tempo de simulação, é acumulado o tempo total de processamento, desconsiderando-se o tempo gasto na espera para leitura/escrita nos buffers. A tabela 5.6 mostra os dados obtidos para 10 unidades de tempo de simulação.

Tabela 5.6 Tempo de processamento por estágio

| TEMPO DE SIMULAÇÃO | NÚMERO DE EVENTOS | TEMPO INICIAL | TEMPO FINAL | TEMPO ESTÁG.1 | TEMPO ESTÁG.2 | TEMPO ESTÁG.3 | TEMPO ESTÁG.4 | TEMPO ESTÁG.5 | TEMPO ESTÁG.6 | TEMPO ESTÁG.7 |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 18 | 32 | 3.724 | 461 | 974 | 876 | 3.028 | 1.501 | 2.335 | 951 |
| 2 | 0 | 3.756 | 3.781 | 4 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 3.813 | 3.838 | 4 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | 0 | 3.870 | 3.895 | 4 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5 | 0 | 3.927 | 3.952 | 4 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | 0 | 3.984 | 4.009 | 4 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | 66 | 4.041 | 16.110 | 1.610 | 3.242 | 2.911 | 10.009 | 5.251 | 7.499 | 3.625 |
| 8 | 0 | 16.142 | 16.167 | 4 | 0 | 0 | 0 | 0 | 0 | 0 |
| 9 | 0 | 16.199 | 16.224 | 4 | 0 | 0 | 0 | 0 | 0 | 0 |
| 10 | 0 | 16.256 | 16.281 | 4 | 0 | 0 | 0 | 0 | 0 | 0 |

Com estes dados é possível fazer uma análise do oesempenho relativo de cada um dos estágios. O estágio 1 confirma nossas hipóteses sendo o mais rápido de todos. Os estágios 2, 3 e 7 possuem tempos próximos. O desempenho da máquina é dado pelos estágios 4, 5 e 6 cujos tempos destacam-se em relação aos demais. O gargalo desta AESL é o estágio 4 o que explica o elevado tamanho alcançado por seu buffer de entrada. Como os estágios 5, 6 e 7 possuem tempos de processamento bastante inferiores ao estágio 4 os buffers entre eles não apresentam tamanho máximo superior a 2.

A obtenção de valores absolutos para o desempenho da AESL exige que façamos algumas suposições. Consideremos que nosso processador ideal possua tempo de busca e execução de uma instrução constante e igual a 1 uS. Na prática isto significa afirmar que este seria o valor médio para o processador escolhido, considerando-se o conjunto de instruções a ser utilizado. O microprocessador Intel 8086 possui um

tempo médio de instrução em torno de 2 uS [WIL83]. No entanto, há microprocessadores mais rápidos como o National NS16032 [NAT83] e o Zilog Z8001 [ZIL82] cujos tempos de instrução margeiam o valor estimado de 1 uS.

Para o cálculo do desempenho aumentamos dez vezes o tamanho do sistema digital ficando com um valor aproximado de 43.000 transistores. Apesar deste aumento, o número de portas lógicas simulado não chega a 10% da capacidade total da máquina ( considerando 4 transistores por porta lógica, em média ). Paradoxalmente, o tamanho máximo do sistema a ser simulado é limitado pelo elevado tempo gasto em cada simulação. Para 50 unidades de tempo de simulação ( o que corresponde a 3 horas de simulação num PC XT ) obtivemos os dados da tabela 5.7.

Tabela 5.7 Desempenho da AESL

| TEMPO DE SIMULAÇÃO | NÚMERO DE EVENTOS SIMULADOS | TEMPO FINAL NO MODELO |
|---|---|---|
| 50 | 2.040 | 381.637 |

O tempo médio de processamento de um evento ficou em 187 instruções o que equivale, segundo nossas hipóteses, a 187 uS. Desta forma, temos um desempenho estimado em 5.348 ev/s ( eventos por segundo ) ou 13.370 avaliações/s considerando que cada evento gera, em média, 2,5 avaliações.

O processamento de um evento geralmente leva de 500 a 1000 intruções [GOE88]. Idealmente, com sete estágios, teríamos de 71 a 143 instruções por estágio o que é coerente com o valor encontrado.

Um acréscimo no desempenho de nossa AESL pode ser obtido com a utilização de um processador microprogramável. Microprocessadores como os da família AM29000 [ADV85]

trabalham com um tempo de microinstrução da ordem de 100 nS.

Este ganho de velocidade levaria nossa AESL a uma taxa de 53.480 ev/s ( 133.700 avaliações/s ) que pode ser considerada excelente dentro das características de nossa proposta.

Devemos considerar, entretanto, que estes valores estão relacionados ao processador ideal definido no item 5.3.1. A utilização de um processador real induz a uma perda que só pode ser determinada pelo mapeamento do modelo para o processador real escolhido.

### 5.3.4 Refinamento da arquitetura

O processo de simulação mostrou disparidades na carga computacional de cada estágio conforme mostrado na tabela 5.5 do item 5.3.3. A solução deste problema está relacionada com o binômio custo x velocidade.

A solução de maior velocidade é melhorar o desempenho dos estágios mais lentos através de seu particionamento ou da migração de algumas tarefas para um hardware dedicado. A avaliação de um elemento, por exemplo, é bastante apropriada para implementação " em hardware ".

A solução de menor custo consiste em manter o desempenho atual otimizando a máquina de forma a minimizar seu custo.

Consideramos que a segunda alternativa é a que mais se aplica tendo em vista o bom desempenho alcançado a partir da tecnologia empregada e da generalidade da solução. A principal providência que deve ser tomada é a eliminação do gargalo causado pelo estágio 4.

A forma de alcançar este objetivo é a redistribuição de tarefas entre os estágios, eliminando as ociosidades.

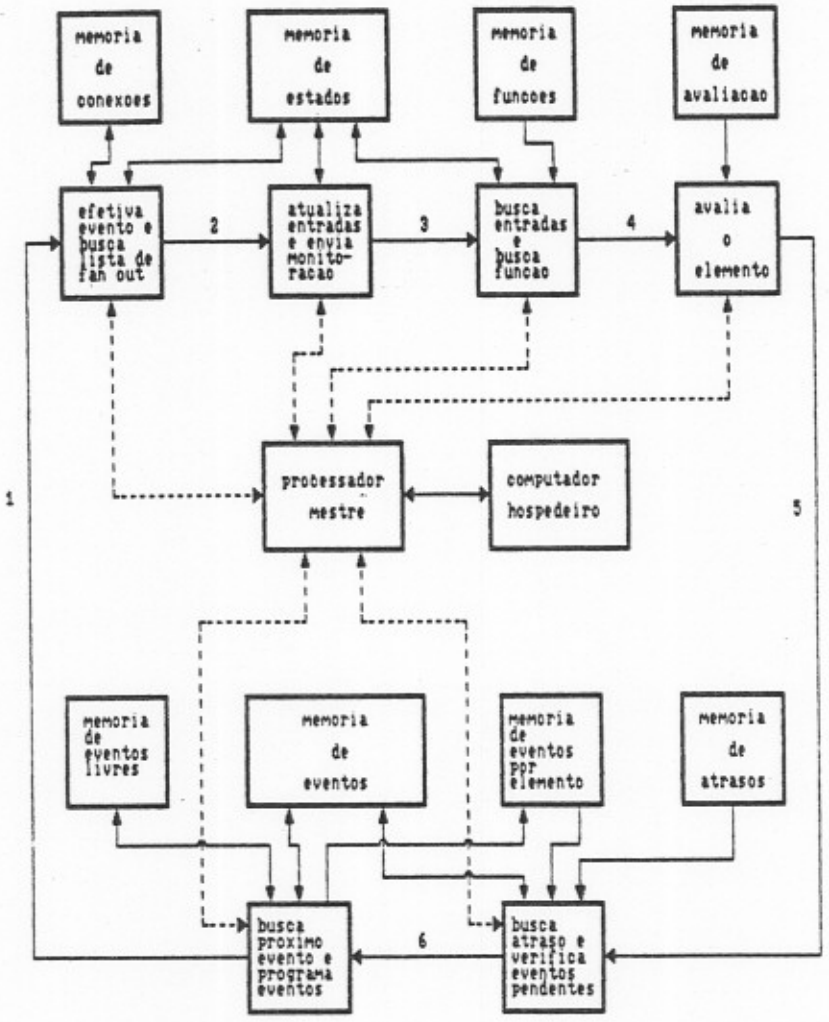
Inicialmente vemos que é possível aglutinar os estágios 1 e 7 sem perda de eficiência. Desta forma eliminamos um processador e parte da concorrência no acesso às memórias, simplificando a implementação. As tarefas dos estágios 2, 3 e 4 devem ser redistribuídas. Criar um acesso do estágio 2 à memória_de_estados estabele as condições necessárias a esta redistribuição. A arquitetura revista é apresentada na figura 5.20.

O estágio busca_próximo_evento_e_programa_eventos realiza as atividades de seus dois antecessores. A memória_de_eventos, a memória_de_eventos_livres e a memória_de_eventos_por_elemento mantém suas estruturas inalteradas.

A efetivação do evento passou para o estágio efetiva_evento_e_busca_lista_de_conexões_de_saída. Para tanto foi estabelecido o acesso deste estágio à memória_de_estados. O evento e a lista dos elementos afetados pelo mesmo é enviado ao estágio seguinte.

A atualização do valor das entradas passou para o estágio atualiza_entradas_e_envia_monitoração. Como o número de entradas está presente na memória_de_funções é necessário agregar à memória_de_estados a informação que permita a este estágio percorrer a lista de entradas. Com apenas 1 bit podemos indicar se o elemento possui mais de 8 entradas o que estabelece a forma de acesso às entradas do elemento.

O estágio busca_entradas_e_busca_função foi simplificado em relação a seu antecessor. Sua função atual é fornecer o elemento e seus atributos para o estágio seguinte.




BUFFERS ENTRE ESTAGIOS

1 EVENTO ( elemento , valor )
2 EVENTO , NRO FAN OUT , FAN OUT ( elemento , pos entrada )
3 ELEMENTO
4 ELEMENTO , FUNCAO , SAIDA , NRO ENT , ENT1 , ENT2 , .... , ENT8
5 ELEMENTO , SAIDA , NOVA SAIDA
6 EVENTO ( elemento , nova saida , tempo ) , END ULTIMO EV ELEMENTO

Figura 5.20 Evolução da AESL proposta

Os estágios avalia_o_elemento e busca_atraso_e_verifica_eventos_pendentes e suas respectivas memórias permanecem inalterados.

Uma forma de estimar o desempenho, após esta reformulação, é utilizar a relação dos tempos de processamento dos estágios 4 e 6 na tabela 5.5 do item 5.3.3 ( 10.089 / 7.499 => 1,35 ). Considerando que nenhum estágio ficará mais lento que o busca_atraso_e_verifica_eventos_pendentes com a redução do número de estágios, associada à redistribuição de tarefas, o desempenho calculado no item 5.3.3 foi, no mínimo, mantido e provavelmente incrementado de até 35%. Desta forma podemos esperar uma taxa de 72.198 ev/s ou 180.495 avaliações/s na implementação microprogramada.

## 5.4 COMPARAÇÃO COM OUTRAS ARQUITETURAS

A análise das características de nossa AESL pode ser complementada pela comparação da mesma ( em suas duas implementações possíveis ) com outras arquiteturas utilizadas no processo de simulação lógica.

Considerando as arquiteturas descritas no item 4.2 e utilizando a taxonomia definida no item 4.1 selecionamos as arquiteturas com características semelhantes à nossa, ou seja, exploração da concorrência por particionamento do algoritmo.

Em [BAR80] é utilizado um algoritmo de dois passos o que torna menos eficiente a simulação visto que o aproveitamento dos estágios é menor que num algoritmo de passo único. É uma arquitetura mais rígida considerando que:

- o número máximo de elementos está limitado a 32 K ;
- o processador de atualização é implementado diretamente em hardware.

A taxa máxima de avaliações é semelhante à nossa AESL se considerarmos a implementação com processadores microprogramáveis.

A arquitetura proposta em [ABR83] é a que mais se aproxima de nossa AESL. O número de estágios é o mesmo navendo diferenças na função dos estágios e na distribuição da estrutura de dados nas memórias. Nossa AESL não possui avaliadores funcionais especializados o que pode ser feito, em software, na unidade_de_avaliação. A existência de um processador mestre não é citada ficando indefinida a forma de comunicação com o usuário. A microprogramação do algoritmo também é utilizada. O desempenho de 1.000.000 avaliações/s foi obtido considerando um tempo médio de avaliação de 0,9 uS, correspondente à execução de nove microinstruções. Em nossa proposta de implementação microprogramada, utilizando um algoritmo sem grandes preocupações de eficiência, chegamos a 5,5 uS por avaliação ( 180.495 avaliações/s ), ou seja, uma média de 55 microinstruções por avaliação. A diferença no desempenho encontra-se, portanto, na alta eficiência do algoritmo considerado em [ABR83] que contrasta com as 30 a 60 microinstruções por avaliação citadas em [SMI86].

A implementação direta "em hardware" de um algoritmo de simulação é proposta em [GLA84]. A especificidade da solução é compensada por uma taxa de avaliações de até 5 milhões de avaliações/s. Um pipeline de 11 estágios, que acessam 10 memórias, executa as diversas partes do algoritmo. Um gargalo no pipeline é causado pelo acesso à lista de conexões de entrada do elemento que foi eliminada em nossa proposta. A avaliação de alguns elementos é feita, num segundo passo, pelo processador mestre, originando uma queda de desempenho cujas consequências não foram avaliadas.

Uma proposta de arquitetura mais simples que nossa AESL é apresentada em [PAS85]. Trata-se de um acelerador de

simulação conectado diretamente ao barramento de uma estação de trabalho. O número de estágios foi reduzido a três com suas respectivas memórias locais. A taxa de avaliações relatada é de 100.000 avaliações/s o que fica aquém de nossa AESL.

A implementação de nossa AESL com microprocessadores pode ser comparada à simulação executada numa estação de trabalho ou num microcomputador IBM PC.

Estações de trabalho rodam, geralmente, a velocidades próximas de 1000 ev/s [GOE87] o que torna nossa AESL aproximacamente 7 vezes mais rápida.

Com um IBM PC são obtidos, em média, de 200 a 500 ev/s [GOE86] elevando nosso ganho para um mínimo de 14. Um dos principais problemas do PC é a sua limitação de memória. Um PC AT, por exemplo, permite simular uma máximo de 3.000 a 5.000 portas lógicas [GOE86] o que é largamente superado por nossa AESL ( 128 K portas lógicas ).

# 6 CONSIDERAÇÕES FINAIS

O problema da ineficiência do processo de simulação lógica tem sido abordado de várias maneiras à medida que aumenta a utilização desta técnica. O uso de arquiteturas especiais para simulação lógica é uma característica da década de 80. A crescente utilização de circuitos integrados de aplicação específica ( ASIC ), bem como o surgimento de AESL de menor custo ou com possibilidade de compartilhamento tem facilitado o crescimento do mercado destas máquinas.

O desempenho de uma AESL deve ser analisado tendo por base a complexidade do algoritmo de simulação que ela executa. Um modelo de atraso zero ou unitário utiliza um algoritmo simples, o que justifica o surpreendente desempenho apresentado por algumas máquinas. Na verdade, o modelo utilizado para descrever um sistema depende, fundamentalmente, da tecnologia empregada na implementação do mesmo. Com o constante avanço da microeletrônica, os modelos, e conseqüentemente os algoritmos, estão mais sujeitos a alterações. Uma AESL eficiente, mas com estrutura rígida, exige que ferramentas de software façam o mapeamento entre os novos modelos e as primitivas disponíveis na máquina. A arquitetura ideal, do ponto de vista técnico e mercadológico, deve possuir uma alta flexibilidade e um baixo custo, aliados a um desempenho que justifique sua utilização.

Nossa proposta foi centrada nos aspectos de custo e flexibilidade. A facilidade de alterar o algoritmo aumenta sobremaneira o número de possíveis usuários. O baixo custo, particularmente na versão com microprocessadores, torna a máquina acessível aos mesmos. A utilização de microprocessadores de 16 bits permite ao usuário, principalmente no caso de universidades, efetuar alterações no algoritmo.

A simulação mostrou-se uma ferramenta extremamente útil na análise de nossa AESL. A proposta inicial, com maior número de estágios, foi uma tentativa de fracionar ao máximo o algoritmo de simulação lógica. Com a simulação da AESL foi possível otimizar o número de estágios e a distribuição de tarefas entre eles.

A implementação da arquitetura proposta deve ser precedida de:

- escolha do processador a ser utilizado;
- adaptação do modelo GPSS para o processador escolhido.

Após uma nova rodada de simulações será possível avaliar o desempenho da máquina e proceder, se necessário, a pequenos ajustes na estrutura da mesma. A arquitetura estará, então, validada e pronta para a implementação segundo a tecnologia escolhida ( microprocessadores ou processadores microprogramáveis).

Uma preocupação constante, durante todo o trabalho, foi a executabilidade da proposta. Nossa experiência como Engenheiro de Desenvolvimento no CPD da UFRGS mostrou que é muito grande a distância entre a definição de uma máquina e a sua entrada em operação. Há, em nosso país, uma grande carência de componentes eletrônicos e ferramentas para o desenvolvimento de software e hardware. Neste contexto a implementação com microprocessadores é a que se mostra mais adequada à realidade nacional.

# ANEXOS

# ANEXO 1

DESCRIÇÃO DO ALGORITMO (POR ESTÁGIO NA AESL)

```
PROCEDIMENTO BUSCA_PRóXIMO_EVENTO

   Inicio
1T SAI_FIM_EVENTOS <- FALSO
1T BLOCO <- MEM_EVENTOS ( TEMPO_ATUAL )
2T Enquanto BLOCO <> nil
   Faça Início
2T      END_BLOCO <- BLOCO << ( LOG2 NRO_PALAVRAS_DO_BLOCO )
1T      PALAVRA0 <- MEM_EVENTOS ( END_BLOCO )
2T      NRO_EVENTOS <- PALAVRA0.NRO_EVENTOS
2T      END_EV <- END_BLOCO + 1
2T      Enquanto  NRO_EVENTOS > 0
        Faça Início                                          % buscar evento
1T           PAL0_SAI <- MEM_EVENTOS ( END_EV )
2T           Se PAL0_SAI.VAL = FALSO
3T           Então END_EV <- END_EV + 3
             Senão Início
1T                 END_EV <- END_EV + 1
1T                 PAL1_SAI <- MEM_EVENTOS ( END_EV )
1T                 END_EV <- END_EV + 1
1T                 PALAVRA1 <- MEM_EVENTOS ( END_EV )
1T                 END_EV <- END_EV + 1
5T                 END_ELEM <- CONCAT(PAL1_SAI.ELEMEMTO_H,PAL0_SAI.ELEMENTO_L)
1T                 END_ELEM <- 2 * END_ELEM
2T                 Se PAL0_SAI.ENCAD = VERDADEIRO            % atualiza memória_de_
                   Então Início                              % eventos_por_elemento
1T                       PALAVRA2.VAL <- VERDADEIRO
3T                       PALAVRA2.END_PRIM_EV_L <- PAL1_SAI.END_PROX_EV_ELEM_L
1T                       MEM_EV_EL ( END_ELEM ) <- PALAVRA2
1T                       END_ELEM <- END_ELEM + 1
1T                       MEM_EV_EL ( END_ELEM ) <- PALAVRA1
1T                       Fim
                   Senão Início
1T                       PALAVRA2.VAL <- FALSO
1T                       MEM_EV_EL ( END_ELEM ) <- PALAVRA2
                         Fim
1T                 Repita Ler ESTADO_BUF_SAI
2T                 Até que ESTADO_BUF_SAI <> CHEIO
1T                 BUF_SAI.PAL0 <- PAL0_SAI                  % envia evento
1T                 BUF_SAI.PAL1 <- PAL1_SAI
                   Fim
1T           NRO_EVENTOS <- NRO_EVENTOS - 1
1T           Fim
3T      ULTIMO_END_BLOCO <- END_BLOCO | MASC_P_FIM_BLOCO     % devolve bloco para a
1T      PROX_BLOCO <- MEM_EVENTOS ( ULTIMO_END_BLOCO )       % memória_de_eventos_livres
1T      MEM_EV_LIVRES ( FIM_LISTA ) <- BLOCO
2T      Se FIM_LISTA = ULTIMO_ENDER
2T      Então FIM_LISTA <- PRIM_ENDER
1T      Senão FIM_LISTA <- FIM_LISTA + 1
1T      BLOCO <- PROX_BLOCO
1T      Fim
1T SAI_FIM_EVENTOS <- VERDADEIRO
   Fim
```

```
PROCEDIMENTO  BUSCA_LISTA_DE_CONEXÕES_DE_SAÍDA

   Início
1T Repita Repita Repita Ler ESTADO_BUF_ENT
1T                      Ler ENT_FIM_EVENTOS
2T                Até que ESTADO_BUF_ENT () VAZIO ou
2T                        ENT_FIM_EVENTOS = VERDADEIRO
2T                Se ESTADO_BUF_ENT () VAZIO
                  Então Início
1T                      SAI_FIM_EVENTOS (- FALSO
1T                      PAL0_ENT (- BUF_ENT.PAL0
1T                      PAL1_ENT (- BUF_ENT.PAL1
5T                      END_ELEM (- CONCAT(PAL1_ENT.ELEMENTO_H,PAL0_ENT.ELEMENTO_L)
1T                      END_ELEM (- END_ELEM * 2
1T                      PAL2_SAI (- MEM_CONEXÕES ( END_ELEM )
1T                      Repita Ler ESTADO_BUF_SAI0
2T                      Até que ESTADO_BUF_SAI0 () CHEIO
1T                      BUF_SAI0.PAL0 (- PAL0_ENT               % envia evento e
1T                      BUF_SAI0.PAL1 (- PAL1_ENT               % nro de fan-out (tamanho
1T                      BUF_SAI0.PAL2 (- PAL2_SAI               % da lista )
2T                      NRO_FAN_OUT (- PAL2_SAI.NRO_FAN_OUT
1T                      END_ELEM (- END_ELEM + 1
1T                      PALAVRA0 (- MEM_CONEXÕES ( END_ELEM )
5T                      END_LISTA_F (- CONCAT(PALAVRA0,PAL2_SAI.END_LISTA_L)
1T                      CONT (- 0
2T                      Enquanto CONT ( NRO_FAN_OUT
                        Faça Início
1T                           PAL3_SAI (- MEM_CONEXÕES ( END_LISTA_F )
1T                           END_LISTA_F (- END_LISTA_F + 1
1T                           PAL4_SAI (- MEM_CONEXÕES ( END_LISTA_F )
1T                           END_LISTA_F (- END_LISTA_F + 1
1T                           Repita Ler ESTADO_BUF_SAI1
2T                           Até que ESTADO_BUF_SAI1 () CHEIO
1T                           BUF_SAI1.PAL3 (- PAL3_SAI          % envia elementos da lista
1T                           BUF_SAI1.PAL4 (- PAL4_SAI
1T                           CONT (- CONT + 1
1T                           Fim
                        Fim
1T          Até que ENT_FIM_EVENTOS = VERDADEIRO
1T          SAI_FIM_EVENTOS (- VERDADEIRO
    Até que haja uma interrupção do MESTRE
    Fim
```

```
PROCEDIMENTO  EFETIVA_EVENTO_E_VERIFICA_MONITORAÇÃO

   Início
1T Repita Repita Repita Ler ESTADO_BUF_ENT0
1T                      Ler ENT_FIM_EVENTOS
2T                Até que ESTADO_BUF_ENT0 () VAZIO ou
2T                        ENT_FIM_EVENTOS = VERDADEIRO
2T                Se ESTADO_BUF_ENT0 () VAZIO
                  Então Início
1T                      SAI_FIM_EVENTOS (- FALSO
1T                      PAL0_ENT (- BUF_ENT0.PAL0
1T                      PAL1_ENT (- BUF_ENT0.PAL1
1T                      PAL2_ENT (- BUF_ENT0.PAL2
2T                      NRO_FAN_OUT (- PAL2_ENT.NRO_FAN_OUT
5T                      END_ELEM (- CONCAT(PAL1_ENT.ELEMENTO_H,PAL0_ENT.ELEMENTO_L)
1T                      END_ELEM (- 2 * END_ELEM
1T                      PALAVRA0 (- MEM_ESTADOS ( END_ELEM )
3T                      PALAVRA0.SAIDA (- PAL0_ENT.VALOR            % efetiva evento
1T                      MEM_ESTADOS ( END_ELEM ) (- PALAVRA0
2T                      Se PALAVRA0.MONITORAÇÃO = VERDADEIRO        % verifica monitoração
                        Então Início
1T                            Repita Ler ESTADO_BUF_RESULT
2T                            Até que ESTADO_BUF_RESULT () CHEIO
1T                            BUF_RESULT.PAL0 (- PAL0_ENT           % envia monitoração
1T                            BUF_RESULT.PAL1 (- PAL1_ENT           % para o mestre
                              Fim
1T                      CONT (- 0
2T                      Enquanto CONT ( NRO_FAN_OUT                 % recebe e envia
                        Faça Início                                 % lista de fan-out
1T                           Repita Ler ESTADO_BUF_ENT1
2T                           Até que ESTADO_BUF_ENT1 () VAZIO
1T                           PAL3_ENT (- BUF_ENT1.PAL3
1T                           PAL4_ENT (- BUF_ENT1.PAL4
1T                           Repita Ler ESTADO_BUF_SAI
2T                           Até que ESTADO_BUF_SAI () CHEIO
1T                           BUF_SAI.PAL0 (- PAL0_ENT
1T                           BUF_SAI.PAL1 (- PAL3_ENT
1T                           BUF_SAI.PAL2 (- PAL4_ENT
1T                           CONT (- CONT + 1
1T                           Fim
                        Fim
1T        Até que ENT_FIM_EVENTOS = VERDADEIRO
1T        SAI_FIM_EVENTOS (- VERDADEIRO
   Até que haja uma interrupção do MESTRE
   Fim
```

```
PROCEDIMENTO  BUSCA_E_ATUALIZA_ENTRADAS/BUSCA_FUNÇÃO

1T Repita Repita Repita Ler ESTADO_BUF_ENT
1T                       Ler ENT_FIM_EVENTOS
2T                 Até que ESTADO_BUF_ENT () VAZIO   ou
2T                         ENT_FIM_EVENTOS = VERDADEIRO
2T                 Se  ESTADO_BUF_ENT () VAZIO
                   Então  Início
1T                        SAI_FIM_EVENTOS (- FALSO
1T                        PAL0_ENT (- BUF_ENT.PAL0
1T                        PAL1_ENT (- BUF_ENT.PAL1
1T                        PAL2_ENT (- BUF_ENT.PAL2
2T                        POS_ENT (- PAL1_ENT.POS_ENT
5T                        END_ELEM (- CONCAT1(PAL2_ENT.ELEM_F_H,PAL1_ENT.ELEM_F_L)
2T                        END_FUNC (- END_ELEM + INICIO_TAB_FUNÇÕES
1T                        FUNÇÃO (- MEM_FUNÇÕES ( END_FUNC )
1T                        NRO_ENT (- MEM_FUNÇÕES ( FUNÇÃO )
5T                        PAL2_SAI (- CONCAT(FUNÇÃO,NRO_ENT)
1T                        END_ELEM (- 2 * END_ELEM
1T                        PALAVRA1 (- MEM_ESTADOS ( END_ELEM )
5T                        PAL0_SAI (- CONCAT(PALAVRA1.SAÍDA,PAL1_ENT.ELEM_F_L)
1T                        Repita  Ler ESTADO_BUF_SAI0
2T                        Até que ESTADO_BUF_SAI0 () CHEIO
1T                        BUF_SAI0.PAL0 (- PAL0_SAI
1T                        BUF_SAI0.PAL1 (- PAL2_ENT
1T                        BUF_SAI0.PAL2 (- PAL2_SAI
2T                        Se NRO_ENT ) 8
                          Então Início
1T                              END_ELEM (- END_ELEM + 1
1T                              PALAVRA2 (- MEM_ESTADOS ( END_ELEM )
5T                              END_ENT (- CONCAT(PALAVRA2,PALAVRA1.END_ENT_L)
1T                              Fim
2T                        Senão END_ENT (- END_ELEM + 1
1T                        CONT_ENT (- 0
1T                        ATUALIZAÇÃO (- FALSO
2T                        Enquanto CONT_ENT ( NRO_ENT
                          Faça Início
1T                             PAL3_SAI (- MEM_ESTADOS(END_ENT)
2T                             CONT_ENT (- CONT_ENT + 8
2T                             Se ATUALIZAÇÃO = FALSO e
2T                                POS_ENT (= CONT_ENT
                               Então Início
5T                                   PAL3_SAI.(POS_ENT) (- PAL0_ENT.VALOR
1T                                   ATUALIZAÇÃO (- VERDADEIRO
1T                                   MEM_ESTADOS(END_ENT) (- PAL3_SAI
                                     Fim
1T                             Repita Ler ESTADO_BUF_SAI1
2T                             Até que ESTADO_BUF_SAI1 () CHEIO
1T                             BUF_SAI1.PAL3 (- PAL3_SAI
1T                             END_ENT (- END_ENT + 1
1T                             Fim
                          Fim
2T              Até que ENT_FIM_EVENTOS = VERDADEIRO
1T              SAI_FIM_EVENTOS (- VERDADEIRO
   Até que haja uma interrupção do mestre
```

```
PROCEDIMENTO  AVALIA_O_ELEMENTO

   Início
1T Repita Repita Repita Ler ESTADO_BUF_ENT0
1T                      Ler ENT_FIM_EVENTOS
2T                Até que ESTADO_BUF_ENT0 () VAZIO ou
2T                        ENT_FIM_EVENTOS = VERDADEIRO
2T                Se ESTADO_BUF_ENT0 () VAZIO
                  Então Início
1T                      SAI_FIM_EVENTOS (- FALSO
1T                      PAL0_ENT (- BUF_ENT0.PAL0
1T                      PAL1_ENT (- BUF_ENT0.PAL1
1T                      PAL2_ENT (- BUF_ENT0.PAL2
2T                      FUNÇÃO (- PAL2_ENT.FUNÇÃO
2T                      SAÍDA (- PAL0_ENT.SAÍDA
2T                      NRO_ENT (- PAL2_ENT.NRO_ENT
1T                      Repita Ler ESTADO_BUF_ENT1
2T                      Até que ESTADO_BUF_ENT1 () VAZIO
1T                      ENTRADAS (- BUF_ENT1.PAL3
1T                      END_TAB (- FUNÇÃO
3T                      END_TAB (- CONCAT(END_TAB,ENTRADAS.ENT1-ENT6)
1T                      NOVA_SAI (- MEM_AVAL ( END_TAB )
2T                      NRO_ENT (- NRO_ENT - 6
1T                      PONT_ENT (- 6
2T                      Enquanto  NRO_ENT ) 0
                        Faça Início
1T                           FUNÇÃO (- FUNÇÃO + 1                % seleciona nova tabela
1T                           END_TAB (- FUNÇÃO
3T                           END_TAB (- CONCAT(END_TAB,NOVA_SAI) % concatena saida da
                             BUSCA_ENTRADA(5,ENT,FIM_ENT)        % tabela anterior e
3T                           END_TAB (- CONCAT(END_TAB,ENT)      % busca as entradas
2T                           Se FIM_ENT = FALSO                  % restantes
                             Então Início
                                  BUSCA_ENTRADA(4,ENT,FIM_ENT)
3T                                END_TAB (- CONCAT(END_TAB,ENT)
2T                                Se FIM_ENT = FALSO
                                  Então Início
                                        BUSCA_ENTRADA(3,ENT,FIM_ENT)
3T                                      END_TAB (- CONCAT(END_TAB,ENT)
2T                                      Se FIM_ENT = FALSO
                                        Então Início
                                              BUSCA_ENTRADA(2,ENT,FIM_ENT)
3T                                            END_TAB (- CONCAT(END_TAB,ENT)
2T                                            Se FIM_ENT = FALSO
                                              Então Início
                                                    BUSCA_ENTRADA(1,ENT,FIM_ENT)
3T                                                  END_TAB (- CONCAT(END_TAB,ENT)
                                                    Fim
                                              Fim
                                        Fim
                                  Fim
                             NOVA_SAI (- MEM_AVAL ( END_TAB )
1T                           Fim
```

```
1T                    PAL0_SAI (- PAL0_ENT
2T                    PAL0_SAI.NOVA_SAIDA (- NOVA_SAI
1T                    Repita Ler ESTADO_BUF_SAI
2T                    Até que ESTADO_BUF_SAI () CHEIO
1T                    BUF_SAI.PAL0 (- PAL0_SAI
1T                    BUF_SAI.PAL1 (- PAL1_ENT
                      Fim
2T        Até que ENT_FIM_EVENTOS = VERDADEIRO
1T        SAI_FIM_EVENTOS (- VERDADEIRO
   Até que haja uma interrupção do mestre
   Fim


PROCEDIMENTO  BUSCA_ENTRADA(POS,VALOR,ÚLTIMA)    % subrotina de AVALIA_O_ELEMENTO
                                                 % busca uma entrada e posiciona-a
   Início                                        % no campo apontado por POS
2T PONT_ENT (- PONT_ENT + 1
2T Se PONT_ENT ) 8
   Então Início
1T       Repita Ler ESTADO_BUF_ENT1
2T       Até que ESTADO_BUF_ENT1 () VAZIO
1T       ENTRADAS (- BUF_ENT1.PAL3
1T       PONT_ENT (- 1
         Fim                                                  % desloca entrada
6T VALOR (- ENTRADAS.(PONT_ENT) )) ( 16 - 2 * PONT_ENT )      % para bit - significativo
3T VALOR (- VALOR (( ( 12 - 2 * POS )                         % desloca entrada para
1T NRO_ENT (- NRO_ENT - 1                                     % posicao indicada por POS
2T Se NRO_ENT = 0
2T Então ÚLTIMA (- VERDADEIRO
1T Senão ULTIMA (- FALSO
   Fim
```

```
PROCEDIMENTO  BUSCA_ATRASO_E_VERIFICA_EVENTOS_PENDENTES

   Início
1T Repita Repita Repita Ler ESTADO_BUF_ENT
1T                      Ler ENT_FIM_EVENTOS
2T                Até que ESTADO_BUF_ENT () VAZIO ou
2T                        ENT_FIM_EVENTOS = VERDADEIRO
2T                Se ESTADO_BUF_ENT () VAZIO
                  Então Início
1T                      SAI_FIM_EVENTOS (- FALSO
1T                      PAL0_ENT (- BUF_ENT.PAL0
1T                      PAL1_ENT (- BUF_ENT.PAL1
2T                      SAÍDA (- PAL0_ENT.SAÍDA
2T                      NOVA_SAI (- PAL0_ENT.NOVA_SAI
5T                      END_ELEM (- CONCAT(PAL1_ENT.ELEM_F_H,PAL0_ENT.ELEM_F_L)
1T                      PALAVRA0 (- MEM_ATRASOS ( END_ELEM )
2T                      Se NOVA_SAI = 0
3T                      Então ATRASO (- PALAVRA0.ATRASO1    % transição 1 -) 0
2T                      Senão ATRASO (- PALAVRA0.ATRASO2    % transição 0 -) 1
2T                      TEMPO_EVP (- TEMPO_ATUAL + ATRASO
1T                      END_PAL (- 2 * END_ELEM
1T                      PALAVRA (- MEM_EV_EL ( END_PAL )
1T                      END_EV1 (- nil
2T                      Se PALAVRA.VAL = VERDADEIRO         % verifica se há eventos
                        Então Início                        % pendentes.  EV2 (- 1º evento
1T                            END_PAL (- END_PAL + 1
1T                            END_EV2_H (- MEM_EV_EL ( END_PAL )
5T                            END_EV2 (- CONCAT(END_EV2_H,PALAVRA.END_PRIM_EV_L)
1T                            EV2 (- MEM_EVENTOS ( END_EV2 )
2T                            END_BLOCO_EV2 (- END_EV2 & MÁSCARA_P_INÍCIO_BLOCO
1T                            PALAVRA1 (- MEM_EVENTOS ( END_BLOCO_EV2 )
2T                            TEMPO_EV2 (- PALAVRA1.TEMPO
2T -                          Enquanto [ (TEMPO_EVP ) TEMPO_ATUAL) e         % tempo do ev.
2T |                                     (TEMPO_EV2 ) TEMPO_ATUAL) e         % a programar
2T |                                     (TEMPO_EVP ) TEMPO TEMPO_EV2) ] ou  % ) tempo EV2
2T | min 8T  -                         [ (TEMPO_EVP ( TEMPO_ATUAL) e         % e EV2 () nil
2T |         -) 13T (adotado)            (TEMPO_EV2 ( TEMPO_ATUAL) e
2T | max 18T -                           (TEMPO_EVP ) TEMPO TEMPO_EV2) ] ou
2T |                                   [ (TEMPO_EVP ( TEMPO_ATUAL) e
2T |                                     (TEMPO_EV2 ) TEMPO_ATUAL) ]  e
2T -                                  [ END_EV2 () nil ]
```

```
                          Faça Início                        % procura evento com tempo
1T                             END_EV1 (- END_EV2            % imediatamente inferior
1T                             EV1 (- EV2                    % a TEMPO_EVP e guarda em EV1
2T                             Se EV2.ENCAD = FALSO
2T                             Então END_EV2 (- nil
                               Senão Início
1T                                   END_EV2 (- END_EV2 + 1
1T                                   PALAVRA2 (- MEM_EVENTOS ( END_EV2 )
1T                                   END_EV2 (- END_EV2 + 1
1T                                   PALAVRA3 (- MEM_EVENTOS ( END_EV2 )
5T                                   END_EV2 (- CONCAT(PALAVRA3,PALAVRA2.END_PROX_EV_L)
1T                                   EV2 (- MEM_EVENTOS ( END_EV2 )
2T                                   END_BLOCO_EV2 (- END_EV2 & MÁSCARA_P_INÍCIO_BLOCO
1T                                   PALAVRA1 (- MEM_EVENTOS ( END_BLOCO_EV2 )
2T                                   TEMPO_EV2 (- PALAVRA1.TEMPO
                                     Fim
1T                             Fim
2T -                           Se  [ (TEMPO_EVP ) TEMPO_ATUAL) e          % tempo do evento
2T |                                 (TEMPO_EV2 ) TEMPO_ATUAL) e          % a programar (=
2T |                                 (TEMPO_EVP (= TEMPO TEMPO_EV2) ] ou  % TEMPO_EVP
2T | min 8T  -                      [ (TEMPO_EVP ( TEMPO_ATUAL) e
2T |         -) 13T (adotado)          (TEMPO_EV2 ( TEMPO_ATUAL) e
2T | max 18T -                        (TEMPO_EVP (= TEMPO TEMPO_EV2) ] ou
2T |                                 [ (TEMPO_EVP ) TEMPO_ATUAL) e
2T |                                   (TEMPO_EV2 ( TEMPO_ATUAL) ]  e
2T -                                  [ END_EV2 () nil ]
1T                             Então Repita EV2.VAL (- FALSO
1T                                          MEM_EVENTOS ( END_EV2 ) (- EV2   % cancela EV2 e
2T                                          Se EV2.ENCAD = FALSO             % subsequentes
2T                                          Então END_EV2 (- nil
                                            Senão Início
1T                                                END_EV2 (- END_EV2 + 1
1T                                                PALAVRA2 (- MEM_EVENTOS ( END_EV2 )
1T                                                END_EV2 (- END_EV2 + 1
1T                                                PALAVRA3 (- MEM_EVENTOS ( END_EV2 )
5T                                                END_EV2 (- CONCAT(PALAVRA3,PALAVRA2.END_PROX_EV_L)
1T                                                EV2 (- MEM_EVENTOS ( END_EV2 )
2T                                       Até que END_EV2 = nil
                                Fim
```

```
5T                    Se (END_EV1 = nil e SAÍDA <> NOVA_SAI) ou (EV1.VALOR <> NOVA_SAI)
                      Então Início
2T                          PAL1_EV.VAL <- VERDADEIRO                 % programa evento
2T                          PAL1_EV.ENCAD <- FALSO                    % se: variou saída
2T                          PAL1_EV.VALOR <- NOVA_SAI                 % e não há eventos
3T                          PAL1_EV.ELEMENTO_L <- PAL0_ENT.ELEM_F_L   % programados ou
3T                          PAL2_EV.ELEMENTO_H <- PAL1_ENT.ELEM_F_H   % o último evento
3T                          PAL2_EV.END_ÚLTIMO_EV_L <- END_EV1.END_L  % programado tem
3T                          PAL3_EV.END_ÚLTIMO_EV_H <- END_EV1.END_H  % valor de saída
1T                          Repita Ler ESTADO_BUF_SAI                 % <> da nova saída
2T                          Até que ESTADO_BUF_SAI <> CHEIO
1T                          BUF_SAI.PAL0 <- TEMPO_EVP
1T                          BUF_SAI.PAL1 <- PAL1_EV
1T                          BUF_SAI.PAL2 <- PAL2_EV
1T                          BUF_SAI.PAL3 <- PAL3_EV
2T          Até que ENT_FIM_EVENTOS = VERDADEIRO
1T          SAI_FIM_EVENTOS <- VERDADEIRO
    Até que haja uma interrupção do mestre
    Fim
```

```
PROCEDIMENTO  PROGRAMA_EVENTOS

   Início
1T Repita Repita Repita Ler ESTADO_BUF_ENT
1T                      Ler ENT_FIM_EVENTOS
2T                Até que ESTADO_BUF_ENT () VAZIO ou
2T                        ENT_FIM_EVENTOS = VERDADEIRO
2T                Se ESTADO_BUF_ENT () VAZIO
                  Então Início
1T                      SAI_FIM_EVENTOS (- FALSO
1T                      TEMPO_EVP (- BUF_ENT.PAL0
1T                      EV0 (- BUF_ENT.PAL1
1T                      EV1 (- BUF_ENT.PAL2
1T                      EV2 (- BUF_ENT.PAL3
                        ALOCA_ESPAÇO ( )
2T                      Se END_ULT_EV () nil                          % verifica se há
                        Então Início                                  % evento anterior
1T                            PALAVRA1 (- MEM_EVENTOS ( END_ULT_EV )  % programado
2T                            PALAVRA1.ENCAD (- VERDADEIRO
1T                            MEM_EVENTOS ( END_ULT_EV ) (- PALAVRA1  % atualiza ponteiro
1T                            END_ULT_EV (- END_ULT_EV + 1            % p/ próximo ev.
1T                            PALAVRA2 (- MEM_EVENTOS ( END_ULT_EV )  % no ev. anterior
2T                            PALAVRA2.END_PROX_EV_L (- END_EV_LIVRE_L
1T                            MEM_EVENTOS ( END_ULT_EV ) (- PALAVRA2
1T                            END_ULT_EV (- END_ULT_EV + 1
2T                            MEM_EVENTOS ( END_ULT_EV ) (- END_EV_LIVRE_H
1T                            Fim
                        Senão Início
5T                            END_PAL3 (- CONCAT(EV1.ELEMENTO_H,EV0.ELEMENTO_L)  % atualiza
1T                            END_PAL3 (- END_PAL3 * 2                  % memória_de_
2T                            PALAVRA3.VAL (- VERDADEIRO                % eventos_por
2T                            PALAVRA3.END_PRIM_EV_L (- END_EV_LIVRE_L  % _elemento
1T                            MEM_EV_EL ( END_PAL3 ) (- PALAVRA3
1T                            END_PAL3 (- END_PAL3 + 1                  % é o único
1T                            MEM_EV_EL ( END_PAL3 ) (- END_EV_LIVRE_H  % ev. do elem
                              Fim
                        SALVA_EVENTO ( )
2T              Até que ENT_FIM_EVENTOS = VERDADEIRO
1T              SAI_FIM_EVENTOS (- VERDADEIRO
```

```
1T Repita Repita Ler ESTADO_BUF_MSG
1T                Ler ENT_FIM_MSG
2T         Até que ESTADO_BUF_MSG <> VAZIO ou
2T                 ENT_FIM_MSG = VERDADEIRO
2T         Se ESTADO_BUF_MSG <> VAZIO
           Então Início
1T               SAI_FIM_PROG <- FALSO
1T               TEMPO_EVP <- BUF_MSG.PAL0
1T               EV0 <- BUF_MSG.PAL1
1T               EV1 <- BUF_MSG.PAL2
1T               EV2 <- BUF_MSG.PAL3
                 ALOCA_ESPAÇO ( )
                 SALVA_EVENTO ( )
                 Fim
2T Até que ENT_FIM_MSG = VERDADEIRO
1T SAI_FIM_PROG <- VERDADEIRO
   Até que haja uma interrupção do mestre
   Fim
```

```
PROCEDIMENTO  ALOCA_ESPAÇO ( )  % alocação de um evento livre num bloco do tempo
                                % associado ao evento a ser programado - caso não haja
                                % evento livre um novo bloco é alocado e encadeado no
                                % início da lista de blocos para o tempo do evento
   Início
5T END_ULT_EV <- CONCAT(EV2,EV1.END_ULT_EV_L)
2T END_LIVRE <- 2 * TEMPO_EVP
1T END_EV_LIVRE_L <- MEM_EV_LIVRES ( END_LIVRE )
1T END_LIVRE <- END_LIVRE + 1
1T END_EV_LIVRE_H <- MEM_EV_LIVRES ( END_LIVRE )
5T END_EV_LIVRE <- CONCAT(END_EV_LIVRE_H,END_EV_LIVRE_L)
2T Se END_EV_LIVRE = nil
   Então Início
1T       NOVO_BLOCO <- MEM_EV_LIVRES ( INÍCIO_LISTA )
2T       Se INÍCIO_LISTA = ÚLTIMO_ENDER
2T       Então INÍCIO_LISTA <- PRIM_ENDER
1T       Senão INÍCIO_LISTA <- INÍCIO_LISTA + 1
2T       END_NOVO_BLOCO <- NOVO_BLOCO (( ( LOG2 NRO_PALAVRAS_DO_BLOCO )
2T       PALAVRA0.TEMPO <- TEMPO_EVP
2T       PALAVRA0.NRO_EVENTOS <- 0
1T       MEM_EVENTOS ( END_NOVO_BLOCO ) <- PALAVRA0
2T       ÚLTIMO_END_NOVO_BLOCO <- END_NOVO_BLOCO | MASC_P_FIM_BLOCO
1T       BLOCO <- MEM_EVENTOS ( TEMPO_EVP )
1T       MEM_EVENTOS ( ÚLTIMO_END_NOVO_BLOCO ) <- BLOCO
1T       MEM_EVENTOS ( TEMPO_EVP ) <- NOVO_BLOCO
1T       END_EV_LIVRE <- END_NOVO_BLOCO
1T       END_EV_LIVRE <- END_EV_LIVRE + 1
         Fim
   Fim
```

```
PROCEDIMENTO  SALVA_EVENTO ( )    % insere o evento na memória_de_eventos
                                  % atualiza nro de eventos do bloco
   Início                         % atualiza endereço do proximo evento livre
1T MEM_EVENTOS ( END_EV_LIVRE ) (- EV0
1T END_EV_LIVRE (- END_EV_LIVRE + 1
1T MEM_EVENTOS ( END_EV_LIVRE ) (- EV1
2T END_BLOCO (- END_EV_LIVRE & MASC_P_INÍCIO_BLOCO
1T PALAVRA4 (- MEM_EVENTOS ( END_BLOCO )
3T PALAVRA4.NRO_EVENTOS (- PALAVRA4.NRO_EVENTOS + 1
1T MEM_EVENTOS ( END_BLOCO ) (- PALAVRA4
2T Se PALAVRA4.NRO_EVENTOS = NRO_EVENTOS_POR_BLOCO
   Então Início
1T       MEM_EV_LIVRES ( END_LIVRE ) (- nil
1T       END_LIVRE (- END_LIVRE - 1
1T       MEM_EV_LIVRES ( END_LIVRE ) (- nil
1T       Fim
   Senão Início
2T       END_EV_LIVRE (- END_EV_LIVRE + 3
1T       MEM_EV_LIVRES ( END_LIVRE ) (- END_EV_LIVRE.END_H
1T       END_LIVRE (- END_LIVRE - 1
1T       MEM_EV_LIVRES ( END_LIVRE ) (- END_EV_LIVRE.END_L
         Fim
   Fim
```

```
PROCEDIMENTO  MESTRE

    Início
1T  TEMPO_ATUAL <- 1
20T Recebe do hospedeiro variações de entrada
2T  Enquanto houver variações a efetivar
    Faça Início
1T       SAI_FIM_EV <- FALSO
1T       Repita Ler ESTADO_BUF_SAI0
2T       Até que ESTADO_BUF_SAI0 <> CHEIO
1T       BUF_SAI0.PAL0 <- TEMPO_EVP              % envia variação e entrada
1T       BUF_SAI0.PAL1 <- EV0                    % como evento a programar
1T       BUF_SAI0.PAL2 <- EV1
1T       BUF_SAI0.PAL3 <- EV2
1T       Fim
1T  SAI_FIM_EV <- VERDADEIRO
1T  Repita Ler ESTADO_BUF_SAI0
1T         Ler ENT_FIM_PROG
2T  Até que ESTADO_BUF_SAI0 <> CHEIO  e
2T          ENT_FIM_PROG = VERDADEIRO
1T  Dispara a simulação para o TEMPO_ATUAL      % ativa procedimento busca_proximo_evento
1T  Repita Ler ESTADO_BUF_ENT1
2T         Se ESTADO_BUF_ENT1 <> VAZIO           % recebe sinais monitorados
           Então Início
2T               Lê BUF_ENT1
5T               Envia para o hospedeiro
                 Fim
1T         Ler ENT_FIM_EV1                       % verifica atividade nos estágios
1T         Ler ENT_FIM_EV2
1T         Ler ENT_FIM_EV3
1T         Ler ENT_FIM_EV4
1T         Ler ENT_FIM_EV5
1T         Ler ENT_FIM_EV6
1T         Ler ENT_FIM_EV7
2T  Até que ENT_FIM_EV1 = VERDADEIRO e           aguarda até o fim de todos os estágios
2T          ENT_FIM_EV2 = VERDADEIRO e
2T          ENT_FIM_EV3 = VERDADEIRO e
2T          ENT_FIM_EV4 = VERDADEIRO e
2T          ENT_FIM_EV5 = VERDADEIRO e
2T          ENT_FIM_EV6 = VERDADEIRO e
2T          ENT_FIM_EV7 = VERDADEIRO
1T  TEMPO_ATUAL <- TEMPO_ATUAL + 1
2T  Até que TEMPO_ATUAL > TEMPO_FINAL_DE_SIMULAÇÃO
    Fim
```

# ANEXO 2

## MODELO DA AESL EM GPSS

```
        NOLIST
*
* definicao de constantes e parametros da simulacao
FALSE    EQU   0
TRUE     EQU   1
T_MAX    EQU   50                          ; tempo maximo de simulacao
TAM_BLO  EQU   32                          ; nro de eventos por bloco
EV_VAL   EQU   950                         ; nro de ev. validos em cada 1000
EV_ENCAD EQU   50                          ; nro de ev. encadeados em 1000
MONIT    EQU   20                          ; nro de saidas a monitorar em 1000
FAN_MIN  EQU   2                           ; variacao do nro de fan out
FAN_MAX  EQU   3
ENT_MIN  EQU   2                           ; variacao do nro de entradas
ENT_MAX  EQU   4
EV_PEND  EQU   150                         ; nro de elementos com eventos
*                                          ; programados a cada 1000 elementos
*
*
* definicao dos parametros das transacoes
NRO_EV   EQU    1                           ; nro de eventos a processar
EV_BLO   EQU    2                           ; nro de ev. no bloco em proces.
NRO_FAN  EQU    3                           ; fan out por elemento
NRO_ENT  EQU    4                           ; nro de entradas por elemento
POS_ENT  EQU    5                           ; posicao da entrada no elemento
CONT_ENT EQU    6                           ; contador auxiliar do nro de ent.
PONT_ENT EQU    7                           ; indice da entrada no cj de 4 ent
NRO_EV_M EQU    8                           ; nro de ev. do mestre a programar
ENCAD    EQU    9                           ; auxiliar no calculo do encadeam.
*
*
*
* definicao de savevalues
CONT_TMP EQU    1                           ; tempo corrente de simulacao
NUM_FAN  EQU    2                           ; nro de fan out util. no estagio 3
NUM_ENT  EQU    3                           ; nro de entradas util. no estagio 5
LOGON    EQU    4                           ; matriz de tempos de proc de ev
TP_EST1  EQU    5                           ; tempo total proc. ev. no estagio 1
TP_EST2  EQU    6                           ; idem estagio 2
TP_EST3  EQU    7                           ; idem estagio 3
TP_EST4  EQU    8                           ; idem estagio 4
TP_EST5  EQU    9                           ; idem estagio 5
TP_EST6  EQU   10                           ; idem estagio 6
TP_EST7  EQU   11                           ; idem estagio 7
*
*
*
* definicao de variaveis
AUX0     EQU    1
AUX1     EQU    2
AUX2     EQU    3
AUX3     EQU    4
AUX4     EQU    5
AUX5     EQU    6
AUX6     EQU    7
EVENTOS  EQU    8
```

```
* definicao de funcoes
EV_MEST  EQU    1
EXPON    EQU    2
ATIV     EQU    3
MAIS_ENT EQU    4
*
*
*
* definicao de variaveis logicas
FIM_EV1  EQU    1                       ; setada apos enviar o ult. ev.
FIM_EV2  EQU    2
FIM_EV3  EQU    3
FIM_EV4  EQU    4
FIM_EV5  EQU    5
FIM_EV6  EQU    6
FIM_EV7  EQU    7
FIM_EV_M EQU    8
BUSC_FAN EQU    9                       ; controla passagem do nro fan out
*                                       ; entre os estagios 2 e 3
BUSC_ENT EQU   10                       ; idem nro de ent. estagios 4 e 5
ATUALIZ  EQU   11                       ; controla atualiz. da ent. que variou
FIM_ENT  EQU   12                       ; indica fim das entradas no est. 5
NOVO_TMP EQU   13                       ; indica inicio de um novo tempo
*                                       ; de simulacao
CANCE_EV EQU   14                       ; indica que o ev. pode ser cancel.
FIM_PROG EQU   15                       ; fim da prog. de ev. do mestre
*
*
*
* definicao de filas
F_M_EV   EQU    1                       ; fila associada a memoria de ev.
F_M_EV_L EQU    2                       ; idem memoria de eventos livres
F_M_EV_E EQU    3                       ; idem mem. de ev. por elemento
F_M_CON  EQU    4                       ; idem memoria de conexoes
F_M_EST  EQU    5                       ; idem memoria de estados
F_M_FUNC EQU    6                       ; idem memoria de funcoes
F_M_AVAL EQU    7                       ; idem memoria de avaliacao
F_M_ATRA EQU    8                       ; idem memoria de atrasos
F_BUF1   EQU    9                       ; filas associadas aos buffers
F_BUF2A  EQU   10
F_BUF2B  EQU   11
F_BUF3   EQU   12
F_BUF3S  EQU   13
F_BUF4A  EQU   14
F_BUF4B  EQU   15
F_BUF5   EQU   16
F_BUF6   EQU   17
F_BUF_M  EQU   18
*
*
*
*
*
*
*
```

```
* definicao de facilidades
MEM1     EQU    1                          ; memoria de eventos
MEM2     EQU    2                          ; memoria de eventos livres
MEM3     EQU    3                          ; memoria de ev. por elemento
MEM4     EQU    4                          ; memoria de conexoes
MEM5     EQU    5                          ; memoria de estados
MEM6     EQU    6                          ; memoria de funcoes
MEM7     EQU    7                          ; memoria de avaliacao
MEM8     EQU    8                          ; memoria de atrasos
*
*
*
* definicao de buffers entre estagios
BUF1     EQU    1
BUF2A    EQU    2
BUF2B    EQU    3
BUF3     EQU    4
BUF3S    EQU    5
BUF4A    EQU    6
BUF4B    EQU    7
BUF5     EQU    8
BUF6     EQU    9
BUF_M    EQU   10
*
*
*
         RMULT  111,333,555,777,999,222,444
*
*
LOGON    MATRIX X,T_MAX,10          ; matriz para coleta de tempos de execucao
*
*
*
* variaveis auxiliares no calculo dos parametros
*
AUX1     VARIABLE  ( P$NRO_EV = 0 ) | ( P$EV_BLO = TAM_BLO )
*
AUX2     VARIABLE  FAN_MIN + ( (FAN_MAX - FAN_MIN + 1) * RN$3 ) / 1000
*
AUX3     VARIABLE  ENT_MIN + ( (ENT_MAX - ENT_MIN + 1) * RN$5 ) / 1000
*
AUX4     VARIABLE  1 + ( P$NRO_ENT * RN$5 / 1000 )
*
AUX5     VARIABLE  1 + ( 1000 / TAM_BLO )
*
AUX6     FVARIABLE 2000 / (EV_VAL * (FAN_MAX + FAN_MIN) / 1000 ) ; nro de ev.
*                                                                 ; a programar
*                                                                 ; em 1000 aval.
*
* Funcoes auxiliares no calculo dos parametros
*
ATIV     FUNCTION RN$1,D2
         .86,0/.999,1
*
*
```

```
EXPON     FUNCTION RN$1,C12
          0,0/.2,.222/.4,.509/.6,.915/.75,1.38/.84,1.83
          .9,2.3/.94,2.81/.96,3.2/.98,3.9/.995,5.3/.999,7
*
EV_MEST   FUNCTION  0,D1
          0,0
*
MAIS_ENT  FUNCTION RN$5,D2
          .95,0/.999,8
*
EVENTOS   FVARIABLE 186 * FN$EXPON
*
*
          SIMULATE
```

```
************                                        ************
************    PROCESSADOR MESTRE DA SIMULACAO     ************
************                                        ************
*
*
BUF_M    STORAGE  1
*
         GENERATE ,,,1
         LOGIC_S FIM_EV_M
         ADVANCE 1
         SAVEVALUE CONT_TMP,1                ; tempo inicial e' um
LBM0     ADVANCE 20
         ASSIGN NRO_EV_M,FN$EV_MEST          ; busca nro de ev. do mestre
LBM1     ADVANCE 2
         TEST_G P$NRO_EV_M,0,LBM3
         ADVANCE 1
         LOGIC_R FIM_EV_M
         QUEUE F_BUF_M
LBM2     ADVANCE 3
         GATE_SNF BUF_M,LBM2
         ADVANCE 4
         DEPART F_BUF_M
         ENTER BUF_M                         ; envia evento
         ADVANCE 1
         DECREMENT NRO_EV_M,1
         TRANSFER ,LBM1
LBM3     ADVANCE 1
         LOGIC_S FIM_EV_M                    ; sinaliza fim eventos
LBM4     ADVANCE 4
         GATE_SE BUF_M,LBM4                  ; garante sincronismo se NRO_EV_M=1
         ADVANCE 2
         GATE_LS FIM_PROG,LBM4               ; aguarda fim prog. ev. do mestre
         ADVANCE 1
         MSAVEVALUE LOGON,X$CONT_TMP,2,AC51 ; salva tempo inicial de simulacao
         SAVEVALUE  TP_EST1,0                ; zera acumuladores tempo p/ estag.
         SAVEVALUE  TP_EST2,0
         SAVEVALUE  TP_EST3,0
         SAVEVALUE  TP_EST4,0
         SAVEVALUE  TP_EST5,0
         SAVEVALUE  TP_EST6,0
         SAVEVALUE  TP_EST7,0
         LOGIC_S NOVO_TMP                    ; ativa estagio busca eventos
LBM5     ADVANCE 3
         GATE_SNE BUF3S,LBM6
         ADVANCE 2
         LEAVE BUF3S
         ADVANCE 5
LBM6     ADVANCE 7
         ADVANCE 2
         GATE_LS FIM_EV1,LBM5                ; aguarda fim em todos os estagios
         ADVANCE 2
         GATE_LS FIM_EV2,LBM5
         ADVANCE 2
         GATE_LS FIM_EV3,LBM5
         ADVANCE 2
```

```
GATE_LS FIM_EV4,LBM5
ADVANCE 2
GATE_LS FIM_EV5,LBM5
ADVANCE 2
GATE_LS FIM_EV6,LBM5
ADVANCE 2
GATE_LS FIM_EV7,LBM5
ADVANCE 1
MSAVEVALUE LOGON,X$CONT_TMP,3,AC$1 ; salva tempo final de simulacao
MSAVEVALUE LOGON,X$CONT_TMP,4,X$TP_EST1  ; salva tempo por estagio
MSAVEVALUE LOGON,X$CONT_TMP,5,X$TP_EST2
MSAVEVALUE LOGON,X$CONT_TMP,6,X$TP_EST3
MSAVEVALUE LOGON,X$CONT_TMP,7,X$TP_EST4
MSAVEVALUE LOGON,X$CONT_TMP,8,X$TP_EST5
MSAVEVALUE LOGON,X$CONT_TMP,9,X$TP_EST6
MSAVEVALUE LOGON,X$CONT_TMP,10,X$TP_EST7
SINCREMENT CONT_TMP,1              ; incrementa valor do tempo
ADVANCE 2
TEST_G X$CONT_TMP,T_MAX,LBM0       ; verifica tempo final de simulacao
TERMINATE 1
```

```
************                                        ************
************    Estagio BUSCA PROXIMO EVENTO        ************
************                                        ************
*
*
BUF1     STORAGE  1
*
         GENERATE  ,,,1
         LOGIC_S FIM_EV1
L80      GATE_LS NOVO_TMP
         LOGIC_R NOVO_TMP
         MARK
         ADVANCE 1
         LOGIC_R FIM_EV1
         TEST_NE FN$ATIV,TRUE,L8A          ; ha' evento para o tempo atual ?
         ASSIGN NRO_EV,0
         TRANSFER ,L88
L8A      ASSIGN NRO_EV,V$EVENTOS                ; calcula nro de eventos
L83      MSAVEVALUE LOGON,X$CONT_TMP,1,P$NRO_EV  ; salva nro de ev. do tempo
         MARK
         TRANSFER SBR,MEM_EV,12
         ADVANCE 2
         TEST_NE  P$NRO_EV,0,L89
L81      ASSIGN EV_BLO,0                   ; zera nro de eventos do bloco
         ADVANCE 2
         TRANSFER SBR,MEM_EV,12
         ADVANCE 4
L82      ADVANCE 2
         TRANSFER SBR,MEM_EV,12            ; inicia processamento do evento
         ADVANCE 3
         ASSIGN ENCAD,RN%2                 ;(EV_ENCAD/10)% dos ev. encadeados
         TEST_G  EV_VAL,RN%2,L84           ;(EV_VAL/10) % dos eventos validos
         TRANSFER SBR,MEM_EV,12
         ADVANCE 1
         TRANSFER SBR,MEM_EV,12
         ADVANCE 9
         TEST_G  EV_ENCAD,P$ENCAD,L87      ; verifica se ha' encadeamento
         ADVANCE 4
         TRANSFER SBR,MEM_EV_E,12
         ADVANCE 1
         TRANSFER SBR,MEM_EV_E,12
         ADVANCE 1
         TRANSFER ,L88
L87      ADVANCE 1
         TRANSFER SBR,MEM_EV_E,12
L88      QUEUE F_BUF1
         SAVEVALUE TP_EST1+,M$1            ; salva tempo de processamento
L86      ADVANCE 3                         ; envia evento para estagio 2
         GATE_SNF BUF1,L86                 ; aguarda, se buf1 esta cheio
         MARK
         ADVANCE 2                         ; escreve em buf1
         ENTER BUF1
         DEPART F_BUF1
         TRANSFER ,L85
L84      ADVANCE 2
```

```
L85     ADVANCE 2
        DECREMENT NRO_EV,1
        INCREMENT EV_BLO,1
        TEST_E  V$AUX1,TRUE,L82          ; testa fim dos eventos do bloco
        ADVANCE 3
        TRANSFER SBR,MEM_EV,12
        TRANSFER SBR,MEM_EV_L,12
        ADVANCE 7
        TEST_E P$NRO_EV,0,LB1            ; verifica fim dos eventos
LB9     ADVANCE 1
        SAVEVALUE TP_EST1+,M$1           ; salva tempo de processamento
        LOGIC_S FIM_EV1                  ; sinaliza fim de eventos
        TRANSFER ,L80
```

```
************                                                ************
************    Estagio BUSCA LISTA DE CONEXOES DE SAIDA    ************
************                                                ************
*
*
BUF2A    STORAGE  1
BUF2B    STORAGE  1
*
         GENERATE ,,,1
         LOGIC_S FIM_EV2
LB10     ADVANCE 4
         GATE_SE BUF1,LB11                ; trata evento se buf1 nao vazio
         ADVANCE 2
         GATE_LS FIM_EV1,LB10             ; se buf1 vazio e fim eventos
         ADVANCE 5                        ; esta setado finaliza
         LOGIC_S FIM_EV2
         TRANSFER ,LB10
LB11     MARK
         ASSIGN NRO_FAN,V$AUX2            ; fan out medio (FAN_MIN+FAN_MAX)/2
         ADVANCE 3
         LOGIC_R FIM_EV2
         ADVANCE 2                        ; le buf1
         LEAVE BUF1
         ADVANCE 6
         TRANSFER SBR,MEM_CON,12
         QUEUE F_BUF2A
         SAVEVALUE TP_EST2+,M$1           ; salva tempo de processamento
LB12     ADVANCE 3
         GATE_SNF BUF2A,LB12              ; aguarda, se buf2a esta cheio
         MARK
         ADVANCE 3                        ; escreve em buf2a
         SPLIT 1,LB29                     ; envia nro_fan ao prox. estagio
         ENTER BUF2A
         DEPART F_BUF2A
         ADVANCE 3
         TRANSFER SBR,MEM_CON,12
         ADVANCE 6
LB14     ADVANCE 2
         TEST_E P$NRO_FAN,0,LB16          ; testa fim da lista de fan out
         ADVANCE 2
         GATE_LS FIM_EV1,LB17
         ADVANCE 1
         LOGIC_S FIM_EV2
LB17     SAVEVALUE TP_EST2+,M$1           ; salva tempo de processamento
         TRANSFER ,LB10
LB16     TRANSFER SBR,MEM_CON,12
         ADVANCE 1
         TRANSFER SBR,MEM_CON,12
         ADVANCE 1
         QUEUE F_BUF2B
         SAVEVALUE TP_EST2+,M$1           ; salva tempo de processamento
LB15     ADVANCE 3
         GATE_SNF BUF2B,LB15              ; aguarda, se buf2b esta cheio
         MARK
         ADVANCE 2                        ; escreve em buf2b
```

```
ENTER BUF2B
DEPART F_BUF2B
ADVANCE 2
DECREMENT NRO_FAN,1
TRANSFER ,L814
```

```
************                                                    ************
************    Estagio EFETIVA EVENTO E VERIFICA MONITORACAO    ************
************                                                    ************
*
*
BUF3     STORAGE  1
BUF3S    STORAGE  1
*
         GENERATE ,,,1
         LOGIC_S FIM_EV3
LB20     ADVANCE 4
         GATE_SE BUF2A,LB21              ; trata evento se buf2a nao vazio
         ADVANCE 2
         GATE_LS FIM_EV2,LB20            ; se buf2a vazio e fim eventos
         ADVANCE 5                       ; esta setado finaliza
         LOGIC_S FIM_EV3
         TRANSFER ,LB20
LB21     MARK
         ADVANCE 3
         LOGIC_R FIM_EV3
         ADVANCE 3
         LOGIC_S BUSC_FAN                ; busca nro de fan out
         LEAVE BUF2A                     ; le buf2a
         ADVANCE 8
         TRANSFER SBR,MEM_EST,12
         ADVANCE 3
         TRANSFER SBR,MEM_EST,12
         ADVANCE 2
         TEST_G MONIT,RN$4,LB23          ; (MONIT/10)% das saidas monitoradas
         QUEUE F_BUF3S
         SAVEVALUE TP_EST3+,M$1          ; salva tempo de processamento
LB22     ADVANCE 3
         GATE_SNF BUF3S,LB22             ; aguarda, se buf3s esta cheio
         MARK
         ADVANCE 2
         ENTER BUF3S                     ; escreve em buf3s
         DEPART F_BUF3S
LB23     ADVANCE 1
LB24     ADVANCE 2
         TEST_E  X$NUM_FAN,0,LB25        ; verifica fim da lista de fan out
         ADVANCE 2
         GATE_LS FIM_EV2,LB2A
         ADVANCE 1
         LOGIC_S FIM_EV3
LB2A     SAVEVALUE TP_EST3+,M$1          ; salva tempo de processamento
         TRANSFER ,LB20
LB25     SAVEVALUE TP_EST3+,M$1          ; salva tempo de processamento
LB27     ADVANCE 3
         GATE_SNE BUF2B,LB27             ; aguarda, se buf2b esta vazio
         MARK
         ADVANCE 2
         LEAVE BUF2B                     ; le buf2b
         QUEUE F_BUF3
         SAVEVALUE TP_EST3+,M$1          ; salva tempo de processamento
LB26     ADVANCE 3
```

```
        GATE_SNF BUF3,L826              ; aguarda se buf3 esta cheio
        MARK
        ADVANCE 3
        ENTER BUF3                      ; escreve em buf3
        DEPART F_BUF3
        ADVANCE 2
        SDECREMENT NUM_FAN,1
        TRANSFER ,L824
*
L829    ADVANCE
        GATE_LS BUSC_FAN                ; faz uma fila com os valores
        LOGIC_R BUSC_FAN                ; de fan out, buscando-os 1 a 1
        SAVEVALUE NUM_FAN,P$NRO_FAN
        TERMINATE                       ; destroi copia da transacao
```

```
************                                                  ************
************   Estagio BUSCA E ATUALIZA ENTRADAS / BUSCA FUNCAO  ************
************                                                  ************
*
BUF4A    STORAGE  1
BUF4B    STORAGE  1
*
         GENERATE ,,,1
         LOGIC_S FIM_EV4
L830     ADVANCE 4
         GATE_SE BUF3,L831                ; trata evento se buf3 nao vazio
         ADVANCE 2
         GATE_LS FIM_EV3,L830             ; se buf3 vazio e fim eventos
         ADVANCE 5                        ; esta setado finaliza
         LOGIC_S FIM_EV4
         TRANSFER ,L830
L831     MARK
         ASSIGN NRO_ENT,V$AUX3            ; nro de entradas medio
         INCREMENT P$NRO_ENT,FN$MAIS_ENT  ; 5% dos elementos c/ + de 8 entr.
         ASSIGN POS_ENT,V$AUX4            ; posicao da entrada
         ADVANCE 3
         LOGIC_R FIM_EV4
         ADVANCE 3
         LEAVE BUF3                       ; le buf3
         ADVANCE 9
         TRANSFER SBR,MEM_FUNC,12
         TRANSFER SBR,MEM_FUNC,12
         ADVANCE 6
         TRANSFER SBR,MEM_EST,12
         ADVANCE 5
         QUEUE F_BUF4A
         SAVEVALUE TP_EST4+,M$1           ; salva tempo de processamento
L832     ADVANCE 3
         GATE_SNF BUF4A,L832              ; aguarda, se buf4a esta cheio
         MARK
         ADVANCE 3
         SPLIT 1,LB4A                     ; envia nro de ent. ao prox estagio
         DEPART F_BUF4A
         ENTER BUF4A                      ; escreve em buf4a
         ADVANCE 2
         TEST_LE P$NRO_ENT,8,L838
         ADVANCE 1
         TRANSFER SBR,MEM_EST,12
         ADVANCE 6
         TRANSFER ,L833
L838     ADVANCE 2
L833     ADVANCE 2
         ASSIGN CONT_ENT,0
         LOGIC_R ATUALIZ
L834     ADVANCE 2
         TEST_GE P$CONT_ENT,P$NRO_ENT,L837  ; verifica fim das entradas
         ADVANCE 2
         GATE_LS FIM_EV3,L839
         ADVANCE 1
         LOGIC_S FIM_EV4
```

```
LB39    SAVEVALUE TP_EST4+,M$1              ; salva tempo de processamento
        TRANSFER ,LB30
LB37    TRANSFER SBR,MEM_EST,12
        INCREMENT CONT_ENT,8
        ADVANCE 4
        GATE_LR ATUALIZ,LB35
        ADVANCE 2
        TEST_LE P$POS_ENT,P$CONT_ENT,LB35  ; verifica atualizacao da
        ADVANCE 6                          ; entrada que variou
        LOGIC_S ATUALIZ
        TRANSFER SBR,MEM_EST,12
LB35    QUEUE F_BUF48
        SAVEVALUE TP_EST4+,M$1              ; salva tempo de processamento
LB36    ADVANCE 3
        GATE_SNF BUF48,LB36
        MARK
        ADVANCE 1
        DEPART F_BUF48
        ENTER BUF48
        ADVANCE 2
        TRANSFER ,LB34
```

```
************                                                ************
************            Estagio AVALIA ELEMENTO             ************
************                                                ************
*
BUF5     STORAGE  1
*
         GENERATE ,,,1
         LOGIC_S FIM_EV5
LB40     ADVANCE 4                          ; trata elemento se buf4a vazio
         GATE_SE BUF4A,LB41
         ADVANCE 2
         GATE_LS FIM_EV4,LB40               ; se buf4a vazio e fim eventos
         ADVANCE 5                          ; setado finaliza
         LOGIC_S FIM_EV5
         TRANSFER ,LB40
LB41     MARK
         ADVANCE 3
         LOGIC_R FIM_EV5
         LOGIC_R FIM_ENT
         ADVANCE 3
         LOGIC_S BUSC_ENT                   ; busca nro de entradas do elem.
         LEAVE BUF4A                        ; le buf4a
         ADVANCE 6
         SAVEVALUE TP_EST5+,MS1             ; salva tempo de processamento
LB42     ADVANCE 3
         GATE_SNE BUF4B,LB42                ; aguarda se buf4b vazio
         MARK
         ADVANCE 1
         LEAVE BUF4B                        ; le buf4b
         ADVANCE 4
         TRANSFER SBR,MEM_AVAL,12
         ADVANCE 3
         SDECREMENT NUM_ENT,6
         ASSIGN PONT_ENT,6
LB43     ADVANCE 2
         TEST_LE XSNUM_ENT,0,LB45           ; verifica se terminaram as ent.
         ADVANCE 3
         QUEUE F_BUF5
         SAVEVALUE TP_EST5+,MS1             ; salva tempo de processamento
LB44     ADVANCE 3
         GATE_SNF BUF5,LB44                 ; aguarda se buf5 esta cheio
         MARK
         ADVANCE 2
         ENTER BUF5                         ; escreve em buf5
         DEPART F_BUF5
         ADVANCE 2
         GATE_LS FIM_EV4,LB48
         ADVANCE 1
         LOGIC_S FIM_EV5
LB48     SAVEVALUE TP_EST5+,MS1             ; salva tempo de processamento
         TRANSFER ,LB40                     ; volta a esperar novo elemento
LB45     ADVANCE 5
         TRANSFER SBR,NOVA_ENT,11
         ADVANCE 5
         GATE_LR FIM_ENT,LB46
```

```
         TRANSFER S8R,NOVA_ENT,11
         ADVANCE 5
         GATE_LR FIM_ENT,L846
         TRANSFER S8R,NOVA_ENT,11
         ADVANCE 5
         GATE_LR FIM_ENT,L846
         TRANSFER S8R,NOVA_ENT,11
         ADVANCE 5
         GATE_LR FIM_ENT,L846
         TRANSFER S8R,NOVA_ENT,11
         ADVANCE 3
L846     TRANSFER S8R,MEM_AVAL,12
         ADVANCE 1
         TRANSFER ,L843
*
*  subrotina
*
NOVA_ENT ADVANCE 4                          ; busca uma entrada
         INCREMENT PONT_ENT,1
         TEST_G P$PONT_ENT,8,L848           ; a cada 8 entradas busca novo
         SAVEVALUE TP_EST5+,M$1
L847     ADVANCE 3                          ; conjunto de 8
         GATE_SNE BUF48,L847
         MARK
         ADVANCE 1
         LEAVE BUF48
         ADVANCE 1
         ASSIGN PONT_ENT,1
L848     ADVANCE 10
         SDECREMENT NUM_ENT,1
         ADVANCE 3
         TEST_E X$NUM_ENT,0,L849            ; testa e sinaliza fim entradas
         LOGIC_S FIM_ENT
L849     TRANSFER P,11,1
*
L84A     ADVANCE
         GATE_LS BUSC_ENT                   ; faz uma fila com os valores de
         LOGIC_R BUSC_ENT                   ; nro de entradas buscando-os 1
         SAVEVALUE NUM_ENT,P$NRO_ENT        ; a 1.
         TERMINATE                          ; destroi as copias das transacoes
```

```
************                                                  ************
************  Estagio BUSCA ATRASO E VERIFICA EVENTOS PENDENTES  ************
************                                                  ************
*
BUF6     STORAGE  1
*
         GENERATE ,,,1
         LOGIC_S FIM_EV6
L850     ADVANCE 4                          ; trata elemento se buf5 vazio
         GATE_SE BUF5,L851
         ADVANCE 2
         GATE_LS FIM_EV5,L850               ; se buf5 vazio e fim eventos
         ADVANCE 5                          ; setado finaliza
         LOGIC_S FIM_EV6
         TRANSFER ,L850
L851     MARK
         ADVANCE 3
         LOGIC_R FIM_EV6
         ADVANCE 2
         LEAVE BUF5                         ; le buf5
         ADVANCE 9
         TRANSFER SBR,MEM_ATRA,12           ; busca atraso
         ADVANCE 8
         TRANSFER SBR,MEM_EV_E,12           ; busca indicacao de ev. pendentes
         ADVANCE 3
         TEST_G EV_PEND,RN$6,L853           ; (EV_PEN/10)% dos elementos
         ADVANCE 1                          ; avaliacoes tem eventos pendentes
         TRANSFER SBR,MEM_EV_E,12           ; busca end. do prim. ev. pendente
         ADVANCE 5
         TRANSFER SBR,BUSCA_EV,10
         ADVANCE 13
         TEST_G EV_ENCAD,RN$6,L852          ; (EV_ENCAD/10)% dos ev. possuem
         ADVANCE 2                          ; outro ev. encadeado
         TRANSFER SBR,PROX_EV,11            ; busca ev. encadeado
         ADVANCE 14
L852     ADVANCE 13
         GATE_LS CANCE_EV,L853              ; verifica se pode cancelar ev.
         LOGIC_R CANCE_EV
         ADVANCE 1
         TRANSFER SBR,MEM_EV,12             ; cancela evento
         TRANSFER SBR,PROX_EV,11            ; busca proximo ev.
         ADVANCE 2
L853     ADVANCE 5
         TEST_G V$AUX6,RN$6,L856            ; (AUX6/10)% das avaliacoes
*                                           ; geram eventos
         TEST_GE RN$6,EV_VAL,L854           ; sinaliza cancelamento de ev. em
         LOGIC_S CANCE_EV                   ; 100 - (EV_VAL/10)% dos eventos
L854     ADVANCE 18
         QUEUE F_BUF6
         SAVEVALUE TP_EST6+,M$1             ; salva tempo de processamento
L855     ADVANCE 3
         GATE_SNF BUF6,L855
         MARK
         ADVANCE 4
         ENTER BUF6
```

```
         DEPART F_BUF6
LB56     ADVANCE 2
         GATE_LS FIM_EV5,LB57
         ADVANCE 1
         LOGIC_S FIM_EV6
LB57     SAVEVALUE TP_EST6+,MS1              ; salva tempo de processamento
         TRANSFER ,LB50
*
* subrotinas
*
PROX_EV  ADVANCE 4                           ; busca prox. evento encadeado
         TRANSFER SBR,MEM_EV,12
         ADVANCE 1
         TRANSFER SBR,MEM_EV,12
         ADVANCE 5
         TRANSFER SBR,BUSCA_EV,10
         TRANSFER P,11,1
*
BUSCA_EV TRANSFER SBR,MEM_EV,12              ; busca evento e tempo associado
         ADVANCE 2
         TRANSFER SBR,MEM_EV,12
         ADVANCE 2
         TRANSFER P,10,1
```

```
************                                              ************
************              Estagio PROGRAMA EVENTOS        ************
************                                              ************
*
BUF6     STORAGE 1
*
         GENERATE ,,,1
         LOGIC_S FIM_EV7
         LOGIC_S FIM_PROG
LB60     ADVANCE 4                          ; verifica se ha' eventos
         GATE_SE BUF6,LB61                  ; do estagio anterior
         ADVANCE 2
         GATE_LS FIM_EV6,LB60
         ADVANCE 5
         LOGIC_S FIM_EV7                    ; sinaliza fim da programacao
LB67     ADVANCE 4
         GATE_SE BUF_M,LB68                 ; verifica se ha' eventos do
         ADVANCE 2                          ; mestre
         GATE_LS FIM_EV_M,LB67
         ADVANCE 5
         LOGIC_S FIM_PROG                   ; sinaliza fim da prog. de ev.
         TRANSFER ,LB60                     ; do mestre
LB61     MARK
         ADVANCE 3
         LOGIC_R FIM_EV7
         ADVANCE 4
         LEAVE BUF6
         TRANSFER SBR,ALOC_ESP,11
         ADVANCE 2
         TEST_G EV_PEND,RN57,LB63           ; ha' evento programado para
         TRANSFER SBR,MEM_EV,12             ; o elemento
         ADVANCE 2
         TRANSFER SBR,MEM_EV,12             ; busca evento e ajusta o
         ADVANCE 1                          ; encadeamento p/ novo evento
         TRANSFER SBR,MEM_EV,12
         ADVANCE 2
         TRANSFER SBR,MEM_EV,12
         ADVANCE 1
         TRANSFER SBR,MEM_EV,12
         ADVANCE 1
         TRANSFER ,LB64
LB63     ADVANCE 9                          ; e' o primeiro evento
         TRANSFER SBR,MEM_EV_E,12           ; atualiza MEM_EV_E
         ADVANCE 1
         TRANSFER SBR,MEM_EV_E,12
LB64     TRANSFER SBR,SALVA_EV,11
         SAVEVALUE TP_EST7+,M$1             ; salva tempo de processamento
         GATE_LS FIM_EV6,LB60
         ADVANCE 1
         LOGIC_S FIM_EV7
         TRANSFER ,LB67
LB68     ADVANCE 4
         LOGIC_R FIM_PROG                   ; trata evento enviado
         ADVANCE 4                          ; pelo mestre
         LEAVE BUF_M
```

```
          TRANSFER SBR,ALOC_ESP,11
          TRANSFER SBR,SALVA_EV,11
          GATE_LS FIM_EV_M,LB67
          ADVANCE 1
          LOGIC_S FIM_PROG
          TRANSFER ,LB60
*
*
* subrotinas
*
ALOC_ESP  ADVANCE 7
          TRANSFER SBR,MEM_EV_L,12          ; busca endereco do prox ev. livre
          ADVANCE 1
          TRANSFER SBR,MEM_EV_L,12
          ADVANCE 8
          TEST_G VSAUX5,RN57,LB62           ; ha' evento disponivel ?
          TRANSFER SBR,MEM_EV_L,12          ; busca novo bloco
          ADVANCE 10
          TRANSFER SBR,MEM_EV_L,12          ; atualiza novo bloco
          ADVANCE 2
          TRANSFER SBR,MEM_EV,12            ; ajusta ponteiro para novo bloco
          TRANSFER SBR,MEM_EV,12            ; na MEM_EV
          TRANSFER SBR,MEM_EV,12
          ADVANCE 2
LB62      TRANSFER P,11,1
*
*
SALVA_EV  TRANSFER SBR,MEM_EV,12            ; salva novo evento na MEM_EV
          ADVANCE 1
          TRANSFER SBR,MEM_EV,12
          ADVANCE 2
          TRANSFER SBR,MEM_EV,12
          ADVANCE 3
          TRANSFER SBR,MEM_EV,12
          ADVANCE 2
          TEST_G VSAUX5,RN57,LB65           ; novo ev. encheu o bloco ?
          ADVANCE 2
          TRANSFER SBR,MEM_EV_L,12          ; atualiza MEM_EV_L
          ADVANCE 1
          TRANSFER SBR,MEM_EV_L,12
          TRANSFER ,LB66
LB65      TRANSFER SBR,MEM_EV_L,12
          ADVANCE 1
          TRANSFER SBR,MEM_EV_L,12
          ADVANCE 1
LB66      ADVANCE 2
          TRANSFER P,11,1
```

```
********                          *******
********        SUBROTINAS        *******
********                          *******
*
*
MEM_EV   QUEUE  F_M_EV                    ; acesso a memoria de eventos
         SEIZE  MEM1
         DEPART F_M_EV
         ADVANCE 1
         RELEASE MEM1
         TRANSFER P,12,1
*
MEM_EV_L QUEUE  F_M_EV_L                  ; acesso a memoria de
         SEIZE  MEM2                      ; eventos livres
         DEPART F_M_EV_L
         ADVANCE 1
         RELEASE MEM2
         TRANSFER P,12,1
*
MEM_EV_E QUEUE  F_M_EV_E                  ; acesso a memoria de
         SEIZE  MEM3                      ; eventos por elemento
         DEPART F_M_EV_E
         ADVANCE 1
         RELEASE MEM3
         TRANSFER P,12,1
*
MEM_CON  QUEUE  F_M_CON                   ; acesso a memoria
         SEIZE  MEM4                      ; de conexoes
         DEPART F_M_CON
         ADVANCE 1
         RELEASE MEM4
         TRANSFER P,12,1
*
MEM_EST  QUEUE  F_M_EST                   ; acesso a memoria
         SEIZE  MEM5                      ; de estados
         DEPART F_M_EST
         ADVANCE 1
         RELEASE MEM5
         TRANSFER P,12,1
*
MEM_FUNC QUEUE  F_M_FUNC                  ; acesso a memoria
         SEIZE  MEM6                      ; de funcoes
         DEPART F_M_FUNC
         ADVANCE 1
         RELEASE MEM6
         TRANSFER P,12,1
*
MEM_AVAL QUEUE  F_M_AVAL                  ; acesso a memoria
         SEIZE  MEM7                      ; de avaliacao
         DEPART F_M_AVAL
         ADVANCE 1
         RELEASE MEM7
         TRANSFER P,12,1
*
*
```

```
MEM_ATRA QUEUE  F_M_ATRA                    ; acesso a memoria
         SEIZE  MEMB                        ; de atrasos
         DEPART F_M_ATRA
         ADVANCE 1
         RELEASE MEMB
         TRANSFER P,12,1
*
*
*
* fazer duas simulacoes: a 1º com buffers unitarios a 2º com buffers = 1000
*
         START     1
BUF1     STORAGE   1000
BUF2A    STORAGE   1000
BUF2B    STORAGE   1000
BUF3     STORAGE   1000
BUF3S    STORAGE   1000
BUF4A    STORAGE   1000
BUF4B    STORAGE   1000
BUF5     STORAGE   1000
BUF6     STORAGE   1000
BUF_M    STORAGE   1000
         RMULT   111,333,555,777,999,222,444
         CLEAR
         START 1
         END
```

# BIBLIOGRAFIA

/ABR 83/ ABRAMOVICI,M. et alii. A logic simulation machine. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, New York, CAD-2(2):82-94, Apr. 1983.

/ADV 85/ ADVANCED MICRO DEVICES. Bipolar microprocessor logic and interface databook. 1985.

/AGE 82/ AGERVALA,T. & ARVIND. Data flow systems. Computer, Los Angeles, 15(2):10-3, Feb. 1982.

/AGR 86/ AGRAVAL,P. Concurrency and communication in hardware simulators. IEEE Transactions on Computer-Aided Design, New York, CAD-5(4):617-23, Oct. 1986.

/ALL 85/ ALLENBAUGH,D.W. Attacking the simulation bottleneck. Digital Design, Boston, 15(7):79-85, July 1985.

/BAR 80/ BARTO,R. & SZYGENDA,S.A. A computer architecture for digital logic simulation. Electronic Engineering, London, 52(642):35-66, Sept. 1980.

/BLA 84/ BLANK,T. A survey of hardware accelerators used in CAD. IEEE Design & Test, Los Alamitos, 1(3):21-39, Aug. 1984.

/BLO 87/ BLOOM,M. More need in accelerators for multilevel simulation. Computer Design, Littleton, 26(7):26-32, Apr. 1987.

/BRE 76/ BREUER,M.A. & FRIEDMAN,A.D. Diagnosis & reliable design of digital systems. Computer Science Press, California, 1976.

/BRY 80/ BRYANT,R.E. An algorithm for MOS logic simulation. Lambda, Palo Alto,1:46-53, 1980.

/BRY 84/ BRYANT,R.E. A switch-level model and simulator for MOS digital systems. IEEE Transactions on Computers, New York, C33(2):160-77, Feb. 1984.

/CAS 78/ CASE,R.G. & STAUFFER,J.D. SALOGS IV - a program to perform logic simulation and fault diagnosis. In: DESIGN AUTOMATION CONFERENCE, 15., Las Vegas, June 19-21, 1978. Proceedings. New York, IEEE, 1978. p.392-97.

/CHA 71/ CHAPPELL,S.G. & YAU,S.S. Simulation of large asynchronous logic circuits using an ambiguous gate model. In: FALL JOINT COMPUTER CONFERENCE, Las Vegas, Nov. 16-18, 1971. Proceedings. Montvale, AFIPS Press, 1971. p.651-61.

/CIR 83/ CIRRUS COMPUTERS. HILO 2 user's manual. July 1983.

/D'A 85/ D'ABREU,M.A. Gate level simulation. IEEE Design & Test, Los Alamitos, 2(6):63-71, Dec. 1985.

/DEN 82/ DENNEAU,M.M. The Yorktown Simulation Engine. In: DESIGN AUTOMATION CONFERENCE, 19., Las Vegas, June 14-16, 1982. Proceedings. New York, IEEE, 1982. p.55-9.

/FLA 83/ FLAKE,P.L. et alii. An algebra for logic strengh simulation. In: DESIGN AUTOMATION CONFERENCE, 20., Miami, June 27-29, 1983. Proceedings. New York, IEEE, 1983. p.615-8.

/FRA 84/ FRANKLIN,M.A. et alii. Parallel machines and algorithms for discrete_event simulation. In: INTERNATIONAL CONFERENCE ON PARALLEL PROCESSING, Bellaire, Aug. 21-24, 1984. Proceedings. ACM/IEEE, 1984. p.449-58.

/GLA 84/ GLAZIER,M.E. & AMBLER,A.P. ULTIMATE: a hardware logic simulation engine. In: DESIGN AUTOMATION CONFERENCE, 21., 1984. Proceedings. New York, IEEE, 1984. p.336-42.

/GOE 86/ GOERING,R. Simulation challenges breadboarding for design verification. Computer Design, Littleton, 25(12):63-82, June 1986.

/GOE 87/ GOERING,R. CAE/CAD tools shift focus to system-level design. Computer Design, Littleton, 26(12):59-76, June 1987.

/GOE 88/ GOERING,R. Simulation accelerators address throughput issues. Computer Design, Littleton, 27(6):42-51, Mar. 1988.

/GRE 72/ GREENBERG,S. GPSS primer. New York, Wiley-Interscience, 1972.

/HAY 86/ HAYES,J.P. Digital simulation with multiple logic values. IEEE Transactions on Computer-Aided Design, New York, CAD-5(2):274-83, Apr. 1986.

/HWA 84/ HWANG,K. & BRIGGS,F.A. Computer architecture and parallel processing. New York, McGraw-Hill, 1984.

/IVE 82/ IVERSEN,W.R. Modular hardware simulates logic. Electronics, New York, 55(15):125-6, July 1982.

/JEF 85/ JEFFERSON,D.R. Virtual time. ACM Transactions on Programming Languages and Systems, New York, 7(3):404-25, July 1985.

/KOI 85/ KOIKE,N. et alii. HAL: a high-speed logic simulation machine. IEEE Design & Test, Los Alamitos, 2(5):61-73, Oct. 1985.

/KRO 81/ KROHN,H.E. Vector coding techniques for high speed digital simulation. In: DESIGN AUTOMATION CONFERENCE, 18., Nashville, June 29 - July 1, 1981. Proceedings. New York, IEEE, 1981. p.525-9.

/KRO 82/ KRONSTADT,E. & PFISTER,G. Software suport for the Yorktown Simulation Engine. In: DESIGN AUTOMATION CONFERENCE, 19., Las Vegas, June 14-16, 1982. Proceedings. New York, IEEE, 1982. p.60-4.

/LEV 82/ LEVENDEL,Y.H. et alii. Special-purpouse computer for logic simulation using distributed processing. The Bell System Technical Journal, New York, 61(10):2873-909, Dec. 1982.

/MEY 85/ MEYER,E.L. Application-specific hardware accelerators: an overview. VLSI Systems Design, Palo Alto, 6(10):48-59, Oct. 1985.

/MIC 87/ MICZO,A. et alii. The effects of modeling on simulator performance. IEEE Design & Test, Los Alamitos, 4(2):46-54, Apr. 1987.

/MOT 85/ MOTT,G. & HALL,R.B. The utility of hardware accelerators in the design environment. VLSI Systems Design, 6(10):62-70, Oct. 1985

/MOT 86/ MOTT,G. Advanced logic simulation deliver breakneck speed in multiuser environments. Digital Design, Boston, 16(5):49-53, Apr. 1986.

/NAT 83/ NATIONAL SEMICONDUCTOR CORPORATION. NS16000 databook. 1983.

/ODO 79/ O'DONAVAN,T.M. GPSS; simulation made simple. Chichester, John Wiley, 1979.

/PAS 85/ PASEMAN,W.G. Data flow concepts speed simulation in CAE systems. Computer Design, Littleton, 24(1):131-40, Jan. 1985.

/RAM 77/ RAMAMOORTHY,C.V. & LI,H.F. Pipeline architecture. Computing Surveys, New York, 9(1):62-102, Mar. 1977.

/SAN 83/ SANGSTER,A. & MONAHAN,J. AQUARIUS: logic simulation on an engineering workstation. In: DESIGN AUTOMATION CONFERENCE, 20., Miami, June 27-29, 1983. Proceedings. New York, IEEE, 1983. p.93-9.

/SMI 86/ SMITH,R.J. Fundamentals of parallel logic simulation. In: DESIGN AUTOMATION CONFERENCE, 23., Las Vegas, June 29 - July 2, 1986. Proceedings. New York, IEEE, 1986. p.2-12.

/STE 83/ STEVENS,P. & ARNOUT,G. BIMOS - a MOS oriented multi-level logic simulation. In: DESIGN AUTOMATION CONFERENCE, 20., Miami, June 27-29, 1983. Proceedings. New York, IEEE, 1983. p.100-2.

/SZY 70/ SZYGENDA,S.A. ; ROUSE,D.M. ; THOMPSON,E.W. A model and implementation of a universal time delay simulator for large digital nets. In: SPRING JOINT COMPUTER CONFERENCE, Atlantic City, N. J., May 5-7, 1970. Proceedings. Montvale,AFIPS Press, 1970. p.207-16.

/SZY 75/ SZYGENDA,S.A. & THOMPSON,E.W. Digital logic simulation in a time-base, table driven environment. Part1: design verification. Computer , Los Angeles, 8(3):24-36, Mar. 1975.

/TAK 86/ TAKASAKI,S. et alii. HAL II - a mixed level hardware logic simulation system. In: DESIGN AUTOMATION CONFERENCE, 23., Las Vegas, June 29 - July 2, 1986. Proceedings. New York, IEEE, 1986. p.581-7.

/ULR 69/ ULRICH,E.G. Exclusive simulation of activity in digital networks. Communications of the ACM, New York, 12(2):102-10, Feb. 1969.

/ULR 72/ ULRICH,E.G. et alii. Fault-test analysis techniques based on logic simulation. In: DESIGN AUTOMATION WORKSHOP, 9., Dallas, June , 1972. Proceedings. ACM/IEEE, 1972. p.111-5.

/WAG 83/ WAGNER,F.R. Ueber die Austauschbarkeit von Universalitaet und Effizienz bei Instanzennetzsimulatoren, insbesodere fuer digitale Hardware. Kaiserslautern, Universidade de Kaiserslautern, 1983. 176p. ( tese de doutorado )

/WAG 84a/ WAGNER,F.R. Basic techniques of gate level simulation. Porto Alegre, PGCC da UFRGS, 1984. (RT12)

/WAG 84b/ WAGNER,F.R. Hazard detection in logic simulation. Porto Alegre, PGCC da UFRGS, 1984. (RT13)

/WAG 85/ WAGNER,F.R. On the properties of event oriented logic simulation according to significant timing models. Porto Alegre, PGCC da UFRGS, 1985. (RT22)

/WAG 87/ WAGNER,F.R. & FREITAS,C.M.D.S. NILO - uma linguagem para descrição de hardware no nível de portas lógicas. Porto Alegre, PGCC da UFRGS, 1987. (RP66)

/WIL 83/ WILLIAMS,G. Benchmarking the Intel 8086 and 8088. Byte, Peterborough, 8(7):147-62, July 1983.

/WON 86/ WONG,K.F. et alii. Statistics on logic simulation. In: DESIGN AUTOMATION CONFERENCE, 23., Las Vegas, June 29 - July 2, 1986. Proceedings. New York, IEEE, 1986. p.13-9.

/WYA 83/ WYATT,D.L. et alii. An experiment in microprocessor-based distributed digital simulation. In: 1983 WINTER SIMULATION CONFERENCE, 1983. Proceedings. New York, IEEE, 1982. p.271-7.

/ZIL 82/ ZILOG INC. Zilog data book. 1982.