Governo do Estado de São Paulo

Universidade Estadual de Campinas

Uma metodologia para a simulação de circuitos ULSI

Norian Marranghello

Este exemplar corresponde à redação final da tese defendida por NORIAN MARRANGHELLO e aprovada pela Comissão Julgadora em 27 / 02 / 92.

Orientador

Tese apresentada à CPG/FEE/UNICAMP em 27 de fevereiro de 1992, como um dos requisitos para a obtenção do título de Doutor em Engenharia Elétrica.

Faculdade de Engenharia Elétrica

Departamento de Semicondutores, Instrumentos e Fotônica

13081 Campinas SP CP 6101 Fax:(0192) 391395 E-Mail: Postmaster@fee.unicamp.br

Banca Examinadora

Presidente: Prof.Dr.Furio Damiani (DSIF/FEE/UNICAMP - orientador)

Membros Titulares: Profª.Drª.Edith Ranzini (LSD/DEE/EPUSP)
Prof.Dr.João Antônio Zuffo (LSI/DEE/EPUSP)
Prof.Dr.Peter Jürgen Tatsch (DEMIC/FEE/UNICAMP)
Prof.Dr.Sérgio Santos Mühlen (DEB/FEE/UNICAMP)

Membros Suplentes: Prof.Dr.Carlos Alberto dos Reis Fº (DEMIC/FEE/UNICAMP)
Prof.Dr.Wilmar Bueno de Moraes (DEMIC/FEE/UNICAMP)

Agradecimentos

Gostaria de agradecer ao Prof. Furio pela valiosa orientação que me dispensou durante este trabalho. *Grazie mille!*

Gostaria, também, de agradecer a todos os membros da banca examinadora, em particular ao Prof. Peter pelas mui frutíferas discussões sobre o modelamento dos dispositivos.

Meus agradecimentos à Srta. Edna Servidone, sempre pronta a auxiliar com sua experiência em secretaria, nas várias dúvidas de redação e pela companhia nos incontáveis cafezinhos. Também, à Sra. Lúcia Cardoso pela confecção de várias figuras desta tese.

Também agradeço o apoio de vários colegas da FEE/UNICAMP, particularmente ao Mauro e à Berenice pelo auxílio no modelamento do ABACUS em redes de Petri; ao Gorgônio, pelo auxílio nos meus primeiros passos com a liguagem C; e aos colegas do DSIF/FEE/UNICAMP que se sensibilizaram com a minha premência por micros, nos últimos meses, muitas vezes, abdicando de seu uso a meu favor.

Aos meus pais, Nadyr e Ítalo, e à minha avó, Alaíde, minha eterna gratidão pela ajuda e pelo incentivo em tudo o que busquei realizar na vida. Aos demais amigos e familiares agradeço o apoio que recebi, particularmente da minha mulher Sandra, cuja companhia e compreensão foram indispensáveis nas horas mais árduas; a ela dedico esta tese.

Agradeço, também, o apoio financeiro, na forma de bolsas de estudo, recebido do Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) e da Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP), durante a realização deste trabalho; e à IBM - Brasil que em 1990 concedeu-me uma passagem para a apresentação de um artigo sobre esta tese num congresso em Calgary, Canadá.

Índice

Sinopse

Les hommes sont compliqués,
il faut tout leur expliquer ...
A.Saint-Exupéry
Le Petit Prince

Nesta tese, propõe-se uma metodologia para a simulação de circuitos eletrônicos, onde buscou-se explorar ao máximo as possibilidades oferecidas pela incorporação das técnicas mais atuais, tanto no aspecto de arquiteturas computacionais, quanto no de produção de sistemas integrados. Esta metodologia utiliza um arranjo de processadores dedicados, onde a computação se dá assincronamente, dirigida apenas pelas condições de contorno e pelo fluxo das informações geradas a partir destas.

Les machines, c'est pire encore...!
G.G.Boulaye
La Microprogrammation
Dunod, Paris, 1971

Introdução

A leitura, após certa idade, distrai
excessivamente o espírito humano de suas
reflexões criadoras...
Albert Einstein [113]

A crescente importância das ferramentas de Projeto Auxiliado por Computador (PAC) na Microeletrônica é motivada pela necessidade de se reduzir o tempo decorrente entre a concepção e a comercialização de um novo circuito integrado[074,079,134]. Este fenômeno, que ocorre também em outras áreas, tem indicado a necessidade de se automatizarem algumas fases tanto do projeto como da fabricação de circuitos integrados. Particularmente, com o advento dos circuitos integrados (CIs) com escala de integração muito grande (VLSI), as ferramentas de PAC tornaram-se indispensáveis.

Atualmente, existem ferramentas de PAC que atendem praticamente a todos os níveis de um projeto[003,012,051,120,192], indo desde a especificação funcional[021,160] do circuito desejado até o desenho das formas geométricas do circuito final[059,100,105]. Estas ferramentas possibilitam a simulação do circuito e do processo de fabricação, já nas fases preliminares de projeto, e, assim, possibilitam a análise e a verificação das idéias do projetista, sem que estas necessitem ser implementadas fisicamente - o que seria impraticável.

O projeto e a fabricação de circuitos integrados são subdivididos em várias fases intermediárias[168], organizadas hierarquicamente. Algumas das fases do projeto envolvem a simulação do circuito que está sendo projetado[127,186]. Na simulação de circuitos são utilizadas ferramentas de PAC, desenvolvidas para representar o comportamento do CI, considerando-se diferentes níveis de abstração. No nível de abstração mais elevado, é modelada a arquitetura do sistema - são considerados apenas o comportamento e as interações de grandes blocos, ou subsistemas. No nível mais baixo, são representados os detalhes do comportamento elétrico do circuito - aqui entram os modelos elétricos de componentes como transistores, resistores, etc. Para evitar-se o crescimento excessivo da complexidade do problema, nas simulações em níveis de abstração mais baixos, são representados apenas subsistemas, aproveitando-se, assim, do fraco acoplamento existente entre as partes (subsistemas) de um

sistema complexo.

Apesar de não existir nem uma tipologia rígida, nem uma maneira fixa de se utilizarem simuladores de circuitos, eles podem ser agrupados de acordo com os níveis de abstração considerados na modelagem dos circuitos em estudo. Assim, para circuitos digitais, uma classificação possível agrupa os simuladores em cinco grupos[002,007]:

1.simuladores em nível de arquitetura[067,068]: permitem uma descrição funcional em alto nível dos grandes blocos que compõem o circuito (saídas em função de entradas e estados internos);

2.simuladores em nível de registradores[056]: os sistemas são descritos por modelos discretos de blocos mais detalhados como, por exemplo, contadores e registradores de deslocamento;

3.simuladores em nível de lógica[008,055,057,080,081,114,144]: os circuitos são descritos a nível de portas lógicas (NE, NOU, NÃO, etc.) e o modelo é excitado de modo a se obter os valores lógicos (0, 1, etc.) das saídas do circuito em função do tempo para determinadas seqüências de sinais aplicadas às suas entradas;

4.simuladores em nível de temporização[042,058,062,190]: utilizam técnicas aproximadas de simulação de circuitos, com modelos simplificados de transistores, para a avaliação dos sinais de um circuito e técnicas de simulação lógica para a propagação dos valores dos sinais;

5.simuladores em nível de circuitos[098,119,138,150,167]: prevêem o desempenho elétrico detalhado de um circuito eletrônico através da utilização de modelos matemáticos ou numéricos dos dispositivos eletrônicos simulados, cujos parâmetros são obtidos por medidas em laboratório sobre dispositivos fabricados pelo mesmo processo a ser utilizado na integração do circuito.

Esta é uma tipologia bastante flexível. Dependendo do circuito, alguns dos níveis de abstração podem ser mais importantes que os demais. Em particular, para circuitos analógicos alguns dos níveis não se aplicam.

Esta tese trata especificamente de simuladores de circuitos. A simulação de circuitos, tradicionalmente, tem sido dividida em simulação lógica e analógica[127]. As técnicas utilizadas em cada um desses dois tipos de simulação são bastante distintas. Assim, na simulação analógica[004,128], são resolvidas equações algébrico-diferenciais utilizando métodos implícitos de integração numérica, método de Newton para solução de sistemas algébricos não-lineares e método de eliminação de Gauss para a resolução de sistemas lineares esparsos. Já no que se refere à simulação lógica, a técnica básica de simulação está centrada no escalonamento

de eventos (transições de estados)[044,045,046,050,057,140,157]. Além disso, na simulação lógica trabalha-se em um nível de abstração bastante elevado, no qual são ignorados detalhes de funcionamento elétrico dos componentes (assim, os transistores são tratados como simples chaves). Como conseqüência, os simuladores lógicos são rápidos, quando comparados com os simuladores analógicos convencionais, porém, não fornecem detalhes precisos do comportamento elétrico do circuito ao longo do tempo. A tendência atual das pesquisas na área se baseia no aproveitamento do que há de melhor nos dois níveis: simplicidade e rapidez da simulação lógica, aliada à riqueza de detalhes de comportamento, fornecida pelos simuladores analógicos. O segredo para se aliar eficiência com detalhe de representação está em se utilizarem procedimentos adaptativos, através dos quais são resolvidas apenas as partes relevantes dos circuitos: por exemplo, em circuitos digitais que utilizam a tecnologia MOS, parte significativa do circuito pode permanecer dormente (sem mudar de estado) durante parte da simulação. Para se tirar partido dessa característica, são resolvidas as partes relevantes do problema através de técnicas usuais de simulação analógica; por outro lado, para se saber o que é realmente relevante, são utilizadas técnicas de escalonamento de eventos do tipo tradicionalmente utilizado por simuladores lógicos.

Além disso, os novos simuladores híbridos do tipo descrito no parágrafo precedente utilizam os mais recentes avanços disponíveis no que se refere ao *hardware*, ou sejam, as diversas arquiteturas paralelas existentes. Como exemplo das arquiteturas mais em voga, nos últimos anos, vide referências [023, 031, 043, 049, 065, 082, 086, 116, 117, 145, 164, 165, 170, 171, 175, 177, 180]. Como exemplo de suas aplicações em simulação de circuitos vide referências [041, 066, 102, 107, 141, 154, 156, 179, 191].

O objetivo deste trabalho é o de propor uma nova metodologia para a simulação de circuitos[130], que seja capaz de tirar o maior proveito possível do casamento das mais eficientes variações arquiteturais com as mais avançadas técnicas de simulação; para viabilizar a simulação elétrica de circuitos ULSI, com milhões de componentes.

Descrição da Metodologia Proposta

O mecanismo do descobrimento não é lógico e intelectual - é uma iluminação subitânea, quase um êxtase. Em seguida, é certo, a inteligência analisa e a experiência confirma a intuição...

Albert Einstein[113]

1. Introdução:

Como foi mencionado na introdução, a simulação de circuitos elétricos tem sido de muita importância para a engenharia elétrica, nos últimos anos. A densidade sempre crescente dos circuitos integrados (CIs) tem tornado imperioso o uso de ferramentas de *software* no auxílio do projeto e da fabricação de circuitos integrados. Entre estas ferramentas, podemos destacar os simuladores de circuitos, que são ferramentas capazes de predizer o comportamento elétrico de um circuito com detalhes suficientes para prover uma idéia muito aproximada de seu comportamento real.

Em 1975 foi apresentado o programa SPICE2, na Universidade da Califórnia, em Berkeley[111], que tem sido um dos simuladores de circuitos mais usados em todo o mundo. O SPICE2 tem uma precisão muito boa na previsão das formas de onda, mas tem um custo computacional bastante alto. Buscaram-se novas técnicas e a última geração de simuladores de circuitos conseguiu um aumento de velocidade de aproximadamente 10 vezes em relação à geração do SPICE2, sem muitas perdas no que se refere à precisão. Apesar destas melhorias, os simuladores de circuitos atuais não são suficientemente velozes para tratar adequadamente circuitos VLSI. Nesta tese, propõe-se uma metodologia para a simulação de circuitos de altíssima escala de integração, a qual poderá, uma vez implementada adequadamente, contribuir, grandemente, para a solução dos problemas de velocidade presentes nos simuladores atuais. Neste capítulo, apresenta-se uma revisão dos simuladores clássicos de circuitos. A seguir, descreve-se a metodologia proposta[130], enfatizando o algoritmo utilizado[131] e reforçando o aspecto da arquitetura subjacente[129].

2. Simuladores clássicos de circuitos:

As pesquisas sobre simulação de circuitos auxiliada por computador teve início na década de 1950[040]. Contudo, somente em meados dos anos '70, com o surgimento dos programas SPICE2[013,111] e ASTAP[149,187], é que começaram a aparecer simuladores de circuitos práticos. Estes programas alcançaram uma grande aceitação e são amplamente utilizados, inclusive como base de comparação para programas mais recentes. Por isso os membros desta geração são conhecidos como simuladores padrão de circuitos[018,111,149,187].

A principal diferença entre os simuladores padrão e seus precursores[052,053,075,076,087,110,174,182,183] é o uso de métodos de integração numérica mais estáveis e de técnicas adequadas para o tratamento de matrizes esparsas. O algoritmo básico empregado pelos simuladores padrão de circuitos é o seguinte[004,005,127,128]:

- A partir de um sistema de equações algébrico-diferenciais não lineares, que descreve o circuito, deriva-se um sistema de equações algébricas não lineares, através de um método de integração numérica implícito e rigidamente estável[017,093];

- O sistema de equações algébricas não lineares obtido é então linearizado, por meio de um algoritmo iterativo, como o de Newton-Raphson[098,181];

- Finalmente, o sistema de equações algébricas lineares é resolvido pelo método de Gauss, devidamente modificado para o tratamento eficiente de matrizes esparsas[095,096,118].

O principal problema dos simuladores padrão vem a ser o grande tempo de CPU necessário para resolver o sistema de equações resultantes de circuitos LSI e VLSI[006]. Pesquisas, no sentido de se obter simuladores de circuitos mais eficientes, começaram no final dos anos '70 e buscavam desenvolver[127]:

-novos métodos de análise numérica para serem usados na solução das equações de circuito - vide, por exemplo, as referências [024, 030, 036, 037, 060, 090, 124, 159, 178];

-novas implementações dos programas de simulação já existentes para tirar proveito dos avanços das arquiteturas de computadores e dos sistemas operacionais - vide, por exemplo, as referências [019, 025, 029, 034, 073, 084, 091, 094, 135, 137, 152];

-novos modelos computacionais simplificados dos elementos de circuito - vide, por exemplo, as referências [016, 020, 026, 115, 123, 147, 161, 168, 185].

Inicialmente, os novos simuladores de circuito propostos mantiveram várias técnicas dos simuladores padrão, para manter suas características de precisão, e incorporaram muitas técnicas usadas nos simuladores lógicos, para acelerar as computações. Estes simuladores

são chamados de simuladores de modo misto[009,010,064,070,071,072,103,106,148], pois, desacoplam o sistema de equações do circuito, simulando suas partes críticas com modelos elétricos, mais precisos, e o resto do circuito com modelos lógicos, mais simplificados.

A terceira geração de simuladores[004,018,061,063,104,124,163] desacopla o sistema de equações algébricas não lineares, com um passo de um método de relaxação e então resolve o sistema desacoplado com um método iterativo. Ocorre que, quando o processo de relaxação não é levado até sua convergência, como é o caso destes simuladores, o rompimento de laços de realimentação e de nós fortemente acoplados, gera instabilidades nos algoritmos e imprecisões nos resultados[007]. Para minimizar estas instabilidades é necessária a utilização de intervalos de tempo muito pequenos nos processos iterativos, levando a altos custos computacionais, reduzindo o ganho de velocidade do processo de relaxação. Além disso, estes simuladores têm um outro problema, mais grave, que motivou sua reduzidíssima aceitação: a baixa confiabilidade em seus resultados[006].

A quarta geração de simuladores[032,033,047107,108,142,143,162] aplica o método de relaxação diretamente ao sistema de equações algébrico-diferenciais não lineares. Eles são conhecidos como simuladores de relaxação por forma de onda[006]. Na tentativa de melhorar a simulação de circuitos, muitos dos simuladores desta geração utilizam computadores paralelos, além de técnicas especiais, tais como certos conceitos das máquinas a fluxo de dados e algoritmos de Gauss-Seidel coloridos[108].

As abordagens, descritas nesta seção, têm se mostrado eficientes e alcançaram um aumento de velocidade de uma ordem de grandeza em relação aos simuladores padrão, conservando praticamente o mesmo nível de detalhes do comportamento previsto de circuitos VLSI. Contudo, eles ainda não são suficientemente rápidos, para tratar adequadamente aqueles circuitos.

3. Proposta de uma metodologia:

Sabe-se que a parte de maior custo computacional (que gasta mais tempo) nos procedimentos clássicos de simulação de circuitos é o cômputo da matriz de equações do circuito[006,127,137]. Então, parece razoável atacar justamente este problema. Nos últimos anos, pode-se notar uma tendência para a utilização de aceleradores de *hardware* na simulação de circuitos[141], de modo a se conseguir simulações mais rápidas, sem perder o grau de precisão no comportamento previsto para o circuito. Contudo, estes aceleradores usam os mesmos

métodos clássicos de simulação de circuitos, adaptados para aproveitar as arquiteturas paralelas existentes. Portanto, têm seu desempenho limitado por aqueles métodos.

Nossa proposta consiste em utilizar um *hardware* dedicado para a simulação de circuitos[131], mas aplicando a este uma nova metodologia de simulação[130], a qual evita a solução dos imensos sistemas de equações algébrico-diferenciais não lineares resultantes dos circuitos VLSI[128], pelo uso de um arranjo especial de processadores, os quais incorporam modelos dos elementos de circuito e sobre o qual é possível mapeá-lo[129]. Chamamos esta arquitetura de ABACUS (***hArdware BAsed CircUit Simulator*** - Fig.1). Este arranjo de processadores é composto por um processador mais complexo que serve de hospedeiro, ou gerente, do arranjo e por um grande número de processadores mais simples. Cada um dos processadores é chamado de MPH (***Model Processing Hardware-element***) e dedica-se à tarefa exclusiva de simular os modelos dos componentes do circuito.

Além das tarefas de interfaceamento com o usuário, o processador hospedeiro é responsável pelo gerenciamento do arranjo de MPHs. O gerente lê a *netlist* de entrada e as demais informações referentes à simulação. A seguir, identifica os elementos de circuito, suas interconexões e as análises a serem feitas. Se o número de elementos do circuito é tal que não é possível mapeá-lo diretamente no arranjo, o gerente particiona o circuito num número adequado de subcircuitos e, então, mapeia sobre o arranjo cada subcircuito a seu turno, fazendo as conexões necessárias entre cada subcircuito. A configuração do arranjo é estabelecida sempre que um novo circuito é nele mapeado. Isto é possível graças a uma rede programável de barramentos, cujas chaves são uniformemente distribuídas por todo o arranjo, de maneira que o maior número possível de variações nas interconexões seja atingível, com um número razoável de barramentos e de circuitos de chaveamento. Após mapear o circuito no arranjo, o gerente transfere os parâmetros relativos a cada elemento e as informações sobre as análises a serem feitas, para os processadores correspondentes. Enquanto espera pelos resultados da simulação, o gerente, além de controlar a convergência global do arranjo, pode fazer o tratamento dos dados disponíveis para a saída. Quando o arranjo chega a uma solução, o hospedeiro pára momentaneamente seu trabalho, recupera os resultados disponíveis, reinicializa o arranjo e retoma seu processamento. Após todas as análises terem sido resolvidas pelo arranjo, o gerente apresenta os resultados na forma solicitada pelo usuário, terminando a simulação.

O arranjo de MPHs é composto por um conjunto de processadores especiais, os quais trabalham assincronamente, de acordo com um princípio de operação de fluxo de dados. A interconexão entre os MPHs do arranjo é reconfigurável dinamicamente pelo hospedeiro, para

refletir a topologia do circuito sendo simulado. Cada MPH do arranjo emula, em princípio, um dispositivo por análise e o arranjo executa a simulação emulando o comportamento do circuito real. Antes do início da simulação, todos os MPHs estão num estado desconhecido. Quando o arranjo é inicializado pelo gerente, cada MPH acessa suas portas de entrada na busca por dados. Na primeira tentativa, apenas aqueles MPHs que estiverem emulando as entradas primárias do circuito, encontrarão dados de entrada reais. Após computarem suas respostas, estas serão colocadas nas suas portas de saída de modo que os processadores que estiverem ligados a estas portas possam encontrá-los e calcular suas respostas. Enquanto estes processadores computam suas análises, aqueles iniciam um segundo ciclo de cálculo, buscando novos dados nas entradas, processando-os e transferindo os resultados para as saídas.

Ao concluir cada instante de simulação (isto é, quando todos os processadores convergiram para uma solução), os dados são transferidos para o gerente, o qual irá, oportunamente, liberar o início da simulação do instante seguinte, se for o caso. Desta forma o circuito é analisado como se tirássemos uma fotografia de seu funcionamento a cada instante simulado, o que significa que algumas dificuldades, como malhas de realimentação, são tratadas de maneira natural e sem maiores complicações.

Cada MPH (Fig. 2) possui uma unidade lógica e aritmética (ULA), uma memória de escrita eleitura (MEL), uma unidade de controle elementar (UCE), uma unidade de modelos armazenados (UMA) e várias filas de entrada e saída (FES).

A ULA é um *hardware* composto por unidades somadora, multiplicadora e de deslocamento, de alta velocidade. Durante a fase de configuração do arranjo, a ULA é configurada de acordo com um dos modelos de dispositivo disponíveis na UMA. A configuração da ULA só é alterada quando há uma mudança do processo no arranjo, isto é, quando muda o circuito simulado.

A MEL é uma memória de acesso aleatório, com ciclos de leitura e escrita bastante reduzidos. Ela é usada para armazenar os dados de entrada do dispositivo correspondente e os dados de saída (as soluções) da simulação. Durante o mapeamento do circuito, o gerente armazena os dados de cada dispositivo e aqueles relativos às análises, na MEL do MPH correspondente. Durante cada fase de simulação, as MELs ficam dedicadas ao respectivo MPH. Toda a comunicação necessária entre os MPHs é feita através das FES. Ao detetar o fim de uma simulação, o gerente lê os resultados da mesma, na MEL de cada MPH.

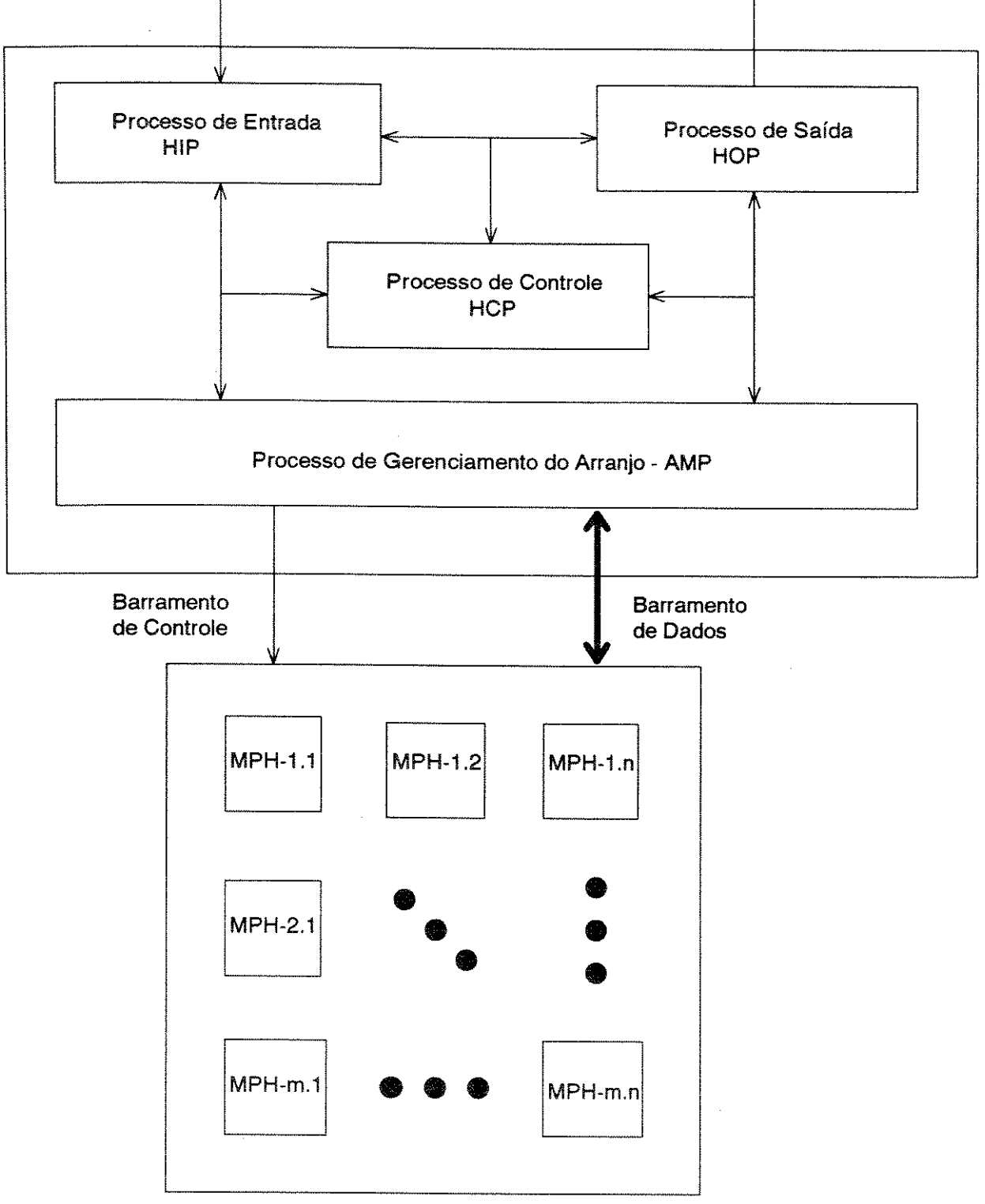

Figura 1 - Arquitetura ABACUS, onde MPH significa *Model Processing Hardware-element*

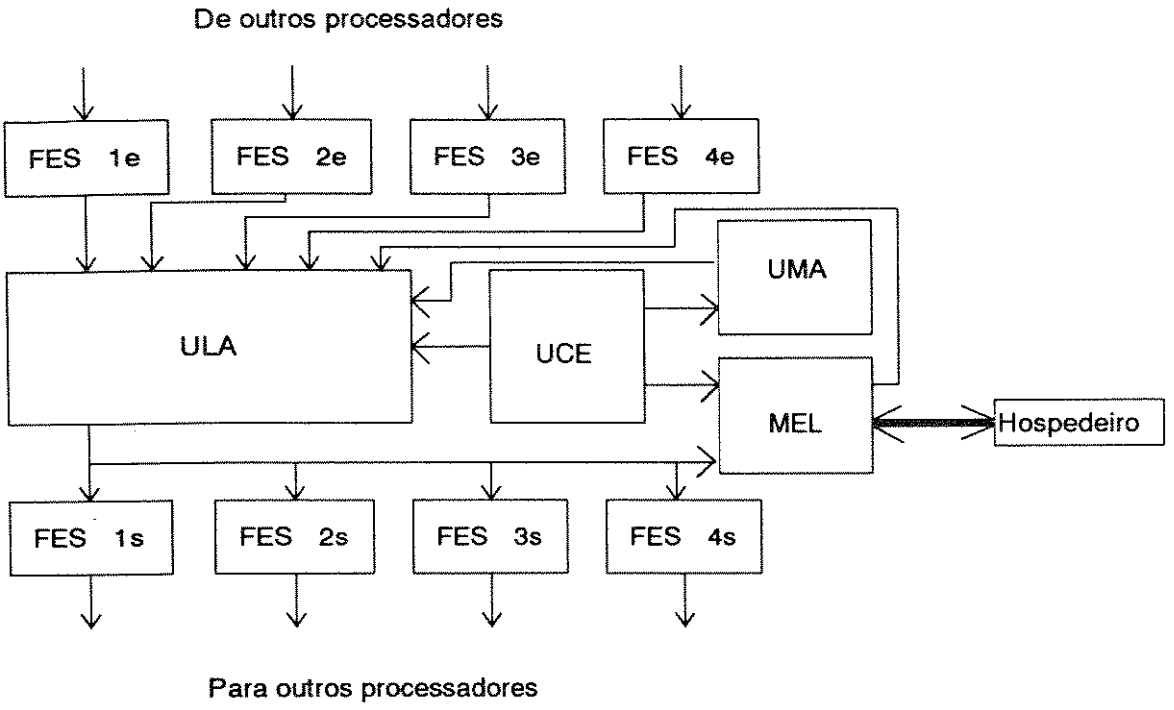

Figura 2 - Arquitetura de um elemento de processamento (MPH)

ULA - Unidade Lógica e Aritmética

UCE - Unidade de Controle Elementar

UMA - Unidade de Modelos Armazenados

MEL - Memória de Escrita e Leitura

FES - Fila de Entrada/Saída (e=entrada, s=saída)

A UCE é uma unidade projetada especialmente para controlar o processamento do MPH e seu interfaceamento, tanto com o hospedeiro quanto com o resto do arranjo. Os três principais fluxos de informação controlados pela UCE são: a sinalização para o gerente do estado de convergência do processador; a transferência das informações do gerente e dos resultados da simulação para o gerente; e a comunicação dos resultados parciais da simulação entre os processadores. Durante a simulação, o controle do comportamento dos MPHs é feito pelas respectivas UCEs.

A UMA é uma memória endereçada pelo conteúdo, onde os diversos modelos conhecidos pela arquitetura ABACUS são armazenados permanentemente. Embora estes modelos não possam ser alterados durante a execução de uma determinada simulação, eles podem ser alterados pelo usuário em modo *offline*. Isto permite que se incluam modelos mais complexos, ou mais simples, e mesmo, que se implementem novos modelos de componentes, no simulador. Os modelos da UMA consistem de microprogramas responsáveis pela configuração da ULA, de acordo com o dispositivo a ser emulado pelo processador.

De maneira geral, pode-se representar o algoritmo de simulação usado por esta metodologia através de cinco processos principais, quatro dos quais rodariam no computador hospedeiro (HCP, HIP, HOP e AMP) e outro que rodaria em cada MPH. Este algoritmo é descrito a seguir:

Processo HCP:

O processo HCP (***Host Control Process***) serve para controlar a operação do sistema de simulação. Este controle deve ser entendido, não como uma sincronização dos demais processos, mas como um gerenciamento assíncrono dos mesmos, ou seja, o HCP inicia e encerra a operação do ambiente de simulação e ativa e desativa cada um dos demais processos do computador hospedeiro, sempre que necessário.

1- Inicia o processo de simulação

2- Inicia o processo HIP

3- Inicia o processo AMP

4- Inicia o processo HOP

5- Termina quando todos os processos estiverem concluidos

Processo HIP:

O processo HIP (*Host Input Process*) tem a finalidade de processar os dados de entrada fornecidos pelo usuário, organizando-os para o tratamento pelos demais processos do simulador.

1-Lê o arquivo de entrada

2-Identifica os elementos do circuito

3-Identifica as análises a serem feitas

4-Monta a base de dados do circuito

5-Monta o grafo de interconexão do circuito

6-Verifica o número de MPHs disponíveis no arranjo

7-Particiona o circuito em subcircuitos (se necessário), de acordo com o número de MPHs disponíveis e mantendo os componentes fortemente conectados no mesmo sub-circuito sempre que possível

8-Prepara cada (sub)circuito para simulação e transfere-o ao processo AMP (avisa o processo AMP a respeito disto)

9-Finaliza o processo HIP e sinaliza o processo HCP

Processo HOP:

O processo HOP (*Host Output Process*) destina-se à função reversa do HIP, ou seja, a partir das soluções geradas pelo simulador e da base de dados organizada pelo processo HIP, HOP fornece as respostas solicitadas pelo usuário.

1-Recebe a base de dados de HIP e formata-a para a saída

2-Recebe os resultados da simulação do processo AMP e formata-os para a saída

3-Quando os processos HIP e AMP sinalizarem o término de funcionamento, transfere os resultados finais formatados para a saída especificada pelo usuário

4-Encerra o processo HOP e avisa o processo HCP do fato

Processo AMP:

O processo AMP (*Array Manager Process*) tem por objetivo controlar o funcionamento do arranjo de processadores. Novamente, o termo controle deve ser entendido aqui, com a conotação dada na descrição do processo HCP, isto é, controla-se a operação dos processadores do arranjo fudamentalmente com base nos dados que cada processador do arranjo fornece ao processo AMP.

1-Verifica o número de MPHs disponíveis no arranjo, armazena-o e informa seu valor quando requisitado

2-Recebe um circuito do processo HIP

3-Mapeia o circuito nos processadores do arranjo

4-Estabelece o estado inicial do arranjo

5-Inicia um contador com o número de elementos do circuito mapeado

6-Incrementa e decrementa o contador, conforme o estado de convergência dos processadores do arranjo, até zerar o contador

7-Busca as soluções no arranjo

8-Transmite as soluções ao processo HOP

9-Repete os passos 2 a 8 até que todo o circuito seja simulado

10-Encerra o processo AMP, avisando os processos HCP e HOP

Processo MPH:

Os processos MPHs *(Model Processing Hardware-element)* são aqueles que vão executar efetivamente a simulação dos componentes do circuito, com base nos dados a eles fornecidos pelo processo AMP.

1-Espera os dados do processo AMP

2-Identifica o elemento a ser simulado

3-Configura a ULA de acordo com o elemento

4-Busca os dados de entrada da memória do hospedeiro

5-Executa os passos a seguir:

5.1-Busca novos dados nas filas de entrada

5.2-Computa o resultado correspondente ou decrementa o contador de AMP

5.3-Salva o resultado parcial na memória local e transfere-os para as filas de saída ou transfere as soluções para o processo AMP

5.4-Volta ao passo 5.1 ou ao passo 1

Cabe aqui abrir um parêntese para discorrer sobre o tratamento das malhas de realimentação e dos nós fortemente acoplados, quando se faz necessário o particionamento do circuito, referido na sétima etapa do processo HIP. Na última década, com a disseminação do uso de processamento paralelo, o particionamento de circuitos VLSI em módulos, tomando-se o cuidado de seccioná-los somente nos nós fracamente acoplados, tornou-se um procedimento corriqueiro. Com isto, mantêm-se os nós fortemente acoplados e as malhas de realimentação dentro de determinados módulos, viabilizando-se a simulação de circuitos com um número elevado de dispositivos, sem prejudicar seu tratamento com relação a esses aspectos[022,033,143,184,189].

Resumindo, em ABACUS, existem vários processadores onde se encontram os modelos que representam o comportamento dos diversos elementos do circuito. Estes processadores são interconectados através de um conjunto de barramentos programáveis, como se interconectássemos o próprio circuito num *protoboard*. A seguir, os dados primários são fornecidos ao arranjo e a simulação é iniciada. A partir do estado inicial, todos os processadores do arranjo começam a trabalhar simultaneamente, buscando dados em suas filas de entrada, processando-os e transferindo os resultados às suas filas de saída. De fato, há um retardo para que os sinais de entrada se propaguem à saída, como num circuito real.

Inicialmente, apenas o primeiro nível de processadores, isto é, aqueles diretamente conectados às entradas primárias do circuito, gerarão sinais diretamente derivados das condições de contorno do circuito. À medida que as iterações são processadas, estes sinais reais propagam-se pelo circuito (arranjo), até alcançarem as saídas. Este processo continua até que todo o arranjo se estabilize para um dado conjunto de sinais de entrada. Neste momento, as soluções são armazenadas, o tempo é incrementado e um novo conjunto de entradas primárias é aplicado ao circuito. O processo descrito neste parágrafo é repetido até que todos os instantes desejados sejam analisados.

Finalmente, é importante salientar que o fluxo de dados através do arranjo de processadores se dá de maneira similar à propagação dos sinais elétricos no circuito real. Esta maneira de encarar a utilização do processamento paralelo para a simulação de circuitos é talvez a grande contribuição deste trabalho, pois permite que busquemos outros limites no tratamento do referido problema. Isto se deve ao fato de encararmos o paralelismo na sua forma mais ampla, ou seja, apenas os instantes inicial e final do funcionamento dos processadores do arranjo são determinados pelo computador hospedeiro, mas durante a execução da simulação, os MPHs funcionam concorrentemente e de forma totalmente assíncrona, tanto em relação ao hospedeiro, quanto entre si.

Método de Validação Utilizado

Você não pode provar uma definição. O que você pode fazer, é mostrar que ela faz sentido.

Albert Einstein[113]

1. Introdução:

Intuitivamente é possível afirmar que a metodologia aqui proposta funciona; entretanto, é necessário demonstrar sua viabilidade. Esta demonstração será, aqui, tratada por validação da metodologia[085].

Buscou-se utilizar ferramentas que se adequassem ao presente problema. Uma opção seria um simulador funcional utilizando-se de uma linguagem de descrição de *hardware* (HDL)[054,097,155,169,188]; todavia, este tipo de ferramenta ainda não se encontrava disponível.

Verificamos a possibilidade de uso de redes de Petri[092], que têm sido empregadas no modelamento de sistemas concorrentes. Um modelo da arquitetura ABACUS foi desenvolvido e analisado em redes de Petri, tendo os resultados desta análise indicado a necessidade de estendê-los (modelo e análise) para melhor caracterizar a metodologia proposta. Entretanto, as ferramentas necessárias para tal estavam em fase inicial de seu desenvolvimento e sua utilização, ainda neste trabalho, foi inviável.

Conseqüentemente, decidiu-se realizar um programa de computador que fosse capaz de representar as características fundamentais da metodologia ABACUS, indicando, como resultados, dados para a aferição de suas propriedades de convergência e para a análise de sua complexidade. Este programa foi desenvolvido em linguagem C[125,126,166,172,173], utilizando o compilador Turbo C++ 2.1 da Borland, rodando num microcomputador NEC - PowerMate 386/25, com sistema operacional DOS 3.3.

Neste capítulo são apresentados, inicialmente, o modelo desenvolvido em rede de Petri e a análise de seus resultados, e, logo após, o programa de computador mencionado no parágrafo anterior.

2. Simulação em rede de Petri:

Com base no algoritmo descrito no capítulo anterior, foi desenvolvido um modelo em redes de Petri. Estas redes têm sido usadas no modelamento de sistemas tanto de *hardware* quanto de ***software***, que exibam concorrência[092,176]. O modelo da arquitetura ABACUS que foi desenvolvido, foi analisado com o programa SIPRO[122].

Redes de Petri constituem uma ferramenta matemática usada para a representação formal de sistemas de eventos discretos[034,044,140,152]. Esta ferramenta tem sido largamente utilizada para a descrição de sistemas concorrentes devido à sua simplicidade funcional e às suas características gráficas, as quais permitem uma visualização fácil dos sistemas modelados.

A descrição informal da arquitetura ABACUS, apresentada no capítulo anterior, foi convertida numa descrição formal em rede de Petri, tomando-se muito cuidado em manter as características da arquitetura original, evitando ambigüidades.

A figura 1 mostra o esquema do modelo da rede de Petri construído, onde os círculos indicados com Lxx indicam os lugares da rede de Petri; as barras marcadas com Bxx indicam as transições da rede de Petri; e as linhas tracejadas servem para relacionar cada parte da rede de Petri com o correspondente processo de ABACUS. Os pontos nos lugares L1 e L20 na figura em questão são fichas, as quais, juntas, constituem a marcação inicial do modelo. Na tabela 1 mostra-se a relação entre cada uma das transições do modelo desenvolvido e o respectivo processo em ABACUS.

O programa SIPRO examina modelos em redes de Petri quanto às suas propriedades de limitabilidade, vivacidade e reiniciabilidade (explicadas abaixo). Ele também gera diagnósticos a serem usados para a correção de falhas no modelo, tais como transições que não disparam.

O modelo de ABACUS foi examinado pelo programa SIPRO, como dissemos anteriormente, tendo sido obtidas 125 marcações. O modelo foi diagnosticado como vivo, reiniciável e limitado.

Uma rede de Petri com um número finito de marcações é dita limitada. A condição de limitabilidade que obtivemos para o modelo significa que a arquitetura ABACUS está livre de contenções nos barramentos e que o sistema de memória está adequadamente dimensionado.

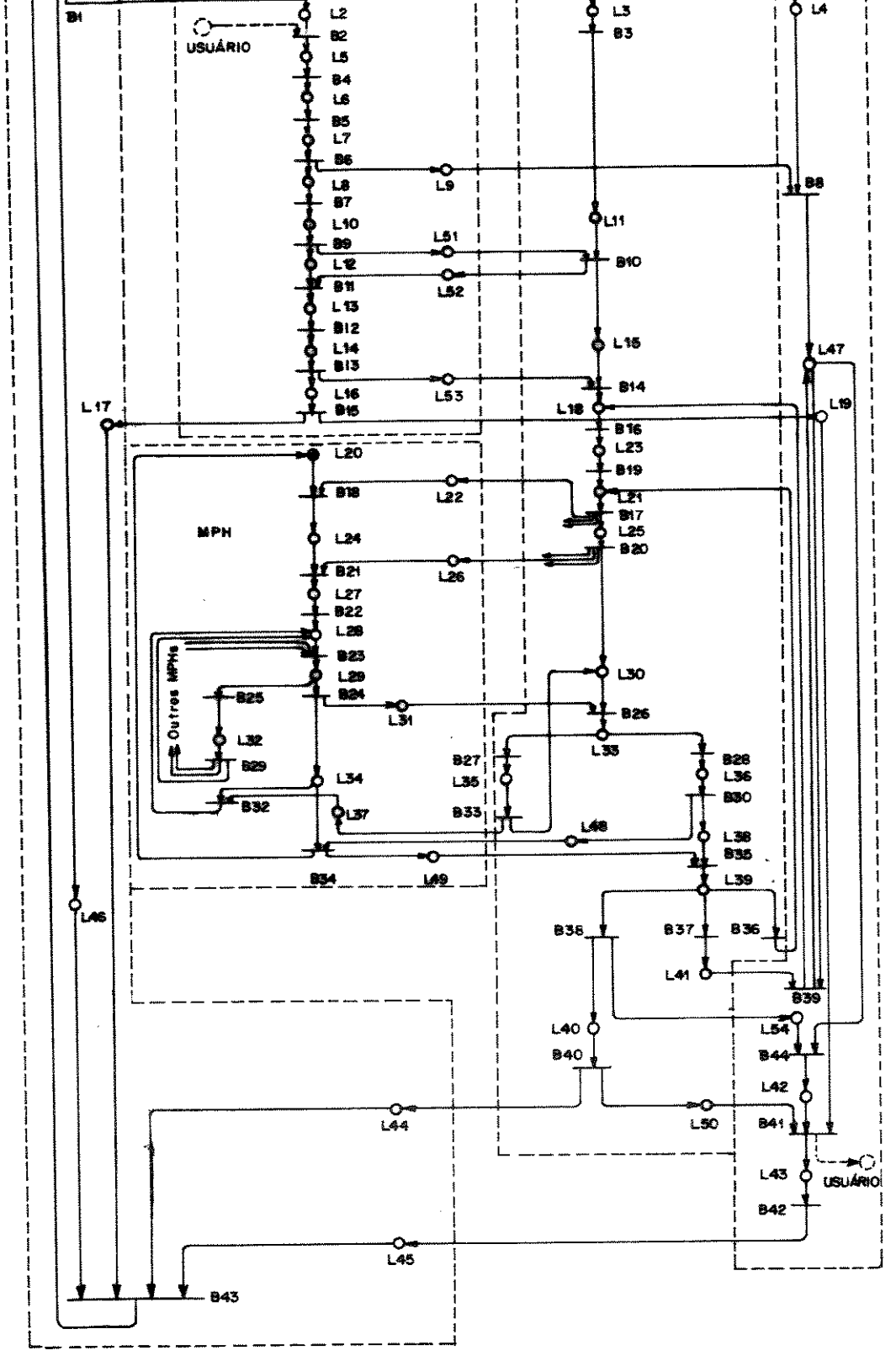

Figura 1 - Modelo da arquitetura ABACUS em rede de Petri

Tabela 1 - Relações entre as transições da figura 1 e os processos de ABACUS

B1	HCP inicia a simulação
B2	HIP lê a entrada
B3	AMP verifica o nº de MPHs disponíveis no arranjo
B4	HIP identifica os elementos do circuito
B5	HIP identifica os tipos de análises a serem feitas
B6	HIP monta uma base de dados
B7	HIP monta um grafo de conexão
B8	HOP recebe uma base de dados de HIP
B9	HIP solicita o nº de MPHs disponíveis para AMP
B10	AMP envia o nº de MPHs disponíveis para HIP
B11	HIP recebe o nº de MPHs disponíveis de AMP
B12	HIP particiona o circuito de acordo com o nº de MPHs
B13	HIP envia as partições do circuito para AMP
B14	AMP recebe as partições do circuito de HIP
B15	HIP conclui o processamento de entrada
B16	AMP mapeia uma partição no arranjo e inicia as variáveis de simulação
B17	AMP envia informações para cada MPH
B18	MPH recebe informações de AMP
B19	AMP programa o sistema de barramentos para interconectar os MPHs
B20	AMP sinaliza o início da simulação para os MPHs
B21	MPH recebe o sinal de inicialização de AMP
B22	MPH identifica o elemento a ser simulado e configura a ULA

B23	MPH recebe resultados dos vizinhos e compara com os seus
B24	MPH convergiu e envia *flag* de convergência para AMP
B25	MPH não convergiu
B26	AMP recebe um *flag* de convergência dum MPH
B27	AMP verifica a não convergência do sistema
B28	AMP verifica a convergência do sistema
B29	MPH processa seus dados e envia seus novos resultados para os MPHs vizinhos
B30	AMP para a simulação no arranjo
B31	Transição inexistente
B32	MPH recebe sinal de reinício de AMP e começa nova iteração
B33	AMP volta a esperar outro *flag* do arranjo
B34	MPH recebe sinal de parada de AMP e envia-lhe os resultados
B35	AMP recebe os resultados do arranjo
B36	AMP atualiza as variáveis de simulação e inicia nova fase
B37	AMP envia os resultados da simulação para HOP e inicia a simulação doutra partição
B38	AMP envia os resultados da simulação da última partição para HOP
B39	HOP recebe os resultados de AMP, formata-os e envia um sinal de reinício para AMP
B40	AMP conclui a simulação no arranjo
B41	HOP envia os resultados finais, formatados, para o usuário
B42	HOP conclui o processamento de saída
B43	HCP finaliza a simulação
B44	HOP recebe os resultados de AMP e formata-os para a saída

Redes de Petri vivas são aquelas nas quais todas as transições são disparadas, pelo menos uma vez. A confirmação da propriedade de vivacidade garante que a rede está livre

de bloqueios e que todas as suas partes são acessíveis, e, conseqüentemente, indica que o sistema por ela modelado deve partilhar as mesmas características.

A propriedade de reiniciabilidade de uma rede de Petri, representa a capacidade do sistema modelado de retornar para o seu estado inicial, após executar um certo número de tarefas e indica que o sistema é, provavelmente, convergente.

Do desenvolvimento e da análise do modelo em rede de Petri recém descrito, infere-se que o tipo de rede de Petri usada não é adequado para representar o paralelismo existente na arquitetura ABACUS, em sua totalidade. Para tanto, devem-se utilizar redes de Petri numéricas[015], para representar completamente o seu paralelismo, e redes de Petri temporizadas[121], para a análise do desempenho. Todavia, estas ferramentas não se encontravam disponíveis, durante a execução deste trabalho; então optou-se por protelar esta abordagem, em favor do desenvolvimento de um programa que pudesse simular a metodologia proposta. Este programa é assunto da seção seguinte.

3. Simulação usando um programa em C:

Foi desenvolvido um programa para simular a metodologia ABACUS. Este programa é constituído de quatro módulos fundamentais (Fig. 2), quais sejam: entrada, controle, execução e saída. Os módulos de entrada e saída são bastante rudimentares, desempenhando apenas as funções necessárias para que seja possível conversar com o programa. Os dados de entrada são apresentados na forma descrita no apêndice I e os dados de saída são colocados num arquivo em forma tabular, conforme pode ser visto nas figuras 1.a e 2.a do capítulo 3. O módulo de controle contém as rotinas que fazem a parte de pré- e pós- processamento do arranjo, além de controlar o estado global deste. O módulo de execução faz as vezes do arranjo de MPHs propriamente dito, ou seja, é o módulo onde se encontram as rotinas responsáveis pela execução da simulação.

A propósito, a simultaneidade na execução do arranjo de processadores é considerada através de uma tabela de eventos[039,045,046,050,057,144,157]. Esta tabela é composta por uma série de pares ordenados do tipo (evento, tempo), onde evento representa cada uma das instâncias dos processadores para o circuito tratado e tempo é o instante de tempo, interno ao simulador, no qual o respectivo evento deve ser ativado. Sempre que o simulador precisar ativar um dos processadores do arranjo, ele procura aquele evento ao qual corresponde o menor tempo, ativando-o. Caso haja mais de um evento com o menor tempo, o simulador

ativa o primeiro que encontrar. Isto porque, os eventos estão organizados, na tabela, em ordem crescente de nível, isto é, das entradas primárias para as saídas do circuito. Portanto, em caso de empate no tempo, primeiro é feita a simulação do elemento que estiver mais próximo dos sinais de entrada e depois a do elemento mais afastado destes sinais, propiciando uma resposta real mais rapidamente. Após a execução de um determinado evento, a tabela de eventos é atualizada, pelo acréscimo do tempo relativo (correspondente ao processamento do elemento recém simulado) ao valor do respectivo campo tempo na tabela de eventos. Os tempos básicos de processamento de cada um dos elementos disponíveis no simulador permanecem armazenados noutra tabela, que é consultada para a atualização da tabela de eventos.

O programa, denominado ABACUS, é composto por 4172 linhas, divididas em 51 rotinas, as quais, por sua vez, podem ser agrupadas nos quatro módulos fundamentais descritos no primeiro parágrafo desta seção.

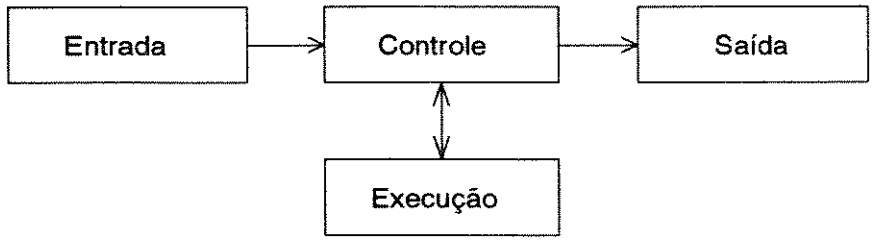

Figura 2 - Diagrama do programa ABACUS

No módulo de entrada agrupam-se seis rotinas, as quais executam as funções de: informar ao usuário o formato necessário de entrada; ler o respectivo arquivo, nele fazendo a verificação de alguns aspectos básicos do circuito (uma detecção de erros bastante limitada); e dar ao usuário a opção de ver os valores *default* de algumas variáveis, conferir o circuito lido e continuar ou não a simulação. Este módulo engloba as características do processo HIP da arquitetura ABACUS.

O módulo de controle possui quatorze rotinas, visando desempenhar as funções dos processos HCP e AMP da arquitetura ABACUS. As funções referidas são: a organização dos dados de entrada e respectiva carga no arranjo; a ativação do arranjo e o controle da convergência global do mesmo; o recolhimento das soluções geradas pelo arranjo em cada

instante simulado, além de iniciar e concluir as operações de todo o simulador.

Trinta rotinas constituem o módulo execução, que representa o processo MPH da arquitetura ABACUS; destas, 22 executam as funções referentes ao processamento dos 13 possíveis modelos de elementos de circuito implementados e as demais servem para representar o processamento simultâneo dos vários processadores do arranjo.

No módulo de saída encontra-se uma única rotina, responsável pela aquisição dos dados de entrada e das soluções do simulador, formatando-os adequadamente e armazenando-os num arquivo de saída. Este módulo representa a operação básica do processo HOP da arquitetura ABACUS.

O algoritmo, empregado na solução do sistema, baseia-se no método numérico da bisseção[017,181], e consiste, fundamentalmente, no seguinte: cada processador recebe os valores de tensão e corrente calculados pelos processadores a ele conectados e calcula a média aritmética das tensões (Va e Vb) em cada um dos terminais (a e b) do elemento de circuito a ele confiado e a respectiva diferença de potencial (Vab1), bem como a somatória dos valores algébricos das correntes (I) que a ele confluem. Com o valor da somatória das correntes, calcula uma segunda diferença de potencial (Vab2) sobre o elemento. Estabelece como nova tensão (Vab) sobre o elemento, o valor médio das duas diferenças de potencial recém mencionadas (Vab1 e Vab2). Calcula a corrente (Iab) correspondente à tensão estabelecida (Vab), de acordo com o modelo do elemento em questão (F). Tomando a voltagem num dos terminais como referência (Va), atribui ao outro (Vb) a soma da voltagem deste (Va) com a tensão estabelecida (Vab) para o elemento. Finalmente, informa os novos valores de corrente e voltagem aos processadores vizinhos. Ao final de cada iteração, verifica se os novos valores obtidos estão em conformidade com a expectativa de erro, declarando seu estado de convergência ao processo gerente.

Um diagrama de fluxo do algoritmo descrito no parágrafo anterior pode ser visto na figura 3 e os principais modelos utilizados no programa na tabela 2.

No programa ABACUS foram inseridos contadores, para se verificar o volume de comunicação entre os processadores e entre estes e o computador hospedeiro (um quesito[089,139] muito significativo quando se trata de arquiteturas paralelas) e para medir o número de iterações que cada processador executa, até que o arranjo chegue a uma convergência global.

Tempos relativos foram atribuídos a cada modelo, levando-se em conta o número e os tipos de operações neles existentes, bem como o número de ciclos de cada operação. Para vincular tais valores com medidas reais, utilizaram-se os dados do transputer T800-20[083], uma

vez que a arquitetura deste processador é semelhante àquela que se tinha em mente desenvolver e por operar numa freqüência compatível com a maioria dos micros disponíveis à época. Com base nisto e no número de iterações medidas, foram reconstituídos os tempos de processamento utilizados na análise de desempenho apresentada no capítulo seguinte. Os tempos e ciclos mencionados neste parágrafo estão mostrados no Apêndice II e a listagem do programa no Apêndice III.

Tabela 2 - Principais modelos (F, na fig.3) utilizados no programa ABACUS

Resistor	$V=RI$
Capacitor	$I=I_0\, e^{-\frac{t-\Delta t}{\tau}}$ $V=V_f[1-e^{-\frac{t-\Delta t}{\tau}}]+V_{carga}\, e^{-\frac{t-\Delta t}{\tau}}$ $\tau=\frac{V_f C}{I_0}$
Diodo	$V<0 \rightarrow I=-I_{sat}$ $V\geq 0 \rightarrow I=\sum I_{entrada} \wedge V=V_t I_n(\frac{I}{I_{sat}}+1)$
Fontes	$V=V_{max}\cos(2\pi f t)$ $I=I_{max}\cos(2\pi f t)$

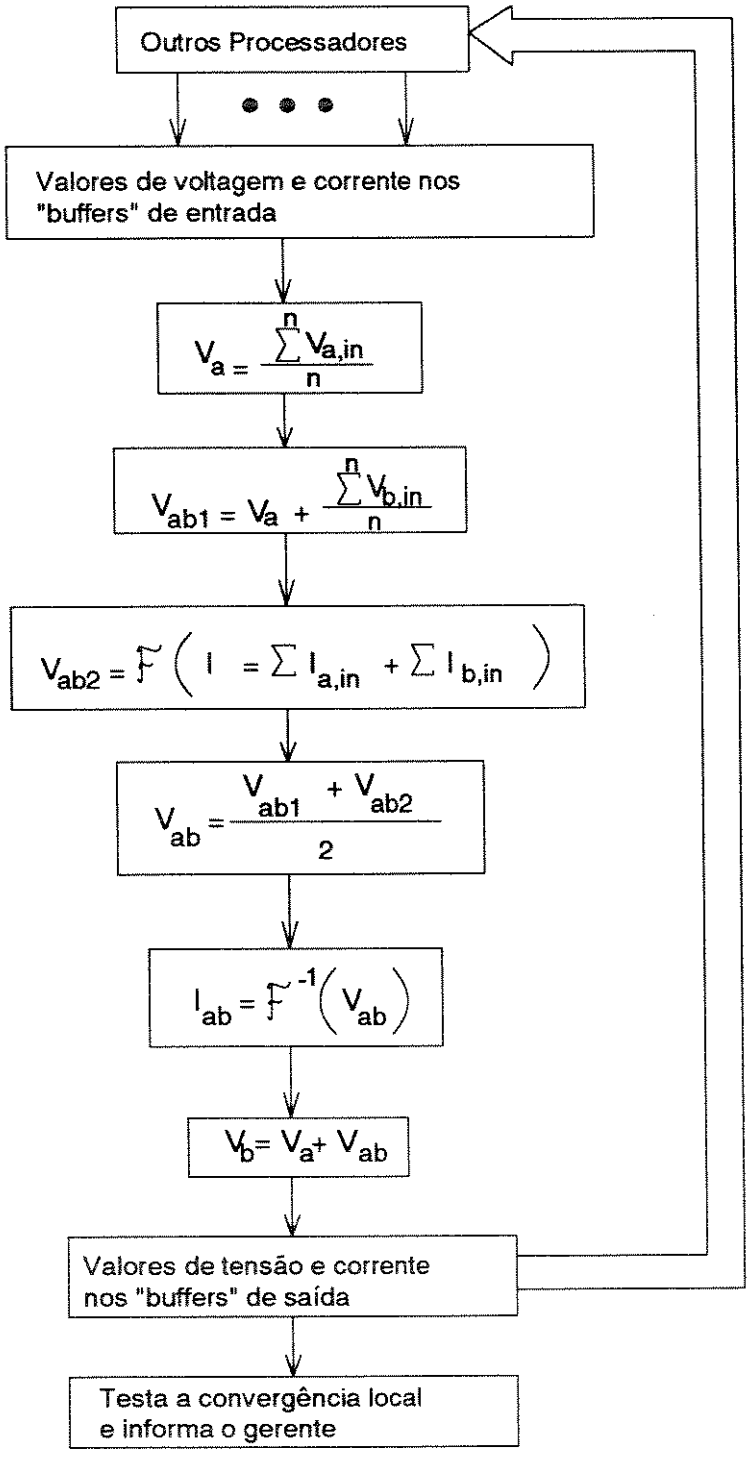

Figura 3 - Diagrama de fluxo do algoritmo de solução implementado no programa ABACUS.

Especificação da arquitetura ABACUS

Na verdade você nunca entende uma
nova teoria. Você simplesmente a utiliza.
Albert Einstein[113]

1. Introdução:

O programa ABACUS, descrito no capítulo precedente, foi desenvolvido para se avaliar características da metodologia proposta, tais como: o volume de informações que circulam na arquitetura e a capacidade de convergência da metodologia em questão. Foram simulados diversos circuitos, com associações série, paralela e mista de elementos lineares e não lineares, representativos dos problemas topológicos que usualmente se desejam simular.

Neste capítulo, são discutidos os resultados das simulações realizadas com o programa ABACUS e, a partir destas, é feita a especificação de uma arquitetura possível para a implementação da metodologia proposta no capítulo I.

2. Discussão dos resultados das simulações:

Um subconjunto dos circuitos simulados foi considerado representativo das características de comunicação da arquitetura ABACUS. Foram simulados componentes lineares, não lineares e junções. Como elementos pilotos foram escolhidos, respectivamente, resistores, capacitores e diodos. A escolha destes dispositivos deu-se por representarem adequadamente cada um dos tipos de componentes desejados e seus modelos serem suficientemente simples para minimizar a possibilidade de erros devido a fatores não relacionados, especificamente, com a metodologia que se queria analisar. Na tabela 1 estão sintetizados os dados referentes ao volume de comunicação obtidos das simulações feitas com este subconjunto. Como exemplo, os resultados das simulações de um filtro RC passa-altas e de um retificador de meia onda, podem ser vistos nas figuras 1 e 2.

Da tabela 1, depreende-se que o número médio (foi utilizada a média aritmética

ponderada em relação ao número de circuitos analisados em cada caso) de ítens transferidos entre o gerente e os processadores do arranjo, por instante analisado, é de 31,60755 e que o número médio de processadores utilizados nestes testes foi de 7,54545. Com estes dados chega-se a uma média de 4,2 ítens/instante/processador. Além disso, obteve-se uma média de 10,6471 iterações, nos diversos testes realizados.

Admita-se, conforme é descrito abaixo, que a cada instante analisado, cada processador, troca três ítens de informação com o gerente. Multipliquem-se os restantes 1,2 ítens/instante/processador pelo número médio de iterações realizadas. O resultado obtido é de 12,7765, ou seja, aproximadamente 13 ítens de informação por processador. Estes 13 ítens são transferidos uma única vez a cada simulação e referem-se à transferência dos dados iniciais, bem como das condições de contorno, do gerente para os processadores do arranjo.

Sinteticamente, a comunicação entre o hospedeiro e os processadores se dá segundo o esquema abaixo.

No início do processamento, acontecem as seguintes transferências do hospedeiro para cada processador do arranjo:

15 ítens informando o componente a ser simulado, seu valor, (exceto para as fontes pulsadas) e inicializando os *buffers* e a memória;

1 item, se for diodo, informando a corrente de saturação;

1 item, se for fonte contínua ou alternada, informando a freqüência;

1 item, se for fonte quadrada, informando o período;

1 item, se for fonte controlada por corrente, informando o elemento cuja corrente controla a fonte;

2 ítens, se for fonte controlada por tensão, informando os nós cuja diferença de potencial controla a fonte;

9 ítens, se for fonte linear por partes, informando seus pontos de inflexão;

N ítens, onde N é determinado em função do número de ligações de cada processador, indicando com quem cada processador está conectado e através de que canal se dá esta conexão (esta operação equivale à interconexão do arranjo através da programação do conjunto de blocos de chaves dos barramentos, conforme será visto mais adiante).

Durante o processamento, ocorrem as seguintes transferências de dados e sinais de controle:

a) Do hospedeiro para os processadores do arranjo, ao final de cada instante simulado, é enviado um sinal indicando a necessidade de transferência das soluções para os "buffers" de

comunicação com o hospedeiro e outro, reiniciando os "flags" de convergência e indicando a liberação dos processadores do arranjo para o início do tratamento de um novo instante;

b) De cada um dos processadores para o hospedeiro, ao final de cada iteração, é enviado um sinal indicando o estado de seus *flags* de convergência e, ao final de cada instante simulado, são enviados quatro ítens, transferindo os valores de tensão e corrente, correspondentes às soluções obtidas;

c) Entre os diversos processadores do arranjo, que estejam diretamente conectados entre si, ao final de cada iteração, são informandos os valores de tensão e corrente, obtidos para o terminal correspondente ao canal ao qual estão ligados.

Tabela 1 - Volume de comunicação de dados na arquitetura ABACUS

Itens transferidos do gerente (G) para o processador (P) por instante analisado		
Número de Processadores	Número médio de itens transferidos de G para P	Número médio de itens transferidos de P para G
2	16,8074	16,3181
3	17,2123	17,2301
4	34,0942	31,8676
5	22,8458	19,7916
6	25,8778	23,5611
7	14,0476	11,1786
8	50,7965	49,6987
9	52,0034	47,6027
10	47,1250	44,9500
13	27,3205	25,0897
16	27,3437	25,0000
Média	30,4977	32,7174

```
****** Arquivo de entrada  ABACUS.IN ******

ftq       10        0.01      0         1
cap       1e-06     2         1
res       1000      0         2
0
0.01
20
1e-06
27

***** Resultados da Simulacao *****

          processador (tipo)  corrente          tensao no polo 1    tensao no polo 0
t: 0:
          mph[0] (4):  ftq    0.00999998        10                  0
          mph[1] (1):  cap    0.00999998        10                  10
          mph[2] (0):  res    0.00999999        9.99999             0
t: 0.0005:
          mph[0] (4):  ftq    0.0060653         10                  0
          mph[1] (1):  cap    0.0060653         10                  6.06531
          mph[2] (0):  res    0.00606531        6.06531             0
t: 0.001:
          mph[0] (4):  ftq    0.00367879        10                  0
          mph[1] (1):  cap    0.00367879        10                  3.6788
          mph[2] (0):  res    0.0036788         3.6788              0
t: 0.0015:
          mph[0] (4):  ftq    0.0022313         10                  0
          mph[1] (1):  cap    0.0022313         10                  2.23131
          mph[2] (0):  res    0.00223131        2.23131             0
t: 0.002:
          mph[0] (4):  ftq    0.00135336        10                  0
          mph[1] (1):  cap    0.00135336        10                  1.35336
          mph[2] (0):  res    0.00135336        1.35336             0
t: 0.0025:
          mph[0] (4):  ftq    0.000820853       10                  0
          mph[1] (1):  cap    0.000820853       10                  0.820855
          mph[2] (0):  res    0.000820854       0.820854            0
t: 0.003:
          mph[0] (4):  ftq    0.000497873       10                  0
          mph[1] (1):  cap    0.000497873       10                  0.497874
          mph[2] (0):  res    0.000497874       0.497874            0
t: 0.0035:
          mph[0] (4):  ftq    0.000301975       10                  0
          mph[1] (1):  cap    0.000301975       10                  0.301976
          mph[2] (0):  res    0.000301976       0.301976            0
t: 0.004:
          mph[0] (4):  ftq    0.000183158       10                  0
          mph[1] (1):  cap    0.000183158       10                  0.183158
          mph[2] (0):  res    0.000183158       0.183158            0
t: 0.0045:
          mph[0] (4):  ftq    0.000111091       10                  0
          mph[1] (1):  cap    0.000111091       10                  0.111091
          mph[2] (0):  res    0.000111091       0.111091            0
```

Figura 1-a - Listagem da simulação do filtro passa-altas da fig.1-b

(continua na página seguinte)

```
t: 0.005:
        mph[0] (4):  ftq   -0.0198889      -10            0
        mph[1] (1):  cap   -0.0198889      -10            -19.8889
        mph[2] (0):  res   -0.0198889      -19.8889       0
t: 0.0055:
        mph[0] (4):  ftq   -0.0120632      -10            0
        mph[1] (1):  cap   -0.0120632      -10            -12.0632
        mph[2] (0):  res   -0.0120632      -12.0632       0
t: 0.006:
        mph[0] (4):  ftq   -0.00731672     -10            0
        mph[1] (1):  cap   -0.00731672     -10            -7.31674
        mph[2] (0):  res   -0.00731673     -7.31673       0
t: 0.0065:
        mph[0] (4):  ftq   -0.00443782     -10            0
        mph[1] (1):  cap   -0.00443782     -10            -4.43783
        mph[2] (0):  res   -0.00443782     -4.43782       0
t: 0.007:
        mph[0] (4):  ftq   -0.00269168     -10            0
        mph[1] (1):  cap   -0.00269168     -10            -2.69168
        mph[2] (0):  res   -0.00269168     -2.69168       0
t: 0.0075:
        mph[0] (4):  ftq   -0.00163259     -10            0
        mph[1] (1):  cap   -0.00163259     -10            -1.63259
        mph[2] (0):  res   -0.00163259     -1.63259       0
t: 0.008:
        mph[0] (4):  ftq   -0.000990215    -10            0
        mph[1] (1):  cap   -0.000990215    -10            -0.990217
        mph[2] (0):  res   -0.000990216    -0.990216      0
t: 0.0085:
        mph[0] (4):  ftq   -0.000600596    -10            0
        mph[1] (1):  cap   -0.000600596    -10            -0.600597
        mph[2] (0):  res   -0.000600597    -0.600597      0
t: 0.009:
        mph[0] (4):  ftq   -0.00036428     -10            0
        mph[1] (1):  cap   -0.00036428     -10            -0.364282
        mph[2] (0):  res   -0.000364281    -0.364281      0
t: 0.0095:
        mph[0] (4):  ftq   -0.000220947    -10            0
        mph[1] (1):  cap   -0.000220947    -10            -0.220947
        mph[2] (0):  res   -0.000220947    -0.220947      0
t: 0.01:
        mph[0] (4):  ftq   0.019779        10             0
        mph[1] (1):  cap   0.019779        10             19.7791
        mph[2] (0):  res   0.019779        19.779         0
```

Figura 1-a - Listagem da simulação do filtro passa-altas da fig.1-b

(continuação da página anterior)

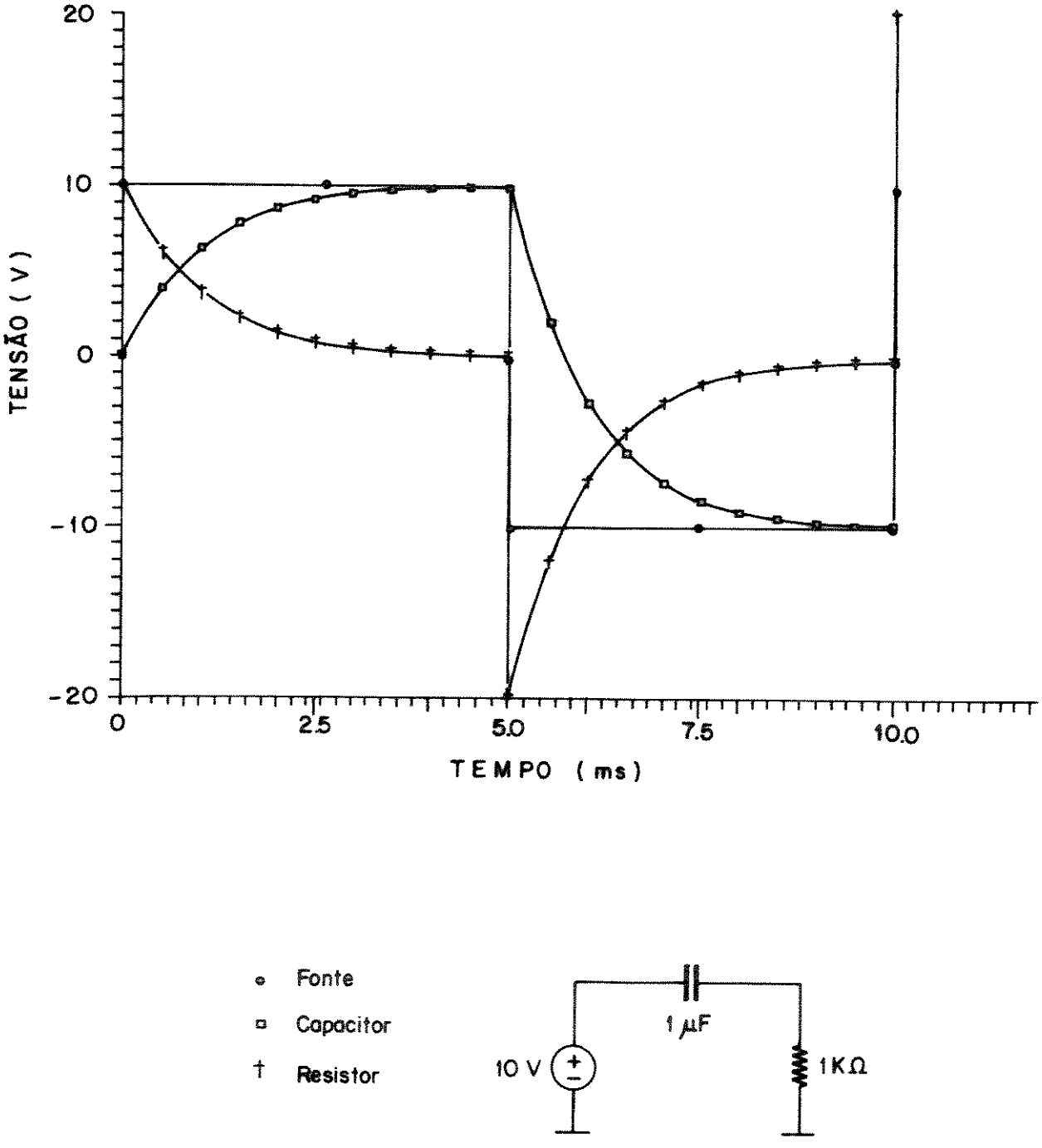

Figura 1-b - Resultado da simulação de um filtro RC passa-altas
Curvas de tensão por tempo - R=1KΩ; C=1μF; fonte=10V

```
***** Arquivo de entrada  ABACUS.IN ******

fti     10      1000    0       1
dio     260     1e-14   2       1
res     1000    0       2
0
0.001
16
1e-06
27

***** Resultados da Simulacao *****

        processador (tipo)   corrente              tensao no polo 1   tensao no polo 0
t: 0:
        mph[0] (3):  fti     0.00928761            10                 0
        mph[1] (2):  dio     0.00928762            10                 9.28764
        mph[2] (0):  res     0.00928762            9.28762            0
t: 6.25e-05:
        mph[0] (3):  fti     0.0085286             9.2388             0
        mph[1] (2):  dio     0.00852861            9.2388             8.52864
        mph[2] (0):  res     0.00852862            8.52862            0
t: 0.000125:
        mph[0] (3):  fti     0.00636844            7.07107            0
        mph[1] (2):  dio     0.00636844            7.07107            6.36846
        mph[2] (0):  res     0.00636845            6.36845            0
t: 0.0001875:
        mph[0] (3):  fti     0.00314247            3.82683            0
        mph[1] (2):  dio     0.00314248            3.82683            3.14248
        mph[2] (0):  res     0.00314248            3.14248            0
t: 0.00025:
        mph[0] (3):  fti     0                     0                  0
        mph[1] (2):  dio     0                     0                  0
        mph[2] (0):  res     0                     0                  0
t: 0.0003125:
        mph[0] (3):  fti     0                     -3.82684           0
        mph[1] (2):  dio     0                     -3.82684           0
        mph[2] (0):  res     0                     0                  0
t: 0.000375:
        mph[0] (3):  fti     0                     -7.07107           0
        mph[1] (2):  dio     0                     -7.07107           0
        mph[2] (0):  res     0                     0                  0
t: 0.0004375:
        mph[0] (3):  fti     0                     -9.2388            0
        mph[1] (2):  dio     0                     -9.2388            0
        mph[2] (0):  res     0                     0                  0
t: 0.0005:
        mph[0] (3):  fti     0                     -10                0
        mph[1] (2):  dio     0                     -10                0
        mph[2] (0):  res     0                     0                  0
```

Figura 2-a - Listagem da simulação do retificador da fig.2-b

(continua na página seguinte)

```
t: 0.0005625:
        mph[0] (3):  fti  0            -9.23879      0
        mph[1] (2):  dio  0            -9.23879      0
        mph[2] (0):  res  0            0             0
t: 0.000625:
        mph[0] (3):  fti  0            -7.07107      0
        mph[1] (2):  dio  0            -7.07107      0
        mph[2] (0):  res  0            0             0
t: 0.0006875:
        mph[0] (3):  fti  0            -3.82683      0
        mph[1] (2):  dio  0            -3.82683      0
        mph[2] (0):  res  0            0             0
t: 0.00075:
        mph[0] (3):  fti  0            0             0
        mph[1] (2):  dio  0            0             0
        mph[2] (0):  res  0            0             0
t: 0.0008125:
        mph[0] (3):  fti  0.00314248   3.82684       0
        mph[1] (2):  dio  0.00314248   3.82684       3.14249
        mph[2] (0):  res  0.00314248   3.14248       0
t: 0.000875:
        mph[0] (3):  fti  0.00636844   7.07107       0
        mph[1] (2):  dio  0.00636845   7.07107       6.36846
        mph[2] (0):  res  0.00636845   6.36845       0
t: 0.0009375:
        mph[0] (3):  fti  0.00852861   9.2388        0
        mph[1] (2):  dio  0.00852862   9.2388        8.52864
        mph[2] (0):  res  0.00852862   8.52862       0
t: 0.001:
        mph[0] (3):  fti  0.00928761   10            0
        mph[1] (2):  dio  0.00928761   10            9.28764
        mph[2] (0):  res  0.00928762   9.28762       0
```

Figura 2-a - Listagem da simulação do retificador da fig.2-b
(continuação da página anterior)

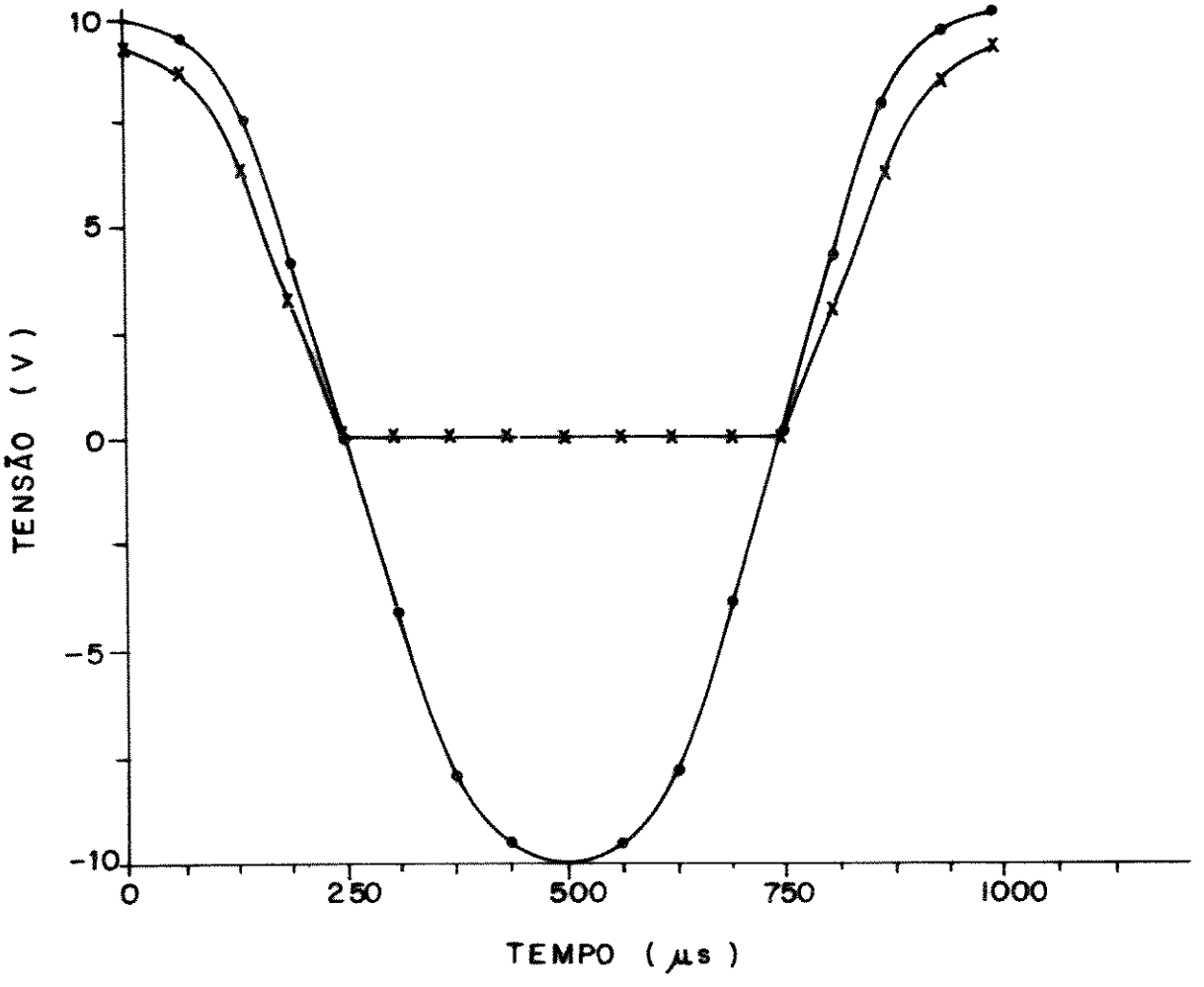

Figura 2-b - Resultado da simulação de um retificador de meia onda
Curvas de tensão por tempo - R=1KΩ; fonte=10V; $ID=I_{sat}[e^{-(qv/kt)}-1]$

Do subconjunto, mencionado no início desta seção, um grupo de circuitos foi escolhido para a comparação da velocidade prevista para o ABACUS com a da medida no programa PSPICE[011] (o PSPICE é uma versão do SPICE2.G-6, fabricado pela MicroSim, para uso em microcomputadores). Dos circuitos escolhidos, a metade é constituída somente por elementos lineares (resistores) e os demais somente por elementos não lineares e junções (como os diodos satisfazem a ambas as características, foram escolhidos estes elementos). Os dispositivos foram associados de diversas formas e conjuntos idênticos de circuitos foram simulados nos programas ABACUS e PSPICE. O PSPICE foi utilizado num computador com freqüência de relógio de 20MHz; a mesma freqüência foi utilizada no cálculo dos tempos gastos pelo programa ABACUS.

O processo utilizado na estimativa dos tempos de processamento do programa ABACUS foi a multiplicação do número de iterações de cada processador pelo número de ciclos correspondentes ao modelo do elemento a ele atribuído e pelo período de cada ciclo (vide apêndice II). Ao resultado desta multiplicação foi adicionado o produto do número de informações trocadas em cada instante da simulação pelo número de ciclos gastos para transferir cada informação e pelo período de cada ciclo. Como os processadores do arranjo operam em paralelo, foi tomado o tempo gasto pelo processador mais lento, para computar as soluções da simulação desejada .

Como tempo gasto pelo PSPICE, foram computados os tempos utilizados pelas rotinas matemáticas, obtidos por meio da opção .ACCT existente no referido programa, a qual fornece uma listagem informando o tempo gasto com cada uma das rotinas do programa durante a simulação.

Os resultados destas medidas podem ser vistos nas tabelas 2 e 3, respectivamente, para o ABACUS e o PSPICE. Para a compreensão daquelas tabelas e do texto a seguir, define-se aqui o número de níveis de um circuito como sendo o número de elementos no maior caminho serial entre uma fonte primária e um terra do circuito. Naquelas tabelas pode-se observar que a metodologia ABACUS é por volta de 1000 vezes mais rápida que a convencional, utilizada no PSPICE.

O custo computacional da metodologia empregada no programa PSPICE é composto, principalmente, pelo tempo gasto na formação das matrizes de solução do sistema de equações que representam o circuito a ser simulado e pelo tempo gasto na solução efetiva do sitema (Ax=b). O tempo gasto com a montagem das matrizes cresce linearmente com o número de elementos do circuito; e o tempo gasto na solução do sistema Ax=b, tem um fator de crescimento proporcional ao número de elementos, elevado à potência β onde $1,1 < \beta < 1,5$

e β depende da diferença entre o tempo necessário para fazer as operações matemáticas e a largura de faixa da memória[006].

Dos resultados mostrados na tabela 2, verifica-se que o tempo de processamento da metodologia ABACUS é independente do número de elementos do circuito. De fato, isto ocorre desde que os processadores do arranjo tenham canais de comunicação suficientes para que se possam conectar diretamente os processadores aos quais forem atribuídos elementos que, no circuito simulado, estejam ligados a um mesmo nó. Se isto não for possível, faz-se necessário inserir um processador para fazer a expansão das ligações. Tal processador atua como um ***buffer***, transferindo as tensões e correntes que chegam em n-1 de seus canais, ao n-ésimo canal. O tempo necessário para isto é equivalente ao tempo de uma leitura e uma escrita por informação a ser transferida, vezes 2n-2 informações por canal, vezes n canais por processador, além de ser dependente da velocidade de comunicação dos canais. Como o sistema opera assincronamente, o tempo total, necessário para tais transferências, será uma fração do tempo de processamento de um modelo, desde que se utilizem canais suficientemente velozes.

Tabela 2 - Tempos do processamento matemático no ABACUS - em milisegundos

Circuitos exclusivamente lineares				
Nº níveis → Nº elementos ↓	2	3	4	5
2	0,33			
3	0,33	1,89		
4	0,33	1,89	5,99	
5	0,33	1,89	5,99	12,70
6	0,33	1,89	5,99	12,70
7	0,33	1,89	5,99	12,70
8	0,33	1,89	5,99	12,70
9	0,33	1,89	5,99	12,70
10	0,33	1,89	5,99	12,70
Circuitos não lineares				
Nº níveis → Nº elementos ↓	2	3	4	5
2	0,90			
3	0,90	3,58		
4	0,90	3,58	9,33	
5	0,90	3,58	9,33	17,51
6	0,90	3,58	9,33	17,51
7	0,90	3,58	9,33	17,51
8	0,90	3,58	9,33	17,51
9	0,90	3,58	9,33	17,51
10	0,90	3,58	9,33	17,51

Tabela 3 - Tempos do processamento matemático no PSPICE - em segundos

Circuitos exclusivamente lineares				
Nº níveis → Nº elementos ↓	2	3	4	5
2	4,29			
3	4,33	4,45		
4	4,59	4,71	4,88	
5	4,56	4,82	5,13	5,33
6	4,84	5,11	5,40	5,17
7	4,79	5,27	5,83	5,66
8	7,28	5,66	5,76	6,08
9	4,95	5,97	6,05	5,82
10	5,11	6,98	6,53	5,89
Circuitos não lineares				
Nº níveis → Nº elementos ↓	2	3	4	5
2	5,79			
3	6,03	6,31		
4	7,01	7,42	7,46	
5	8,13	8,51	8,74	8,56
6	8,70	9,22	9,25	9,62
7	9,45	10,61	10,39	10,47
8	10,12	11,55	11,30	11,85
9	11,58	12,03	12,54	12,72
10	11,83	13,28	13,52	13,43

Os dados da tabela 2 indicam que o custo computacional da metodologia ABACUS é proporcional ao número de níveis existentes no circuito, segundo a expressão n^2-2n, onde n representa o número de níveis do circuito considerado. Quando o número de níveis do circuito é muito grande, pode-se aproximar a expressão acima por n^2. Foram observados em torno de 300 circuitos diferentes, com um número de componentes entre 2 e 100; nestes circuitos foram contados até 7 níveis. Além disto, as características dos circuitos digitais observados induzem a um número de níveis dificilmente superior a 10. Portanto, admite-se que, em circuitos LSI, ou maiores, por sua natureza digital, o número de níveis é inferior à raiz quarta do seu número de elementos. Por conseguinte, pode-se inferir que o custo computacional da metodologia ABACUS, no pior caso, segue uma função com razão de crescimento proporcional à raiz quadrada do número de elementos do circuito a ser analisado.

3. Especificação da arquitetura ABACUS:

Analisam-se, a seguir, as características gerais, necessárias à arquitetura ABACUS, para viabilizar a implementação da metodologia proposta.

A arquitetura ABACUS deve ter as seguintes características:

-Um computador hospedeiro composto de quatro processadores, cada um dos quais executando um dos processos do hospedeiro (HCP, HIP, AMP e HOP) e ligado diretamente aos outros três por meio de canais bidirecionais de alta velocidade, de modo a possibilitar o máximo de paralelismo. Para permitir um fluxo rápido de informações entre os processadores do computador hospedeiro, para garantir uma boa precisão dos dados e para viabilizar um amplo espectro desses, tais canais devem ser constituídos por dois barramentos paralelos de 32 bits, cada um dos quais transferindo informações num sentido. A memória de cada um destes processadores deve ser suficientemente grande para permitir o armazenamento de todas as informações referentes ao circuito proposto para simulação. Como foi mencionado no início deste trabalho, o objetivo da metodologia aqui proposta é viabilizar a simulação elétrica de circuitos ULSI. Circuitos desse porte podem conter centenas de milhares, ou mesmo alguns milhões de componentes. O processador 486, da Intel, possui aproximadamente um milhão e meio de componentes e as maiores memórias disponíveis no mercado, de 4Mbits, contêm mais de quatro milhões de componentes. Com base nos dados necessários para descrever um circuito, apresentados no início do presente capítulo e no apêndice I, para definir cada componente são necessários, em média, vinte palavras (cada

palavra na arquitetura proposta tem 4 ***bytes***). Portanto, para que se possa armazenar os dados referentes a um circuito desse porte e ainda se tenha uma área de trabalho razoável, será necessária uma memória da ordem de 500MB ($5x10^6$ componentes vezes 80 ***bytes*** por componente, mais 25% para área de trabalho e programas). O processador responsável pelo controle do computador hospedeiro, que terá uma carga computacional menor, deverá fazer a interface com o usuário. O processador responsável pelo gerenciamento do arranjo deverá ter canais especiais para a transferência de dados entre o hospedeiro e o arranjo, bem como para o controle do arranjo e para a programação das chaves dos barramentos deste.

-Um arranjo de processadores, especialmente projetados para o processamento dos modelos. O ideal seria um arranjo com tantos processadores quantos fossem os elementos do circuito a ser analisado. Os processadores deste arranjo deverão ter uma memória RAM local, para leitura e escrita de dados durante a simulação; uma memória ROM, para o armazenamento permanente dos microprogramas com os modelos dos dispositivos; uma unidade lógica aritmética, configurável de acordo com os microprogramas citados, com capacidade de operação em ponto flutuante; e um conjunto de canais bidirecionais, com ***buffers*** de entrada e saída, para a comunicação entre os processadores do arranjo e destes com o computador hospedeiro.

Como exemplo, considere-se um arranjo com 32728 (2^{15}) processadores, que pode ser imaginado como um circuito com 32 camadas, onde cada camada contenha 32x32 processadores. Entre cada linha e cada coluna de processadores numa mesma camada, passam quatro barramentos paralelos de 32 bits. Cada par destes barramentos permite a conexão dos processadores que o margeiam, através de conjuntos de chaves localizados entre o barramento e as portas de entrada e saída dos processadores. As ligações entre os barramentos e, conseqüentemente, entre os processadores, podem ser realizadas, tanto no plano vertical quanto no horizontal, por meio de blocos de chaves localizados nas regiões de interseção dos barramentos, ou seja, no cruzamento das linhas imaginárias que unem os vértices dos processadores de um mesmo plano horizontal. Esta configuração é mostrada na figura 3.

Neste exemplo, considerando-se o tamanho do arranjo de processadores proposto e um aproveitamento médio de 90% destes processadores simultaneamente, o tamanho médio dos módulos a serem tratados é de 30000 elementos. Portanto, a memória RAM necessária a cada processador do computador hospedeiro, seguindo o mesmo raciocínio desenvolvido no início desta seção, será de 50MB.

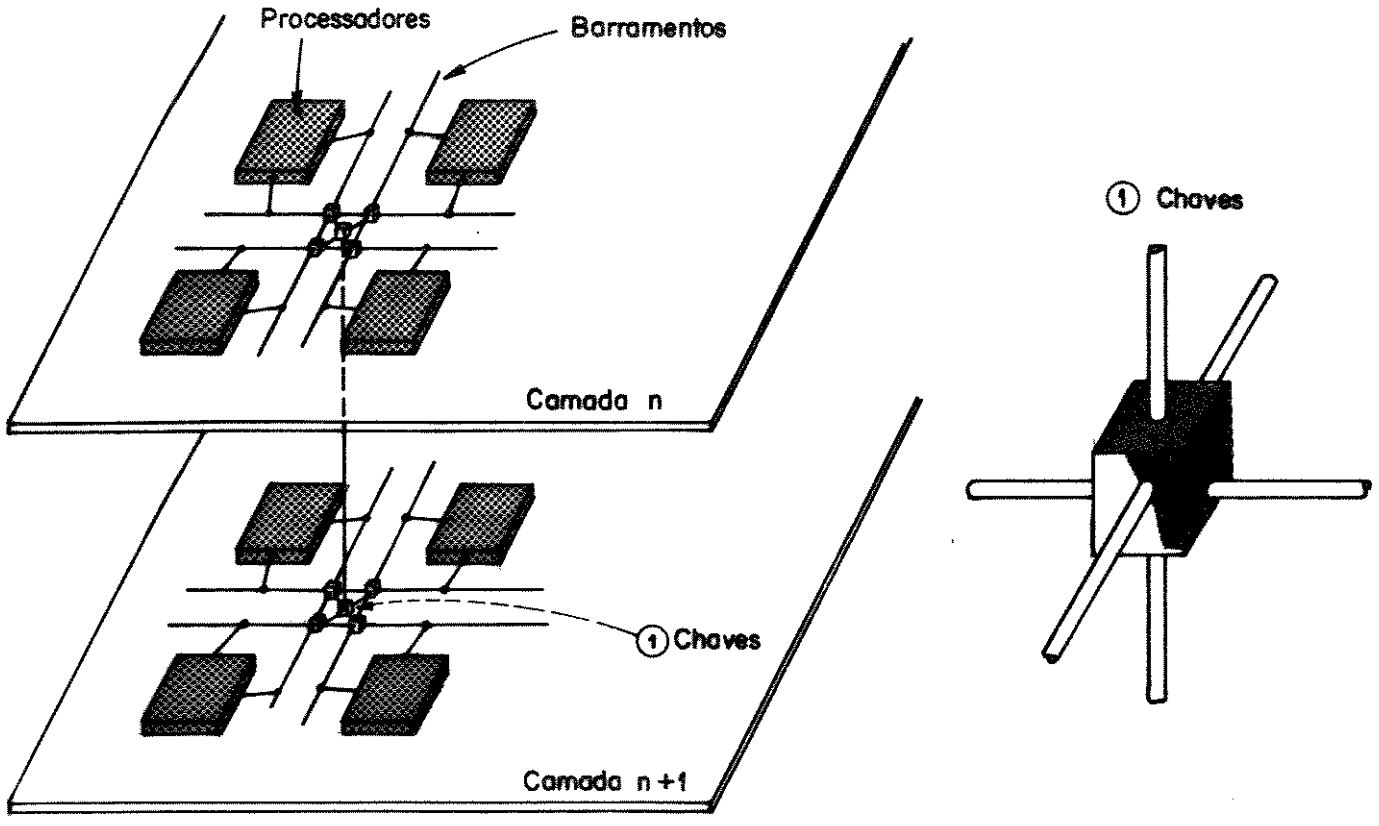

Figura 3 - Esquema de interconexão dos processadores do arranjo

As memórias RAM, dos processadores deste exemplo, deverão ter 1KB, sendo: 256 *bytes* para o *buffer* de comunicação com o hospedeiro; 256 *bytes* referentes às informações do componente e da análise; 128 *bytes* para o armazenamento das informações provenientes dos demais processadores do arranjo e para os resultados parciais da simulação; e os 384 *bytes* restantes para uso como rascunho durante a execução.

O tamanho da memória ROM deve ser tal que possibilite o armazenamento permanente dos microprogramas, com os modelos de dispositivos desejados. Seu tamanho depende, fundamentalmente, da quantidade e da complexidade dos modelos que se deseja armazenar. Além disso, essa memória deve ser programável, para permitir a alteração dos modelos nela gravados, a inclusão de novos modelos e a remoção de modelos em desuso.

O número de canais de um processador do arranjo deve ser definido em função da expectativa de conectividade dos nós num circuito típico. A simetria quadrangular do arranjo aqui apresentado é devida à utilização de quatro filas de entrada e saída (FES) no MPH. Foram empregados quatro canais (4 FES) por considerar-se que a um determinado nó de um circuito, dificilmente convergirão mais do que quatro elementos, exceção feita aos nós terra e àqueles que formam as fontes de alimentação e relógios do circuito, casos em que deverão ser consideradas implementações alternativas, que são discutidas abaixo.

Para os processadores ordinários, têm-se então quatro canais bidirecionais com ***buffers*** de quatro entradas e quatro saídas, os quais servirão para a comunicação entre os processadores do arranjo. Além destes, é necessário um quinto canal, a ser implementado através de um ***buffer*** de pelo menos 256 ***bytes***, como já foi mencionado, diretamente na memória RAM, para comunicação entre cada processador do arranjo e o computador hospedeiro.

Para o caso das exceções, existem três principais possibilidades de solução, quais sejam: a) atribuir mais de um processador para executar a função destes nós especiais; b) expandir os nós com os processadores de nós mencionados anteriormente neste capítulo; c) incluir processadores especiais com um maior número de FES. A alternativa c requer um número muito elevado de FES para atender ao caso genérico. No caso do item b, tem-se a desvantagem da inclusão de atrasos na simulação. A primeira alternativa não aumenta o tempo de simulação e não altera o seu resultado, desde que os processadores sejam corretamente interligados, porém, reduz o número de componentes que podem ser simulados de uma única vez. A opção mais viável é uma solução híbrida das alternativas a e c, onde implementam-se alguns processadores especiais, com um número bem maior de ***buffers***, por exemplo 128, e, se necessário, utilizam-se mais de um desses processadores para um mesmo dispositivo.

Há que se notar, ainda, ser desejável que o sistema aqui descrito opere na maior freqüência possível. Em vista das tecnologias disponíveis atualmente, é razoável admitir a operação do sistema, por exemplo, a uma freqüência de relógio de 100MHz[099].

Para a comunicação entre o hospedeiro e o arranjo, no exemplo proposto, é necessário um barramento de 56bits, cuja estrutura de dados é apresentada na figura 4. Neste barramento, os 15bits menos significativos servem para endereçar, diretamente, cada processador do arranjo; o 16º bit indica se a operação a ser feita é de escrita ou de leitura; os seis bits seguintes constituem o endereço da palavra do ***buffer*** de comunicação (que deve possuir 64 palavras, cada uma com 32 bits) na qual será efetuada a operação; os bits 22 a

53 contêm a informação a ser transferida entre o processador do arranjo e o hospedeiro; e os dois restantes informam a condição de convergência do arranjo e o controle de sua operação.

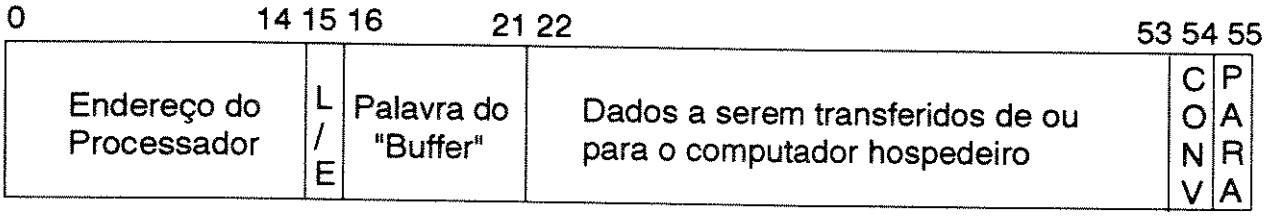

Figura 4 - Estrutura de dados para a comunicação entre o hospedeiro e o arranjo de processadores

Operando paralelamente ao barramento de dados descrito no parágrafo anterior, deve-se ter outro barramento para a programação da matriz de chaves dos barramentos do arranjo. A cada processador correspondem 4 chaves nas linhas e 4 nas colunas de cada camada. Têm-se, portanto, $4x32=128=2^7$ chaves por linha ou coluna. Portanto, o barramento deve ter uma largura de 34 bits (fig. 5) dos quais 5 servem para endereçar as 32 camadas do arranjo, 7 para endereçar as linhas e 7 as colunas de chaves dentro de cada camada. Os demais 15 bits são usados para programar efetivamente as chaves.

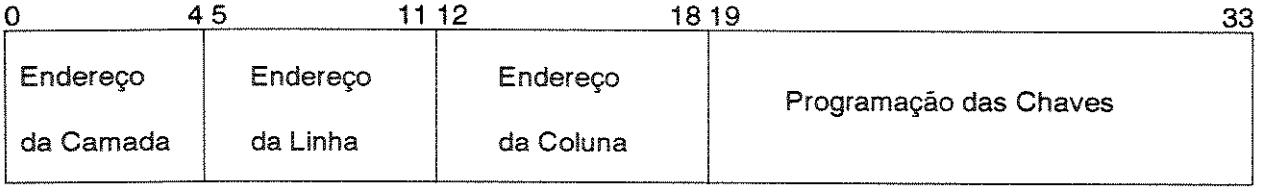

Figura 5 - Estrutura de dados para a programação de chaves dos barramentos do arranjo

A matriz de chaves pode ser vista como uma extrapolação tri-dimensional das chaves dispostas em treliça de Snyder[109], onde incluem-se elementos de memória[069]. Em sua exposição, Snyder utiliza uma matriz plana de chaves, na qual intercala seus processadores elementares (fig.6.a). A matriz aqui proposta é apresentada na figura 6.b, onde as chaves

representadas por círculos vazados correspondem às chaves quadradas intra-camada da figura 7.a. Estas chaves servem para interligar os processadores do arranjo aos barramentos adjacentes e/ou a outros processadores imediatamente vizinhos a eles, dentro de uma mesma camada. As chaves representadas por círculos cheios na fig. 6.b, correspondem às chaves hexagonais intra-camada da figura 7.b, que servem para a interligação dos processadores com vizinhos mais distantes, para as interligações das rotas norte-sul (NS) com as leste-oeste (LO) ou para a conexão das rotas planares (NS/LO) com as chaves de roteamento inter-camadas, representadas na figura 6.b por quadrados cheios e que correspondem às chaves hexagonais da figura 7.c. As chaves inter-camadas permitem a ligação entre os processadores das diversas camadas do arranjo. Todas estas chaves possuem uma pequena memória, o que permite sejam elas programadas no início do processamento, para atender à topologia do circuito a ser simulado, mantendo esta configuração até que o computador hospedeiro a refaça.

Na figura 8, reapresenta-se, com mais detalhes, a arquitetura de um processador do arranjo, os MPHs apresentados no capítulo I.

A unidade lógica e aritmética (ULA) deve ter capacidade de processamento em ponto flutuante e compõe-se de módulos básicos (somadores, multiplicadores, comparadores, etc...) a serem interligados de acordo com um dos modelos de dispositivos, microprogramados, contidos na unidade de modelos armazenados (UMA). O número de linhas entre a UMA e a ULA não foi explicitado por ser fortemente dependente do projeto da ULA. As linhas de controle entre a unidade de controle elementar (UCE) e as FES não constam da figura, para maior clareza do desenho.

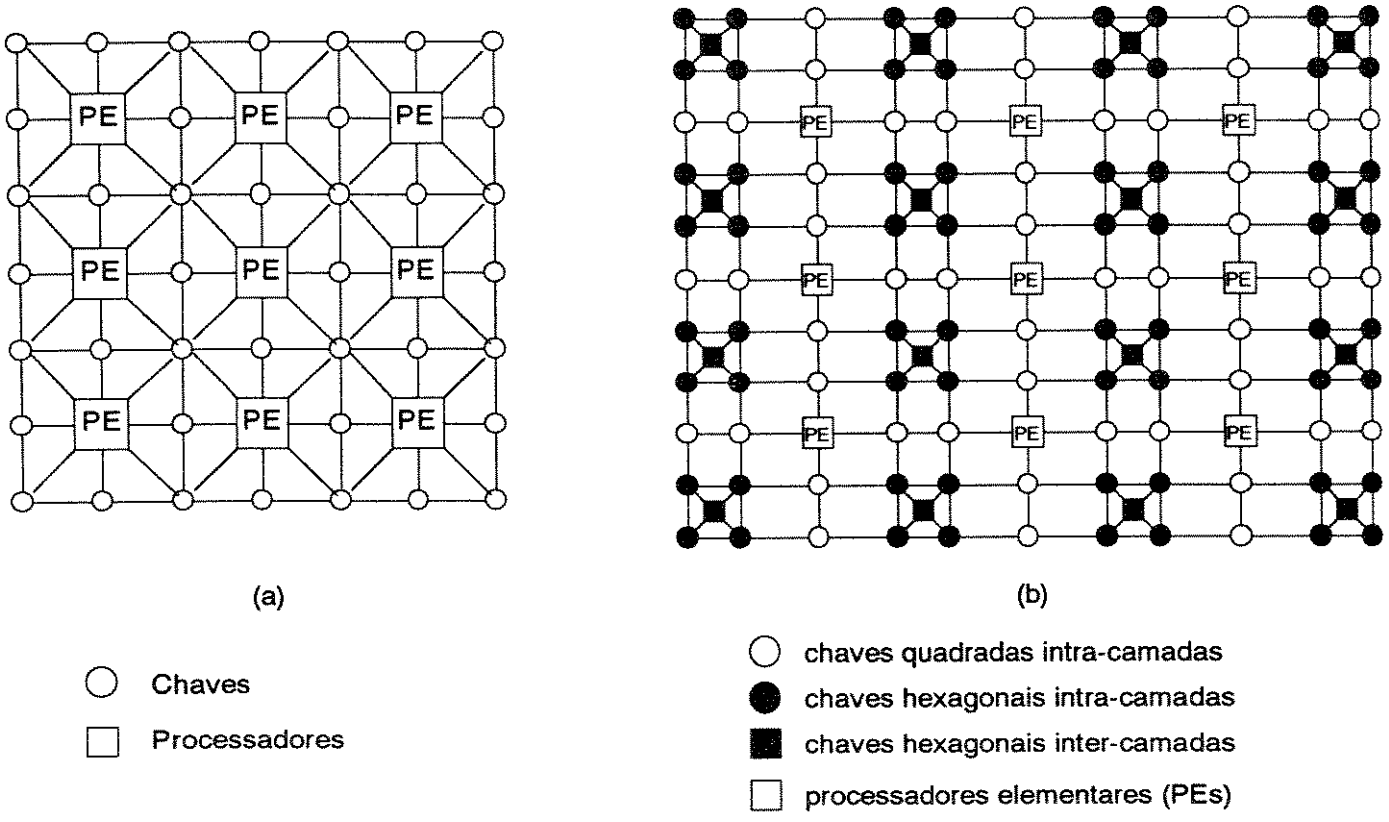

Figura 6 - Matriz de chaves para a interconexão de um arranjo de processadores, reconfigurável: a) matriz plana de Snyder; b) matriz tridimensional proposta

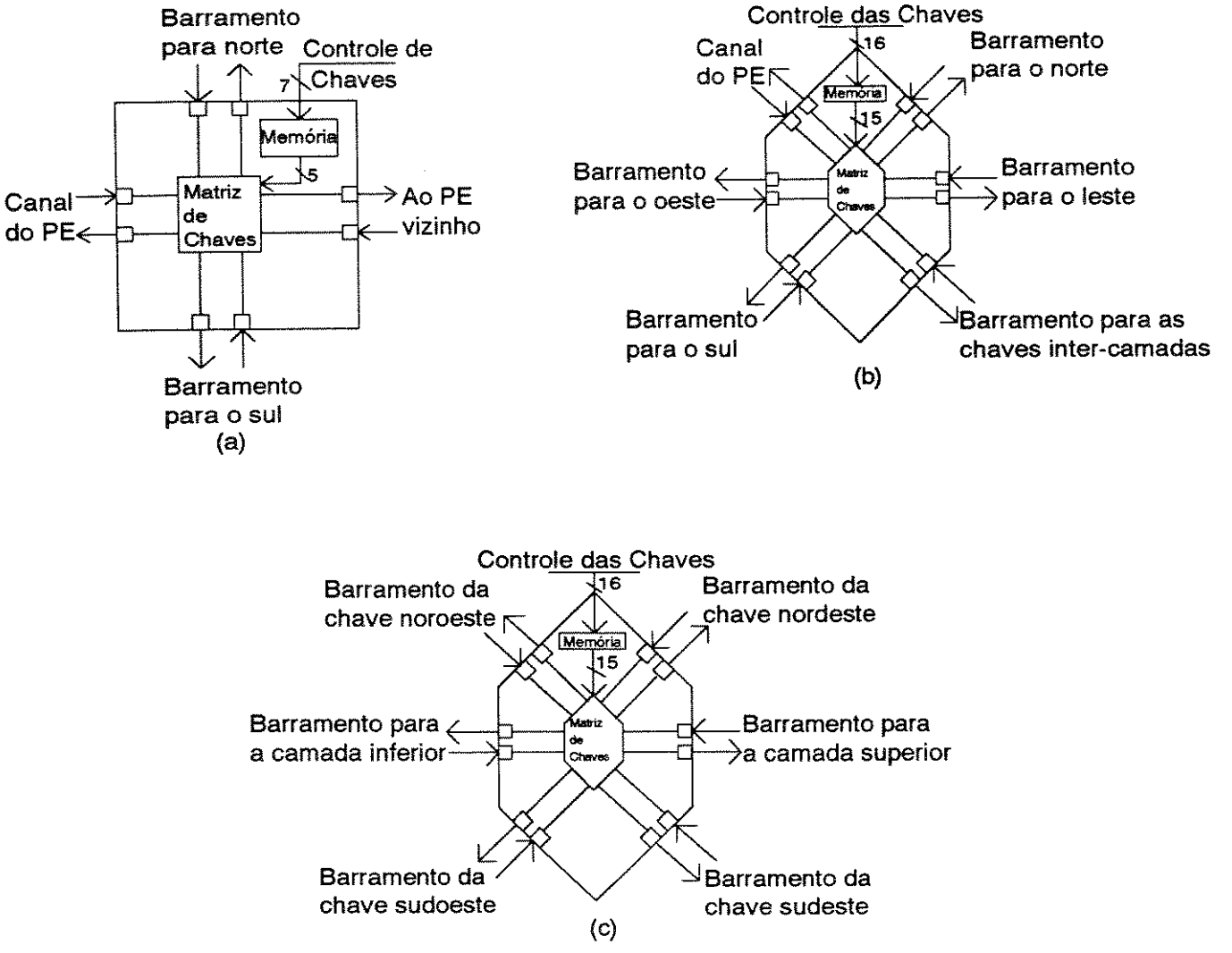

Figura 7 - Arquitetura das chaves dos barramentos do arranjo: a) chave intra-camada, quadrada; b) chave intra-camada, hexagonal; c) chave inter-camada, hexagonal;

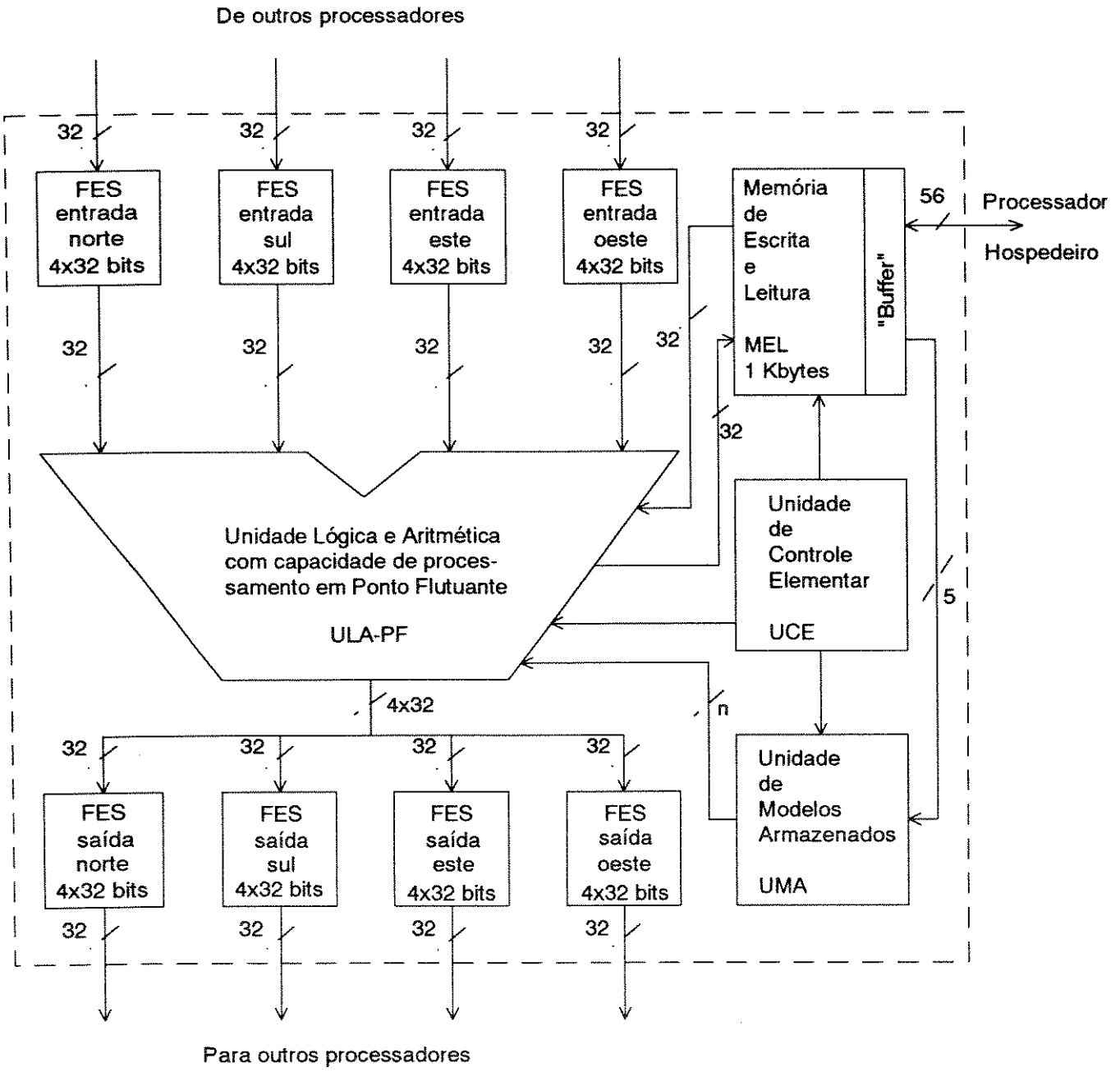

Figura 8 - Arquitetura básica de um processador (MPH) do arranjo

Conclusões

Às vezes me pergunto como pode ter acontecido de eu ter sido o único a desenvolver a Teoria da Relatividade. A razão, creio eu, é que um adulto normal nunca pára para pensar sobre problemas de espaço e tempo. Isso são coisas que ele pensou quando criança. Mas o meu desenvolvimento intelectual foi retardado, motivo pelo qual comecei a questionar sobre espaço e tempo somente quando já era adulto. Naturalmente, pude ir muito mais fundo no problema do que uma criança com suas habilidades normais.

Albert Einstein[113]

A metodologia aqui proposta apresenta características muito interessantes. Trata-se de uma maneira de encarar o problema de simulação de circuitos totalmente diversa da usualmente empregada. Com ela, procura-se tirar o máximo proveito das possibilidades tecnológicas atuais, tais como, o processamento paralelo assíncrono[116,132,146] e a alta capacidade de integração de sistemas[038,088,112,175], para incorporar no *hardware*, da forma mais genérica possível, as características intrínsecas dos circuitos eletrônicos .

Durante este trabalho, observou-se, em conversas que foram mantidas com várias pessoas, uma certa dificuldade das mesmas em entender o esquema de funcionamento da metodologia ABACUS. Na verdade, as pessoas parecem ter uma certa dificuldade em imaginar um processamento assíncrono e a desvinculação entre a simulação de circuitos e os métodos numéricos tradicionalmente utilizados para tanto. Os conceitos, tais como: processamento paralelo, multiprocessamento, arquiteturas não convencionais, processamento assíncrono, etc, parecem ainda meio incipientes e, às vezes, meio confusos, mesmo na bibliografia mais recente[035,112,116,132,145,146,151,158].

A tolerância a falhas[001,078,112] é uma característica muito importante quando se trabalha com sistemas complexos. Uma das vantagens da arquitetura proposta no capítulo III, para a implementação da metodologia ABACUS, é exatamente sua tolerância a falhas. Devido à sua

reconfigurabilidade[027,028], um processador defeituoso pode ser facilmente detetado e não considerado no mapeamento, sem que com isto o sistema sofra uma degradação significativa. O próprio sistema de barramentos programáveis permite uma tolerância a falhas, devido à possibilidade de evitarem-se chaves e trechos de barramentos defeituosos, durante a interconexão dos processadores do arranjo.

Com relação à precisão deste esquema de simulação, considere-se um erro de arredondamento e um erro de truncamento. O erro de arredondamento deprecia um bit do total disponível no tamanho da palavra da máquina. O erro de truncamento obedece a um fator de 1+2log10N, onde N é o número de equações do sistema proposto[111].

Considerando-se que cada processador resolve uma única equação do sistema, o erro devido ao truncamento numérico, neste caso, também é de um bit.

Os dispositivos eletrônicos não conhecem a matemática, a física ou a lógica; eles apenas reagem aos estímulos a que são submetidos, em função de suas características elétricas intrínsecas. Quanto melhor se conseguir reproduzir este comportamento, tanto melhor será a resposta obtida. Obviamente não se pode, pelo menos com o conhecimento disponível hoje, construir um circuito genérico. Também, não é viável construir cada circuito imaginado, somente para efeitos de teste. Portanto, é necessário que se lance mão de modelos para representar as características dos dispositivos em questão. Todavia, há que se insistir na busca de modelos abrangentes e cada vez mais próximos da realidade elétrica deste universo. O que se propõe aqui, é uma tentativa de se chegar um pouco mais perto desta realidade.

A metodologia proposta abre uma nova linha de pesquisa no que tange à simulação de circuitos. Portanto, são necessárias experiência, ferramentas adequadas e mais pessoas para trabalhar no assunto.

Atualmente, o projeto de um processador elementar para a arquitetura ABACUS é assunto de outra tese de doutorado que está sendo desenvolvida no DSIF/FEE/UNICAMP. Além disto, estão sendo propostos outros trabalhos, com vistas à realização da arquitetura ABACUS.

Sugere-se, como primeiro passo para a continuação deste trabalho, a re-simulação de todo o sistema, utilizando uma linguagem de descrição de *hardware*, que possibilite uma representação mais pormenorizada de suas partes operativas.

Paralelamente a isto, é fundamental a investigação de métodos numéricos mais eficientes, adequados à solução global do sistema, com a metodologia ABACUS. A escolha de modelos elaborados de dispositivos, adequados para a sua representação na arquitetura ABACUS, é outro ponto importante que deve ser considerado.

Com relação ao *hardware* proposto, vários elementos podem ser considerados. Por exemplo, desejando-se simular, com freqüência, uma determinada classe de circuitos, na qual seja grande a ocorrência de mais de quatro elementos conectados por nó, gerando a necessidade de inclusão de muitos expansores e degradando o desempenho do sistema, a extensão do arranjo, digamos, para uma simetria octogonal, é razoavelmente simples e direta. A utilização de processadores especiais, como os sugeridos para as fontes de alimentação, constitui outra alternativa de solução para o problema proposto neste parágrafo.

A otimização dos modelos dos dispositivos e do método numérico empregado, provavelmente possibilitará a sustentação de um ganho de velocidade de aproximadamente três ordens de grandeza, em relação às metodologias convencionais mais recentes; além de viabilizar a simulação de circuitos de grande porte, ora impossível.

Outro caminho, muito interessante, conduz à computação óptica[077,133,136]. Perquirir o uso de interconexões ópticas afigura-se como opção natural, assim como a realização das chaves comutadoras dos barramentos por meio de redes neurais ópticas.

Finalmente, resta sugerir que a metodologia tratada no presente trabalho serve, não somente para circuitos eletrônicos, como poderia ser inferido do escopo do mesmo, mas também para outras áreas, como a simulação de sistemas de distribuição de energia elétrica[153], que atualmente se utiliza das mesmas ferramentas empregadas na simulação de circuitos eletrônicos; ou a previsão metereológica[101] onde é necessária a análise de uma malha de pontos que se comunicam com seus vizinhos imediatos causando-lhes um estímulo que é transmitido em cadeia a toda a malha; ou a problemas de simulação de estruturas biológicas[048]; ou ainda a aspectos relacionados com a construção de barcos[014].

Bibliografia

[001] A.D.Singh and S.Murugesan (Editors) "Fault-tolerant systems", IEEE Comp. Mag., vol.23, nº7, 136pp.,1990

[002] A.E.Ruehli & G.S.Ditlow "Circuit analysis, logic simulation and design verification for VLSI", Proc. of the IEEE, 71(1)34-48, 1983

[003] A.L.Reibman and M.Veeraraghavan "Reliability modelling: an overview for system designers", IEEE Comp. Mag., 24(4)49-57, 1991

[004] A.R.Newton "Techniques for the simulation of large scale integrated circuits", IEEE Trans. on Cct. and Syst., 26(9)741-749, 1979

[005] A.R.Newton "Computer-aided design of VLSI circuits", Proc. of the IEEE, 69(10)1189-1199, 1981

[006] A.R.Newton & A.L.S.Vincentelli "Relaxation-based electrical simulation", IEEE Trans. on Elec. Dev., 30(9)1184-1207, 1983

[007] A.R.Newton & A.L.S.Vincentelli "Computer-aided design for VLSI circuits", IEEE Comp. Mag., 19(4)38-59, 1986

[008] A.Telichevesky, A.Ichihara, D.F.G.Azevedo e C.C.Coitinho "Minuano - um simulador de lógica", Rel. Tec. nº2, CPGCC/UFRGS, 89pp., 1982

[009] Autor desconhecido "ELDO - a high performance VLSI circuit simulator", Advanced Comp. Syst., 10pp., 1989

[010] Autor desconhecido "FIDELDO - a true VLSI mixed-mode simulator", Advanced Comp. Syst., 18pp, 1989

[011] Autor desconhecido "PSPICE electrical circuit simulator", MicroSim Corp., 138pp., 1987

[012] Autores diversos "Design methodologies and CAD tools", Proc. of the I Brazilian Microelectronics School, pp.17-279, 1990

[013] A.Vladimirescu and S.Liu "The simulation of MOS integrated circuits using SPICE2", UCB/ERL- M80/7, 66pp., 1980

[014] A.W.Browning "A mathematical model to simulate small boat behaviour", SCS Jrnl., 56(5)329-336, 1991

[015] B.C.Damasceno & A.Yamakami "FMS: a tool for analysis and simulation based on numerical Petri nets (NPN)", trabalho apresentado ao 30º encontro internacional do Institute of Management Sciences, 15-17 de julho de 1991

[016] B.J.Sheu "MOS transistor modelling and characterization for circuit simulation", Ph.D. Thesis, UCB/ERL-M85/85, 210pp., 1985

[017] B.P.Demidovich & I.A.Maron "Computational mathematics", Mir Publishers, 687pp., 1987

[018] B.R.Chawla, H.K.Gummel and P.Kozak "MOTIS - an MOS timing simulator", IEEE Trans. on Cct. and Syst., 22(12)901-910, 1975

[019] C.A.R.Hoare "Communicating sequential processes", CACM, 21(8)666-677, 1978

[020] C.A.Zukowski "Relaxing bounds for linear RC mesh circuits", IEEE Trans. on C.A.D. of ICs and Syst., 5(2)305-312, 1986

[021] C.E.Stroud, R.R.Muñoz and D.A.Pierce "Behavioral model synthesis with cones", IEEE Des. & Test of Comp., 5(3)31-42, 1988

[022] C.K.Cheng and Y.C.A.Wei "An improved two-way partitioning algorithm with stable performance", IEEE Trans. on C.A.D. of ICs and Syst., 10(12)1502-1511, 1991

[023] C.L.Seitz "The cosmic cube", CACM, 28(1)22-33, 1985

[024] C.MacInnes "The use of small pivot perturbation in circuit analysis", IEEE Trans. on C.A.D. of ICs and Syst., 10(11)1441-1446, 1991

[025] C.Moler and D.S.Scott "Communication utilities for the iPSC", iPSC Tech. Rep. nº.2, 32pp., 1986

[026] C.Viesweswariah and R.A.Rohrer "Piecewise approximate circuit simulation", IEEE Trans. on C.A.D. of ICs and Syst., 10(7)861-870, 1991

[027] D.A.Nicole, E.K.Lloyd and J.S.Ward "Switching networks for transputer links", Tech. Rep. 1, Esprit P1085, Southampton Transputer Support Centre, 21pp., Sep. 1988

[028] D.A.Nicole "Reconfigurable transputer processor architecture", Tech. Rep. 2, Esprit P1085, Southampton Transputer Support Centre, 18pp., Sep. 1988

[029] D.A.Zein and C.W.Ho "APL simplifies interactive circuit design", Electronic Design, pp.81-88+90, 31 Mar. 1982

[030] D.A.Zein "Solution of a set of nonlinear algebraic equations for general purpose CAD programs", IEEE Cct. & Dev. Mag., (5)7-20, 1985

[031] D.D.Gajski, D.A.Padua, D.J.Kuck and R.H.Kuhn "A second opinion on dataflow machines and languages", IEEE Comp. Mag., 15(2)58-70, 1982

[032] D.Dumlugöl, H.J.DeMan, P.Stevens and G.G.Schrooten "Local relaxation algorithms for event driven simulation of MOS networks including assignable delay modelling", IEEE Trans. on C.A.D. of ICs and Syst., 2(3)193-202, 1983

[033] D.Dumlugöl "The segmented waveform relaxation method for mixed-mode simulation of digital VLSI circuits", Ph.D. Thesis, Katholieke Universiteit Leuven, 248pp., 1986

[034] D.Eldredge, J.D.McGregor and M.K.Summers "Applying the object-oriented paradigm

to discrete event simulations using C++ language", SCS Jrnl., 54(2)83-91, 1990

[035] D.G.Feitelson and L.Rudolph "Distributed hierarchical control for parallel processing", IEEE Comp. Mag., 23(5)65-77, 1990

[036] D.J.Erdman and D.J.Rose "A Newton waveform relaxation approach to circuit simulation", Tech. Rep. 89-43, 99pp., MCNC, 1989

[037] D.J.Erdman, S.W.Kenkel, G.B.Nifong, D.J.Rose and R.Subrahmanyan "CAzM: a numerically robust, table-based circuit simulator", Tech. Rep. 89-23, 49pp., MCNC, 1989

[038] D.L.Landis "A test methodology for wafer scale systems", IEEE Trans. on C.A.D. of ICs and Syst., 11(1)76-82, 1992

[039] D.M.Lewis "A hierarchical compiled code event-driven logic simulator", IEEE Trans. on C.A.D. of ICs and Ccts., 10(6)729-737, 1991

[040] D.O.Pederson "A historical review of circuit simulation", Trans. on Ccts. and Syst., 31(1)103-111, 1984

[041] D.S.Reves, F.Brglez, K.Kozminski and G.Kedem "Design of ASICs with OASIS", paper presented at the 1st Brazilian Microelectronics School, 8pp., 1990

[042] D.Tsao and C.F.Chen "A fast timing simulator for digital MOS circuits", IEEE Trans. on C.A.D. of ICs and Syst., 5(4)536-540, 1986

[043] D.W.Blewis, E.W.Davis and J.H.Reif "Processing element and custom chip architecture for the BLITZEN massively parallel processor", Tech. Rep. 87-22, 27pp., MCNC, 1987

[044] E.DeBenedictis, S.Ghosh and M.L.Yu "A novel algorithm for discrete-event simulation", IEEE Comp. Mag., 24(6)21-33, 1991

[045] E.G.Ulrich "Exclusive simulation of activity in digital networks", CACM, 12(2)102-110, 1969

[046] E.G.Ulrich "Event manipulation for discrete simulations requiring large numbers of events", CACM, 21(9)777-785, 1978

[047] E.Lelarasmee, A.E.Ruehli and A.L.S.Vincentelli "The waveform relaxation method for time domain analysis of large scale integrated circuits", IEEE Trans. on C.A.D. of ICs and Syst., 1(3)131-145, 1982

[048] E.S.Lander, R.Langridge and D.M.Saccocio "Computing in molecular biology: mapping and interpreting biological information", IEEE Comp. Mag., 24(11)6-13, 1991

[049] E.W.Davis and J.Rosenberg "System architecture concepts for BLITZEN massively parallel machines", Tech. Rep. 88-30, 29pp., MCNC, 1988

[050] E.W.Thompson and S.A.Szygenda "Digital logic simulation in a time-based, table-driven environment - Part 2. Parallel fault simulation", IEEE Comp. Mag., 8(3)38-49, 1975

[051] F.Guterl, P.Wallich and M.A.Fischetti "Inpursuit of the one-month chip", IEEE Spectrum Mag., 21(9)28-49, 1984

[052] F.H.Branin Jr. "Computer methods of network analysis", Proc of the IEEE, 55(11)1787-1801, 1967

[053] F.H.Branin Jr., J.R.Hogsett, R.L.Lunde and L.E.Kugel "ECAP II - a new electronic circuit analysis program", IEEE Jrnl. of Sol. State Cct., 6(4)146-166, 1971

[054] F.J.Hill, Y.Chu, D.L.Dietmeyer and D.Siewiorek "Introducing Hardware description languages", IEEE Comp. Mag., 7(12)27-44, 1974

[055] F.R.Wagner "Basic techniques for gate level simulation", Rel. Tec. nº12, CPGCC/UFRGS, 54pp., 1984

[056] F.R.Wagner "Algoritmos de simulação de hardware no nível RT", Rel. Tec. nº21, CPGCC/UFRGS, 28pp., 1985

[057] F.R.Wagner "On the properties of event oriented logic simulation according to significant timing models", Rel. Tec. nº22, CPGCC/UFRGS, 13pp., 1985

[058] F.S.Jenkins and S.P.Fan "TIME - a nonlinear DC time domain circuit simulation program", IEEE Jrn;. of Sol. State Cct., 6(4)182-188, 1971

[059] F.T.Leighton and A.L.Rosenberg "Three-dimensional circuit layouts", Tech. Rep. 84-08, 33pp., MCNC, 1984

[060] F.Yamamoto and S.Takahashi "Vectorized LU decomposition algorithms for large scale circuit simulation", IEEE Trans. on C.A.D. of ICs and Syst., 4(3)232-239, 1985

[061] G.Arnout and H.J.DeMan "The use of threshold functions and boolean controlled network elements for macromodelling of LSI circuits", IEEE Jrn;. of Sol. State Cct., 13(3)326-332, 1978

[062] G.DeMichelle, A.R.Newton and A.L.S.Vincentelli "Symmetric displacement algorithms for the timing analysis of large scale circuits", IEEE Trans. on C.A.D. of ICs and Syst., 2(3)167-180, 1983

[063] G.D.Hachtel and A.L.S.Vincentelli "A survey of third generation simulation techniques", Proc. of the IEEE, 69(10)1264-1280, 1981

[064] G.G.El-Karim, D.Erdman and S.Kenkel "Design-oriented mixed-level simulation", Tech. Rep. 89-37, 41pp., MCNC, 1989

[065] G.Kedem "Multiple bus interconnection schemes", Tech. Rep. 86-13, 7pp., MCNC, 1986

[066] G.Kedem and F.Brglez "OASIS: open architecture silicon implementation system", Tech. Rep. 88-06, 10pp., MCNC, 1988

[067] G.Ruan, J.Vlach, J.Barby and A.Opal "Analog functional simulator for multilevel systems", IEEE Trans. C.A.D. of ICs and Syst., 10(5)565-576, 1991

[068] G.Zimmermann "Playout - a hierarchical design system", Proc. of the IFIP - Information Processing 89, pp.905-910, 1989

[069] H.E.Mizrahi, J.L.Baer, E.D.Lazowska and J.Zahorjan "Introducing memory into the switch elements of multiprocessor interconnection networks", ACM Comp. Archit. News, 17(3)158-166, 1989.

[070] H.J.DeMan "Computer-aided design for integrated circuits: trying to bridge the gap", IEEE Jrnl. of Sol. State Cct., 14(3)613-621, 1979

[071] H.J.DeMan, G.Arnout and P.Reynaert "Mixed-mode simulation techniques and their implementation in DIANA", Nato Advanced Series Institutes, series E48, pp.113-174, 1981

[072] H.J.DeMan "Mixed-mode simulation for MOS-VLSI: why, where and how?", uma publicação do IEEE, pp.699-701, 1982

[073] H.J.DeMan, J.Rabaey, J.Vanhoof, G.Goossens, P.Six and L.Claesen "CATHEDRAL II - a computer- aided synthesis system for digital signal processing VLSI systems", C.A.E.Jrnl., pp.55-66, Apr. 1988

[074] H.J.DeMan, A.R.Newton, P.Odryna, R.A.Rohrer, D.Smith and P.Weil "A D&T roundtable: Mixed- mode simulation", IEEE Des. & Test of Comp. Mag., 6(2)67-75, 1989

[075] H.Shichman "Computation of DC solutions for bipolar transistor networks", IEEE Trans. on Cct. and Syst, 16(4)460-466, 1969

[076] H.Shichman "Integration system of a nonlinear DC analysis program", IEEE Trans. on Cct. and Syst, 17(3)378-386, 1970

[077] H.S.Stone and J.Cocke "Computer architecture in the 1990s", IEEE Comp. Mag., 24(9)30-38, 1991

[078] H.T.Kung "Why systolic architectures?", IEEE Comp. Mag., 15(1)37-46, 1982

[079] H.W.Carter "Computer-aided design of integrated circuits", IEEE Comp. Mag., 19(4)19-36, 1986

[080] H.Y.Chang and S.G.Chappell "Deductive techniques for simulating logic circuits", IEEE Comp. Mag., 8(3)52-59, 1975

[081] H.Y.Chang, G.W.Smith, R.B.Walford, S.G.Chappell, C.H.Elmendorf, L.D.Schmidt, G.W.Heinbigner, T.T.Butler, T.G.Hallin, J.J.Kulzer and W.Johnson "LAMP: Logic analyser for maintenance planning", Bell Syst. Tech. Jrnl., 53(8)1431-1555, 1974

[082] I.Koren and I.Peled "The concept and implemantation of data-driven processor arrays", IEEE Comp. Mag., 20(7)102-103, 1987

[083] INMOS Inc. "The Transputer databook", Bath Press Ltd., 477pp., 1988

[084] J.Backus "Can programming be liberated from the von Neumann style? A functional style and its algebra of programs", CACM, 21(8)613-641, 1978

[085] J.Banks "Verifying and validating complex simulation models by analogy", SCS Jrnl., 54(1)33-36, 1990

[086] J.B.Dennis "Dataflow supercomputers", IEEE Comp. Mag., 13(11)48-56, 1980

[087] J.C.Bowers and S.R.Sedore "SCEPTRE: a computer program for circuit and systems analysis. Chapter 1 - Mathematical formulation", Prentice Hall Inc, pp.2-29, 1971

[088] J.Cong and C.L.Liu "On the k-layer planar subset and topological via minimization problems", IEEE Trans. on C.A.D. of ICs and Syst., 10(8)972-981, 1991

[089] J.E.Brandenburg and D.S.Scott "Embeddings of communication trees and grids into hypercubes", iPSC Tech. Rep. nº.1, 7pp., 1986

[090] J.H.Moreno and T.Lang "Matrix computations on systolic-type meshes", IEEE Comp. Mag., 23(4)32- 51, 1990

[091] J.J.Navarro, J.M.Llaberia and M.Valero "Partitioning: an essential step in mapping algorithms into systolic array processors", IEEE Comp. Mag., 20(7)77-88, 1987

[092] J.L.Peterson "Petri theory and modelling of systems", Prentice-Hall Inc., 263pp., 1981

[093] J.M.Ortega and W.C.Rheinboldt "Iterative solution of nonlinear equations in several variables", Academic Press Inc., 572pp., 1970

[094] J.Rattner "Concurrent processing: a new direction in scientific computing", AFIPS - National Comp. Conf. Proc., vol.54, pp.157-165, 1985

[095] J.R.Bunch and D.J.Rose "Sparse matrix computation", Academic Press Inc., 453pp., 1976

[096] J.R.Rice "Matrix computation and mathematical software", McGraw Hill Book Co. Japan Ltd.,248pp., 1983

[097] J.Vaucher, J.Mermet, R.Piloty, F.Marcoz, O.Tedone, H.Watanabe, K.Fujino and B.Demisoglu "A survey of computer hardware description languages abroad", IEEE Comp. Mag., 7(12)52-66, 1974

[098] J.Vlach and K.Singhal "Computer methods for circuit analysis and design", Van Nostrand Reinhold Co., 594pp., 1983

[099] J.Yuan and C.Svensson "Pushing the limits of standard CMOS", IEEE Spectrum Mag., 28(2)52-53, 1991

[100] K.Chaudhary and P.Robinson "Channel routing by sorting", IEEE Trans. on C.A.D. of ICs and Syst., 10(6)754-760, 1991

[101] K.Hwang and F.A.Briggs "Computer architecture and parallel processing", McGraw Hill Book Co., 845pp., 1985

[102] K.Kozminski "Design control in MCNC's open architecture silicon implementation

system OASIS", Tech. Rep, 89-54, 9pp., MCNC, 1989

[103] K.A.Sakallah "Mixed simulation of electronic integrated circuits", Ph.D. Thesis, EDRC, 160pp., 1981

[104] K.A.Sakallah and S.W.Director "SAMSON2: an event driven VLSI circuit simulator", IEEE Trans. on C.A.D. of ICs and Syst., 4(4)668-684, 1985

[105] L.E.Bays, C.F.Chen, E.M.Fields, R.N.Gandez, W.P.Hays, H.S.Moscovits and T.G.Szymanski "Post- layout verification of the WE DSP2 digital signal processor", IEEE Des. & Test of Comp. Mag., 6(2)56-66, 1989

[106] L.M.Vidigal, S.R.Nassif and S.W.Director "CINNAMON: coupled integration and nodal analysis of MOS networks", Proc. of the 23rd DAC, pp. 179-185, 1986

[107] L.Peterson and S.Mattison "Circuit simulation on a hypercube", Proc. of the ISCAS, pp.1119-1122, 1988

[108] L.Peterson and S.Mattison "Circuit partitioning and iteration scheme for waveform relaxation on multicomputers", Proc. of the ISCAS, pp.570-573, 1989

[109] L.Snyder "Introduction to the configurable, highly parallel computer", IEEE Comp. Mag., 15(1)47- 56, 1982

[110] L.W.Nagel and R.A.Rohrer "Computer analysis of nonlinear circuits excluding radiation (CANCER)", IEEE Jrnl. of Sol. State Cct., 6(4)166-182, 1971

[111] L.W.Nagel "SPICE2: a computer program to simulate semiconductor circuits", Ph.D. Thesis, UCB/ERL-M520, 413pp., 1975

[112] M.Chean and J.A.B.Fortes "A taxonomy of reconfiguration techniques for fault-tolerant processor arrays", IEEE Comp. Mag., 23(1)55-69, 1990

[113] M.Claret "O pensamento vivo de Einstein", Martin Claret Editores Ltda., 5ª edição,

109pp., 1986

[114] M.C.G.Madureira "Contribuição à análise e síntese de circuitos digitais", Dissertação de mestrado, CPG/FEE/UNICAMP, 170pp., 1987

[115] M.D.Matson and L.A.Glasser "Macromodelling and optimization of digital MOS VLSI circuits", IEEE Trans. on C.A.D. of ICs and Syst., 5(4)659-678, 1986

[116] M.Dubois and S.Thakkar "Cache architectures in tightly coupled multiprocessors", IEEE Comp. Mag., 23(6)9-11, 1990

[117] M.Ferreti "Le architeture sistoliche", Alta Frequenza, 56(3)41-54, 1987

[118] M.Fiedler "Special matrices and their application in numerical mathematics", Martinus Nijhoff Publishers, 307pp., 1986

[119] M.G.Walker and J.McGregor "Computer-aided engineering for analog circuit design", IEEE Comp. Mag., 19(4)100-108, 1986

[120] M.I.Elmasry "Digital VLSI Systems", IEEE Press, 572pp., 1985

[121] M.Marton "Analisador automático de redes de Petri temporizadas", dissertação de mestrado, FEE/UNICAMP, 107pp., 1989

[122] M.P.C.Arantes "Analisador automático de redes de Petri para protocolos de comunicação", dissertação de mestrado, FEE/UNICAMP, 169pp., 1988

[123] M.Yanilmaz "Numerical MOSFET modelling for time increment circuit simulation", LDES Rep. nº89-1, 36pp., 1989

[124] N.B.G.Rabat, A.L.S.Vincentelli and H.Y.Hsieh "A multilevel Newton algorithm with macromodelling and latency for the analysis of large scale nonlinear circuits in the time domain", IEEE Trans. on Cct. and Syst., 26(9)733-741, 1979

[125] N.H.Gehani and W.D.Roome "Concurrent C", Software Practice and Experience, 16(9)821-844, 1986

[126] N.H.Gehani and W.D.Roome "Concurrent C++", Software Practice and Experience, 18(12)1157-1177, 1988

[127] N.Marranghello e A.J.Monticelli "Estudo sobre a viabilidade de implementação de um simulador híbrido baseado no programa de simulação em nível de circuitos SPICE", Rel. Tec. nº010/88, 41pp., 1988

[128] N.Marranghello e F.Damiani "Um estudo sobre a simulação de circuitos", Anais do 10º seminário ADUNESP - Guaratinguetá, pp167-174, 1989

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

[130] N.Marranghello e F.Damiani "A methodology for circuit simulation", Proc. of the 22nd SCSC, pp.111- 115, 1990

[131] N.Marranghello e F.Damiani "Architecture and algorithm of a circuit simulator", Proc. SPIE, vol.1405, pp.167-173, 1990

[132] O.Serlin "Parallel processing: Fact or fancy?", Datamation, pp.112-118, Jun. 1986

[133] P.A.Mitkas "Optical computing: the next step?", IEEE Computing Features, 2nd issue, pp.61-66, 1991

[134] P.Antognetti and A.DeGloria "La progettazione VLSI", Alta Frequenza, 56(9)195-202, 1987

[135] P.B.Hansen "Distributed processes: A concurrent programming concept", CACM, 21(11)934-941, 1978

[136] P.E.Green "The future of fiber-optic computer networks", IEEE Comp. Mag.,

24(9)78-87, 1991

[137] P.F.Cox, R.G.Burch, D.E.Hocevar, P.Yang and B.D.Epler "Direct circuit simulation algorithms for parallel processing", IEEE Trans. on C.A.D. of ICs and Syst., 10(6)714-725, 1991

[138] P.Feldman, T.V.Nguyen, S.W.Director and R.A.Rohrer "Sensitivity computation in piecewise approximate circuit simulation", IEEE Trans. on C.A.D. of ICs and Syst., 10(2)171-183, 1991

[139] P.Fraigniaud "Complexity analysis of broadcasting in hypercubes with restricted communication capabilities", Rapport 90-16, École Normale Supérieure de Lyon, LIP, 1990

[140] P.Heidelberger "Discrete event simulation", CACM, 33(10)28-29, 1990

[141] P.Odent, D.Dumlugöl and H.J.DeMan "Hardware acceleration of circuit simulation on a multi- microprocessor system", Intl. Workshop on Hardware Accelerators, Univ. of Oxford, England, 10pp., 30 Sep. - 02 Oct., 1987

[142] P.Odent, L.Claesen and H.J.DeMan "New parallel techniques for simulating MOS circuits with waveform relaxation algorithms in CSWAN", Proc. of the ISCAS, 4pp., 1989

[143] P.Odent, L.Claesen and H.J.DeMan "Feedback loops and large subcircuits in the multiprocessor implementation of a relaxation based circuit simulator", Proc. of the DAC, 6pp., 1989

[144] R.D.Chamberlain and M.A.Franklin "Collecting data about logic simulation", IEEE Trans. on C.A.D. of ICs and Syst., 5(3)405-412, 1986

[145] R.D.Rettberg, W.R.Crowther, P.P.Carrey and R.S.Tomlinson "The Monarch parallel processor hardware design", IEEE Comp. Mag., 23(4)18-30, 1990

[146] R.Duncan "A survey of parallel computer architectures", IEEE Comp. Mag., 23(2)5-16, 1990

[147] R.E.Oakley and R.J.Hocking "CASMOS - an accurate MOS model with geometry dependent parameters", IEE Proc. Part I, 128(6)239-247, 1981

[148] R.Goering (Ed.) "A full range of solutions emerge to handle mixed-mode simulation", Comp. Des. Mag., 27(3)57-65, 1988

[149] R.H.Silva "Advanced statistical analysis program (ASTAP)", monografia apresentada ao curso de IE325 - tópicos especiais em microeletrônica - CPG/FEE/UNICAMP, 55pp., 1º semestre de 1987

[150] R.K.Brayton, G.D.Hachtel and A.L.S.Vincentelli "A survey of optimization techniques for integrated- circuit design", Proc. of the IEEE, 69(10)1334-1362, 1981

[151] R.L.Jones "Do they really pay for every line?", Dr. Dobb's Jrnl., 13(8)130-138, 1988

[152] R.M.Fijimoto "Parallel discrete event simulation", CACM, 33(10)30-53, 1990

[153] R.M.Nelms, G.B.Sheblé, S.R.Newton and L.L.Grisby "Simulation of transmission line transients with a distributed-parameter model using a personal computer", SCS Jrnl., 55(2)103-108, 1990

[154] R.Salama, W.Liu and W.T.Krakow "VLSI simulation in a concurrent environment", Tech. Rep. 87-07, 18pp., MCNC, 1987

[155] R.Waxman "Hardware description languages for computer design and test", IEEE Comp. Mag., 19(4)90-97, 1986

[156] S.A.Kravitz, R.E.Bryant and R.A.Rutenbar "Massively parallel switch level simulation: a feasibility study", IEEE Trans. on C.A.D. of ICs and Syst., 10(7)871-894, 1991

[157] S.A.Szygenda and E.W.Thompson "Digital logic simulation in a time-based, table-driven environment - Part 1. Design Verification", IEEE Comp. Mag., 8(3)24-36, 1975

[158] S.Dasgupta "A hierarchical taxonomic system for computer architectures", IEEE Comp.

Mag., 23(3)64-74, 1990

[159] S.Dimitriadis and W.J.Karplus "Multiprocessor implementation of algorithms for ordinary differential equations", SCS Jrnl., 55(4)236-246, 1990

[160] S.Ghosh "Behavioral-level fault simulation", IEEE Des. & Test of Comp., 5(3)31-42, 1988

[161] S.Liu "A unified CAD model for MOSFETS", Ph.D. Thesis, UCB/ERL-M81/31, 72pp., 1981

[162] S.Mattison "CONCISE - a concurrent circuit simulation program", Ph.D. Thesis, Lund University, 86pp., 1986

[163] S.P.Fan, M.Y.Hsueh, A.R.Newton and D.O.Pederson "MOTIS-C: a new circuit simulator for MOS LSI circuits", Proc. of the ISCAS, pp.700-703, 1977

[164] S.Sakai, Y.Yamaguchi, K.Hiraki, Y.Kodama and T.Yuba "Architecture of a dataflow single chip processor", ACM Comp. Archit. News, 17(3)46-53, 1989

[165] S.S.Takkar (Ed.) "Selected reprints on dataflow and reduction architectures", IEEE Comp. Soc. Press, 446pp., 1987

[166] Subtlesoft International "Parallel C", Product Description, 1991

[167] S.W.Director "Survey of circuit-oriented optimization techniques", IEEE Trans. on Cct. Theory, 18(1)3-10, 1971

[168] S.Y.Foo and Y.Takefuji "Databases and cell-selection algorithms for VLSI cell libraries", IEEE Comp. Mag., 23(2)18-30, 1990

[169] S.Y.H.Su "A survey of computer hardware description languages in the U.S.A.", IEEE Comp. Mag., 7(12)45-51, 1974

[170] S.Y.Kung, K.S.Arun, R.J.G.Ezer and D.V.B.Rao "Wavefront array processor: language, architecture and applications", IEEE Trans. on Comp., 31(11)1054-1066, 1982

[171] S.Y.Kung "On supercomputing with systolic/wavefront array processors", Proc. of the IEEE, 72(7)867-884, 1984

[172] 3L Ltd "Parallel C, V2.2", Software Product Description, Aug. 1991

[173] 3L Ltd "Parallel C++, V2.1", Software Product Description, Jan. 1992

[174] T.E.Idleman, F.S.Jenkins, W.J.McCalla and D.O.Pederson "SLIC - a simulator for linear integrated circuits", IEEE Jrnl. of Sol. State Cct., 6(4)188-203, 1971

[175] T.Leighton and C.E.Leiserson "Wafer scale integration of systolic arrays", Trans. on Comp., 34(5)448-461, 1985

[176] T.Smigelski, T.Murata & M.Sowa "A timed Petri net model and simulation of a dataflow computer", IEEE Trans. on Comps, pp.56-63, 1985

[177] T.Yumasaki, K.Shima, S.Komori, H.Takata, T.Tamura, F.Asai, T.Ohno, O.Tomisawa and H.Terada "VLSI implementation of a variable length pipeline scheme for data-driven processors", IEEE Jrnl. of Sol. State Ccts., 24(4)933-937, 1989

[178] U.Schendel and B.W.Conoly "Introduction to numerical methods for parallel processing", Hellis Horwood Ltd., 151pp., 1984

[179] V.Malbassa "A multiprocessor system for dynamic system simulation", SCS Jrnl., 56(1)31-40, 1991

[180] V.P.Srini "An architectural comparison of data flow systems", IEEE Comp. Mag., 19(3)68-88, 1986

[181] V.R.B.Santos "Curso de cálculo numérico", LTC Editores, 1974

[182] W.J.McCalla "BIAS3 - a program for the nonlinear DC analysis of bipolar transistor circuits", IEEE Jrnl. of Sol. State Ccts, 6(1)14-19, 1971

[183] W.J.McCalla and D.O.Pederson "Elements of computer-aided circuit analysis", IEEE Trans. on Cct. Theory, 18(1)14-26, 1971

[184] W.L.Engl, R.Laur and H.K.Dirks "MEDUSA - a simulator for modular circuits", IEEE Trans. on C.A.D. of ICs and Syst., 1(2)85-93, 1982

[185] W.M.Coughran Jr., E.Grose and D.J.Rose "CAzM: a circuit analyser with macromodelling", IEEE Trans. on Elec. Dev., 30(9)1207-1213, 1983

[186] W.M.Coughran Jr., E.Grose and D.J.Rose "Aspects of computational circuit analysis", Tech. Rep. 86- 08, 23pp., MCNC, 1986

[187] W.T.Weeks, A.J.Jimenez, G.W.Mahoney, D.Mehta, H.Quassemzadeh and T.R.Scott "Algorithms for ASTAP - a network analysis program", IEEE Trans. on Cct. Theory, 20(6)628-634, 1973

[188] Y.Chu "Why do we need computer hardware description languages?", IEEE Comp. Mag., 7(12)18-22, 1974

[189] Y.C.Wei and C.K.Cheng "Ratio cut partitioning for hierarchical design", IEEE Trans. on C.A.D. of ICs and Syst., 10(7)911-921, 1991

[190] Y.H.Jun, C.W.Lee, K.J.Lee and S.B.Park "Timing simulator by waveform relaxation considering the feedback effect", IEE Proc. Part G, 136(1)38-42, 1989

[191] Y.Robert and M.Tcheunte "Calcul en parallèle sur des réseaux systoliques", E.D.F. - Bulletin de la Direction des Études et Recherches, serie C, nº.1, pp.125-128, 1983

[192] Z.Navabi and M.Massouni "Investigating simulation of hardware at various levels of abstraction and timing back-annotation dataflow descriptions", SCS Jrnl., 57(5)321-332, 1991

APÊNDICES

Formato de entrada do programa ABACUS

O formato genérico para a entrada de dados do programa ABACUS é:

Definição dos elementos do circuito
FIM
Declaração dos dados da simulação

Os dados da simulação são declarados na seguinte ordem:

tempo inicial	em segundos
tempo final	em segundos
número de intervalos a examinar	
tolerância de erro admissível	
temperatura	em graus Celsius

A definição dos elementos do circuito deve obedecer às estruturas especificadas a seguir, onde todos os valores devem ser declarados nas unidades mostradas na tabela 1. Nas estruturas em questão, utilizam-se os símbolos apresentados na tabela 2. Além disso, o primeiro nó (nó_0) corresponde ao terminal ligado ao polo de referência negativa e o segundo (nó_1) ao de referência positiva.

Tabela 1 - **Definição das unidades fundamentais**

Grandeza	**Unidade**
resistência	Ohms
capacitância	Farads
corrente	Ampères
tensão	Volts
freqüência	Hertz
transcondutância	mhos/Volts
trans-resistência	Ohms/Ampères
período	segundos

Tabela 2 - Símbolos utilizados na apresentação das estruturas de entrada do programa ABACUS

Símbolo	**Significado**
RE	resistência
CA	capacitância
IS	corrente de saturação
TE	tensão
CO	corrente
FR	freqüência
PE	período
TC	transcondutância
TR	trans-resistência
GC	ganho de corrente
GT	ganho de tensão
NCm	nó cuja tensão controla a fonte (minuendo)
NCs	nó cuja tensão controla a fonte (subtraendo)
EC	elemento cuja CO controla a fonte
AM?	* amplitude máxima
TI?	* instante a partir do qual inicia-se a subida ou descida do pulso
TM?	* tempo durante o qual é mantida a amplitude máxima do pulso
RS?	* razão de subida do pulso **
RD?	* razão de descida do pulso **
SP	tempo de duração do semi-período

Observações:

*Os símbolos seguidos de ? referem-se ao primeiro ou segundo semi-períodos da forma de onda, quando a ? for substituída por 1 ou 2 respectivamente.

**Amplitude máxima dividida pelo tempo gasto pela onda para variar seu valor, de zero até a amplitude máxima (para RS) ou da amplitude máxima até zero (para RD).

Resistor:

RESxxxxxxxx	RE	nó_0	nó_1

Capacitor:

CAPxxxxxxxx	CA	nó_0	nó_1

Diodo:

DIOxxxxxxxx	IS	nó_0	nó_1

Fontes de tensão e corrente (CC ou CA):

FTIxxxxxxxx	TE	FE	nó_0	nó_1
FCIxxxxxxxx	CO	FE	nó_0	nó_1

Fonte de corrente controlada por tensão:

VCTxxxxxxxx	TC	NCm	NCs	nó_0	nó_1

Fonte de corrente controlada por corrente:

CCTxxxxxxxx	GC	EC	nó_0	nó_1

Fonte de tensão controlada por corrente:

CVTxxxxxxxx	TR	EC	nó_0	nó_1

Fonte de tensão controlada por tensão:

VVTxxxxxxxx	GT	NCm	NCs	nó_0	nó_1

Fontes quadradas de tensão e corrente:

FTQxxxxxxxx	TE	PE	nó_0	nó_1
FCQxxxxxxxx	CO	PE	nó_0	nó_1

Fontes de tensão e corrente pulsadas:

FTPxxxxxxxx	AM1	TI1	TM1	RS1	RD1	SP		
	AM2	TI2	TM2	RS2	RD2	PE	nó_0	nó_1
FCPxxxxxxxx	AM1	TI1	TM1	RS1	RD1	SP		
	AM2	TI2	TM2	RS2	RD2	PE	nó_0	nó_1

Cálculo dos tempos dos processadores

Como tempos de execução dos processadores do arranjo, tomou-se por base o número de ciclos utilizados pelo Transputer T800-20 da INMOS[083], na execução de cada uma das operações empregadas nos modelos do programa ABACUS.

Optou-se por este padrão por ser, aquele, o processador com as características mais próximas da arquitetura que se tinha em mente e por operar numa freqüência de 20MHz, compatível com a velocidade da máquina na qual seria executado o programa PSPICE, para posteriores comparações. A tabela 1 mostra o número de ciclos das operações fundamentais consideradas.

Tabela 1 - Número de ciclos por operação no T800-20

Operação	Ciclos
atribuição	003
divisão	031
exponenciação	796
escrita	005
incremento	009
leitura	005
logaritmo	690
multiplicação	018
salto	003
soma	003
subtração	006
teste	004

A partir dos dados da tabela 1 e das operações de cada modelo implementado no programa ABACUS, considerando-se os diversos caminhos possíveis nos algoritmos em questão, chegou-se a um número médio de ciclos para cada modelo e destes aos números de ciclos relativos, normalizados em função do modelo mais lento, conforme é apresentado

na tabela 2. Estes números relativos de ciclos foram utilizados no programa ABACUS para as atualizações da tabela de eventos.

Tabela 2 - Números de ciclos totais e relativos dos modelos implementados no programa ABACUS

Modelo	Total de Ciclos	Ciclos Normalizados
capacitor	1020,875	0,400
diodo	2555,875	1,000
fonte_corrente	855,875	0,335
fonte_tensão	707,830	0,277
fontel_controlel	870,875	0,341
fontel_controleV	870,875	0,341
fontel_pulsada	1101,875	0,431
fontel_quadrada	1027,875	0,402
fonteT_controlel	722,830	0,283
fonteT_controleV	722,830	0,283
fonteT_pulsada	953,830	0,373
fonteT_quadrada	879,830	0,344
resistor	919,875	0,360

Finalmente, os ciclos encontrados na tabela 2 foram convertidos em tempos de processamento, considerando-se um período de relógio de 50ns. A estes tempos foram adicionados os tempos de comunicação necessários, de forma que se pudesse estimar o desempenho da arquitetura e compará-lo com aquele do PSPICE.

Listagem do programa ABACUS

Nas páginas a seguir encontra-se a listagem completa do programa ABACUS. Nas quatro primeiras páginas mostra-se o arquivo abacus.h, com as definições das rotinas e das constantes utilizadas no referido programa. Posteriormente, vem a listagem do arquivo abacus.cpp, com as rotinas propriamente ditas.

```
/*****************/
/* ABACUS.H */
/*****************/

#ifndef _ABACUS_H
#define _ABACUS_H

#define ERR_SHORT_CCT              9
#define ERR_AMBIGUOUS_SRC_DEF      8
#define ERR_ROUTINE_PROBLEM        7
#define ERR_TOO_MANY_ELEMENTS      6
#define ERR_CANT_FIND_IN_FILE      5
#define ERR_CANT_CLOSE_FILE        4
#define ERR_CANT_OPEN_FILE         3
#define ERR_UNKNOWN_ELEMENT        2
#define ERR_OUT_OF_MEMORY          1
#define OK                         0

/* Definicao de algumas constantes globais, usadas como valores "default". */

#define ERRO        1e-6        /* Margem de erro admitida nos calculos */
#define MAXEL       11          /* Numero maximo de elementos no circuito */
#define AMOSTRAS    10          /* Numero de instantes de tempo a amostrar */
#define MAXITER     10          /* Numero maximo de iteracoes possiveis */
#define NUM_POLOS   2           // NUMero de POLOS
#define PI          3.1415926535 /* Valor da constante matematica pi */

/* Definicao de valores numericos aos nomes dos dispositivos admissiveis, para facilitar o manuseio futuro. */

#define RES   0     // Resistor
#define CAP   1     // Capacitor
#define DIO   2     // Diodo
#define FTI   3     // Fonte de Tensao Independente (DC ou senoidal)
#define FTQ   4     // Fonte de Tensao Quadrada
#define FTP   5     // Fonte de Tensao Linear por Partes
#define CVT   6     // Fonte de Tensao Controlada por Corrente
#define VVT   7     // Fonte de Tensao Controlada por Tensao
#define FCI   8     // Fonte de Corrente Independente (DC ou senoidal)
#define FCQ   9     // Fonte de Corrente Quadrada
#define FCP   10    // Fonte de Corrente Linear por Partes
#define CCT   11    // Fonte de Corrente Controlada por Corrente
#define VCT   12    // Fonte de Corrente Controlada por Tensao

typedef struct Analise Analise;
typedef struct Buffer Buffer;
typedef struct Entrada Entrada;
typedef struct Memory Memory;
typedef struct Processadores Processadores;
typedef struct Saida Saida;
typedef struct Fonte_Pulsada Fonte_Pulsada;

struct Fonte_Pulsada
{
        float amplitude[2];
        float t_zero[2];
        float t_max[2];
        float subida[2];
        float descida[2];
        float periodo[2];
};
```

```
struct Entrada
{
        char elemento[11];
        float valor;
        union {float I_sat; float Freq; int no_Cntrl[2]; char elmto_Cntrl[11]; float Per; Fonte_Pulsada fip;};
        unsigned short int no[NUM_POLOS];
};

struct Analise
{
        float erro;
        float t_inicio;
        float t_fim;
        float delta_t;
        int amostras;
        float temperatura;
};

struct Buffer
{
   float tensao[2];        // O indice 0 indica o valor calculado na iteracao
   float corrente[2];      // anterior e o 1 aquele calculado na iteracao atual.
   int fonteTA;
   int fonteQQ;
   int polo;
   int no;
   int processador;
};

struct Memory
{
        float tensao_A;
        float media_tensao_A;
        float grandeza_tensao_A;
        float tensao_B;
        float media_tensao_B;
        float grandeza_tensao_B;
        float corrente;
        float media_corrente;
        float grandeza_corrente;
        int num_iteracoes;
        int flag;
        float sobe[2];
        float desce[2];
        int num_per;
        float tau;
        float Io;
        float Vf;
        float Vfo;
        float transicao;
        float carga;
};

struct Processadores
{
        int ula;
        float valor;
        union {float I_sat; float Freq; int no_Cntrl[2]; int elmto_Cntrl; float Per; Fonte_Pulsada fip;};
        Buffer fila_saida[NUM_POLOS];
        int num_entradas[NUM_POLOS];
        Buffer *fila_entrada[NUM_POLOS][10];
```

```
        Entrada *pte;
        Saida *pts;
        Memory memoria;
};

struct Saida
{
        int elemento;
        float corrente[MAXITER];
        float tensao_A[MAXITER];
        float tensao_B[MAXITER];
};

/* Declaracao das funcoes  de  entrada e saida  e daquelas de controle geral da simulacao, chamadas
diretamente por main. */

int prolegomenos (void);
int le_entrada (void);
void imprime_entrada (void);
void controlador (void);

/* Declaracao das funcoes de apoio ao controle da simulacao. */

void verifica_entrada (void);
float busca_tempo (int tipo_ula);
int pesquisa_tabela (void);
void limpa_tabela (void);
int processador (int i, float t);
int gerente (int flag);
void calcula_medias (float t);
void carrega_saida (float t);
int e_fonte(int i);
int e_aterrado(int i);
int e_ligado(int i, int nivel, int *elementos);
int ligados(int c1, int c2);
float *verifica_impedancia (int i, float *impedancias);
void atribui_cargas (float *impedancias);
int e_fonte_de_tensao (int i);
int e_fonte_de_corrente (int i);
int e_fonte_TA (int i);
int testa_terra (int i);
int em_curto(int c1, int c2, int polo1, int *polo);
void atualiza_capacitores (void);
int inicio (float t);
int transicao (int i, float t);
float carga_armazenada (int i, float t);
float busca_fonte_noCAP (int i, float *V);
int fonteTA_RECADI (int i, float *V, float Vab);

#define tTol(t)   ((int) (((t) - dados.t_inicio)/dados.delta_t))

/* Declaracao das funcoes com os modelos dos dispositivos disponiveis. */

int resistor (int i, float t);
int capacitor (int i, float t);
int diodo (int i);
int fonte_tensao (int i, float t);
int fonteT_quadrada (int i, float t);
int fonteT_pulsada (int i, float t);
int fonteT_controleI (int i);
int fonteT_controleV (int i);
```

```
int fonte_corrente (int i, float t);
int fontei_quadrada (int i, float t);
int fontei_pulsada (int i, float t);
int fontei_controleV (int i);
int fontei_controlei (int i);

/* Declaracao de funcoes para suporte matematico aos modelos. */

float *media_tensoes (int i);
float soma_correntes (int i);
int tramites_finais (float Va, float Vb, float Iab, int i);

#define e_terra(proc, polo) (mph[(proc)].fila_saida[(polo)].no == 0)

/* Declaracao de outras funcoes de apoio. */

int testa_elemento (int i);
void ordena_circuito (int *ordem);
void carrega_arranjo (int *ordem);
void imprime_saida (int *ordem);
int testa_conectividade (int c1, int n1, int c2);

#endif
```

```
/*********************/
/* ABACUS.CPP */
/*********************/

#include <time.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <fstream.h>
#include <string.h>
#include <dos.h>
#include "abacus.h"

Entrada circuito[MAXEL];
Analise dados;
Processadores mph[MAXEL];
Saida solucoes[MAXEL];

/* Declaracao das variaveis globais. */

int cntrl_conv = 0;
int num_elementos = 0;
float tabela_de_eventos[MAXEL];
int comunica_proc_proc[MAXEL][MAXEL];
int comunica_proc_host[MAXEL];
int comunica_host_proc[MAXEL];
int uso_proc[MAXEL][MAXITER];
float tempo_proc[MAXEL];

/* Fim dos preliminares, aqui inicia o programa propriamente dito. */

void main ()
    {
    num_elementos = prolegomenos ();
    if (num_elementos == -1) goto fim;
    system ("cls");
    num_elementos = le_entrada ();
    if (num_elementos == -1) goto fim;
    system ("cls");
    imprime_entrada ();
    system ("cls");
    controlador ();
fim:
    system ("cls");
    exit (OK);
    }

int prolegomenos ()
    {
    char resposta;
    system ("cls");
    printf ("\n*********************************************************************************");
    printf ("\n**                    Voce entrou no programa ABACUS.                          **");
    printf ("\n**        Por favor, leia com atencao as instrucoes a seguir.                  **");
    printf ("\n*********************************************************************************");
    printf ("\n\n\007Defina seu circuito num arquivo chamado ABACUS.IN.");
    printf ("\nVoce pode definir ate\' %d elementos num circuito. Os nomes dos elementos podem", MAXEL);
    printf ("\nter ate\' 10 caracteres e devem iniciar, respectivamente, pelos tres caracteres");
    printf ("\nseguintes: \n\t\tRES\tresistor\n\t\tCAP\tcapacitor\n\t\tDIO\tdiodo\n");
    printf ("\t\tFTI\tfonte de tensao independente\n");
```

```
    printf ("\t\tFTQ\tfonte de tensao quadrada\n");
    printf ("\t\tFTP\tfonte de tensao linear por partes\n");
    printf ("\t\tCVT\tfonte de tensao controlada por corrente\n");
    printf ("\t\tVVT\tfonte de tensao controlada por tensao\n");
    printf ("\t\tFCI\tfonte de corrente independente\n");
    printf ("\t\tFCQ\tfonte de corrente quadrada\n");
    printf ("\t\tFCP\tfonte de corrente linear por partes\n");
    printf ("\t\tVCT\tfonte de corrente controlada por tensao\n");
    printf ("\t\tCCT\tfonte de corrente controlada por corrente\n\007");
    sleep (15);
    system ("cls");
    printf ("\nO circuito deve ser definido da seguinte forma:\007");
    printf ("\n\t- declare o nome de cada elemento,  seu  valor e os nos aos quais esta'");
    printf ("\n\t  conectado;");
    printf ("\n\t- se o elemento for um diodo ou uma fonte controlada por tensao,  entre");
    printf ("\n\t  o valor e os nos, voce deve declarar o valor da corrente de saturacao");
    printf ("\n\t  ou do no' cuja tensao controla a fonte;");
    printf ("\n\t- se o elemento for uma fonte controlada por corrente,  entre o valor e");
    printf ("\n\t  os nos voce deve  declarar o elemento cuja corrente controla a fonte;");
    printf ("\n\t- se o elemento for uma fonte independente, entre o valor e os nos, vo-");
    printf ("\n\t  ce deve declarar a respectiva frequencia de operacao, ou seu periodo,");
    printf ("\n\t  caso contrario, declare zero neste campo;");
    printf ("\n\t- se o elemento  for uma  fonte linear por partes,  voce deve declarar,");
    printf ("\n\t  para cada semiperiodo, as amplitudes,  os tempos para o inicio da su-");
    printf ("\n\t  bida,  os tempos durante os quais a amplitude declarada deve ser man-");
    printf ("\n\t  tida, as respectivas razoes  de subida e de descida,  o semiperiodo e");
    printf ("\n\t  o periodo da fonte;");
    printf ("\n\t- cada item  declarado  deve  ser  separado por pelo menos um espaco em");
    printf ("\n\t  branco;");
    printf ("\n\t- apos declarar todos os elementos de seu circuito, escreva FIM;");
    printf ("\n\t- em seguida declare os tempos inicial e final da simulacao;");
    printf ("\n\t- o numero de instantes a serem amostrados;");
    printf ("\n\t- a margem de erro admissivel;");
    printf ("\n\t- e a temperatura a ser considerada na simulacao.\007");
    sleep (15);
    system ("cls");
    printf ("\n\nAqui deveriamos ter uma bela descricao deste programa.");
    printf ("\nMas nao temos, mesmo assim voce quer continuar? (S/N) \007");
LP0: fflush(stdin);
    resposta = getchar();
    if (resposta == 'n' || resposta == 'N')
        return (-1);
    else if (resposta == 's' || resposta == 'S')
            return (0);
    else
        {
        printf ("\nPor favor responda S para sim e N para nao ...  \007\007");
        goto LP0;
        }
    }

int le_entrada ()
   {
   ifstream entrada("abacus.in");
   if (!entrada)
     {
     fprintf (stderr, "\nNao encontrei o arquivo de entrada.\n\007\007");
     exit (ERR_CANT_FIND_IN_FILE);
     }
   int i=0;
   while (1)
```

```
        {
        entrada >> circuito[i].elemento;
        if (!strcmp (circuito[i].elemento, "FIM")) break;
        if (circuito[i].elemento[0] == 'D' || circuito[i].elemento[0] == 'd')
          entrada >> circuito[i].valor >> circuito[i].I_sat;
        else if (circuito[i].elemento[0]=='F' || circuito[i].elemento[0]=='f')
                {
                if (circuito[i].elemento[2]=='I' || circuito[i].elemento[2]=='i')
                  entrada >> circuito[i].valor >> circuito[i].Freq;
                else if (circuito[i].elemento[2]=='Q' || circuito[i].elemento[2]=='q')
                  entrada >> circuito[i].valor >> circuito[i].Per;
                else if (circuito[i].elemento[2]=='P' || circuito[i].elemento[2]=='p')
                  for (int j=0 ; j<2 ; j++)
                        {
                        entrada >> circuito[i].flp.amplitude[j];
                        entrada >> circuito[i].flp.t_zero[j];
                        entrada >> circuito[i].flp.t_max[j];
                        entrada >> circuito[i].flp.subida[j];
                        entrada >> circuito[i].flp.descida[j];
                        entrada >> circuito[i].flp.periodo[j];
                        }
                }
        else if ((circuito[i].elemento[0]=='C' && circuito[i].elemento[1]!='A' && circuito[i].elemento[1]!='a') ||
                  (circuito[i].elemento[0]=='c' && circuito[i].elemento[1]!='A' && circuito[i].elemento[1]!='a') )
                  entrada >> circuito[i].valor >> circuito[i].elmto_Cntrl;
        else if (circuito[i].elemento[0]=='V' || circuito[i].elemento[0]=='v')
               entrada >> circuito[i].valor >> circuito[i].no_Cntrl[0] >> circuito[i].no_Cntrl[1];
        else entrada >> circuito[i].valor;
        entrada >> circuito[i].no[0] >> circuito[i].no[1];
        i++;
        if (i >= MAXEL)
          {
          fprintf (stderr, "\nLamento, mas o numero maximo de elementos");
          fprintf (stderr, " permitidos e' %d.\n\007\007", MAXEL);
          exit (ERR_TOO_MANY_ELEMENTS);
          }
        }
        entrada >> dados.t_inicio >> dados.t_fim >> dados.amostras;
        entrada >> dados.erro >> dados.temperatura;
        if (dados.amostras < 0 || dados.amostras > AMOSTRAS)
          {
          printf ("\nVou utilizar o numero de amostras default: %d.", AMOSTRAS);
          printf ("\nVoce quer continuar? (S/N)  \007");
          while (1)
                {
                fflush (stdin);
                char resposta = getchar ();
                if (resposta == 'n' || resposta == 'N')
                {
printf ("\nVoce pode aumentar o numero maximo de amostras alterando o valor da constante AMOSTRAS.\n");
return (-1);
                }
                 else if (resposta == 's' || resposta == 'S')
                        {
                        dados.amostras = AMOSTRAS;

                        break;
                        }
                 else printf ("\nPor favor, responda S para sim ou N para nao ...  \007\007");
                 }
          }
```

```
        if (dados.erro <= 0 || dados.erro >= 1)
          {
          printf ("\nVou utilizar o valor do erro default: %g.", ERRO);
          printf ("\nVoce quer continuar? (S/N)  \007");
          while (1)
                {
                fflush (stdin);
                char resposta = getchar ();
                if (resposta == 'n' || resposta == 'N')
                  {
                  printf ("\nVoce pode alterar o valor do erro alterando o valor da constante ERRO.\n");
                  return (-1);
                  }
                else if (resposta == 's' || resposta == 'S')
                     {
                     dados.erro = ERRO;
                     break;
                     }
                else printf ("\nPor favor, responda S para sim ou N para nao ...  \007\007");
                }
          }
         dados.delta_t = (dados.t_fim - dados.t_inicio) / dados.amostras;
   return i;
   }

void imprime_entrada ()
    {
    int j, elemento;
    char resposta;
    printf ("\n\nVoce quer ver os valores defaults? (S/N)  ");
LP1: fflush(stdin);
    resposta =  getchar();
    if (resposta == 's' || resposta == 'S')
      {
      printf ("\n\nO valor do erro e\' %g.\n", ERRO);
        printf ("O numero de amostras e\' %d.\n", AMOSTRAS);
        sleep (5);
        }
    else if (resposta != 'n' && resposta != 'N')
        {
        printf ("\n\nPor favor, responda S para sim ou N para nao.  \007\007");
        goto LP1;
        }
    system ("cls");
    printf ("\n\nVoce quer conferir os valores lidos? (S/N)  ");
LP2: fflush(stdin);
    resposta = getchar();
    if (resposta == 's' || resposta == 'S')
      {
      printf ("\nOs valores abaixo estao dispostos em tres colunas ");
      printf ("que correspondem, respectiva-\nmente, ao ");
      printf ("componente lido, seu valor e seus dois terminais.\n");
        for (int i=0 ; i<num_elementos ; i++)
            {
            j = i;
            elemento = testa_elemento (i);
            printf ("\nO %3d§ elemento do circuito lido e\': ", ++j);
            printf ("     %10s", circuito[i].elemento);
            if (elemento!=FTP && elemento!=FCP)
              printf ("     %g", circuito[i].valor);
            else printf ("          ");
```

```
            for( int c_polos = 0; c_polos < NUM_POLOS; c_polos++)
                printf ("     %2u", circuito[i].no[c_polos]);
            if (elemento == DIO)
                printf ("\n\t\tcorrente de saturacao: %g", circuito[i].I_sat);
            else if (elemento == VVT || elemento == VCT)
                printf ("\n\t\tnos de controle da fonte: %d - %d", circuito[i].no_Cntrl[0], circuito[i].no_Cntrl[1]);
            else if (elemento == CCT || elemento == CVT)
                printf ("\n\t\tramo de controle da fonte: %10s", circuito[i].elmto_Cntrl);
            else if (elemento == FTI || elemento == FCI)
                printf ("\n\t\tfrequencia da fonte: %g", circuito[i].Freq);
            else if (elemento == FTQ || elemento == FCQ)
                printf ("\n\t\tperiodo da fonte: %g", circuito[i].Per);
            else if (elemento == FTP || elemento == FCP)
                for (int k=0 ; k<2 ; k++)
                   {
            printf ("\n\t\tamplitude maxima do %3d§ semiperiodo: %g", k+1, circuito[i].flp.amplitude[k]);
            printf ("\n\t\ttempo inicial do pulso no %3d§ semiperiodo: %g", k+1, circuito[i].flp.t_zero[k]);
            printf ("\n\t\ttempo de duracao do pulso no %3d§ semiperiodo: %g", k+1, circuito[i].flp.t_max[k]);
            printf ("\n\t\tangulo de subida do pulso no %3d§ semiperiodo: %g", k+1, circuito[i].flp.subida[k]);
            printf ("\n\t\tangulo de descida do pulso no %3d§ semiperiodo: %g", k+1, circuito[i].flp.descida[k]);
            if (k)   printf ("\n\t\tduracao do periodo total: %g", circuito[i].flp.periodo[k]);
            else  printf ("\n\t\tduracao do %3d§ semiperiodo: %g", k+1, circuito[i].flp.periodo[k]);
                    }
           }
        printf ("\n\nCom relacao \'a analise, os dados disponiveis sao:\n");
        printf ("\ninstante de tempo inicial = %g", dados.t_inicio);
        printf ("\ninstante de tempo final = %g", dados.t_fim);
        printf ("\ntotal de instantes a amostrar = %d", dados.amostras);
        printf ("\ntolerancia maxima de erro = %g", dados.erro);
          printf ("\ntemperatura ambiente = %g", dados.temperatura);
        printf ("\n\n");
          sleep (5);
        }
      else if (resposta != 'n' && resposta != 'N')
            {
              printf ("\n\nPor favor, responda S para sim ou N para nao.  \007\007");
            goto LP2;
              }
      system ("cls");
      printf ("\n\nVoce quer prosseguir com a simulacao? (S/N)  ");
LP3: fflush (stdin);
      resposta = getchar ();
      if (resposta == 'n' || resposta == 'N')
           exit (OK);
      else if (resposta == 's' || resposta == 'S')
               verifica_entrada ();
      else {
            printf ("\n\nPor favor, responda S para sim ou N para nao.  \007\007");
            goto LP3;
            }
      return;
      }

int testa_conectividade (int c1, int n1, int c2)
     {
     for (int i=0 ; i<num_elementos ; i++)
           for (int j=0 ; j<NUM_POLOS ; j++)
              if (i!=c1 && i!=c2 && circuito[c1].no[n1]==circuito[i].no[j])
                 return !0;
     return 0;
     }
```

```
void verifica_entrada ()
    {
    int conectividade=!0, erro=0;
    for (int i=0 ; i<num_elementos-1 ; i++)
        for (int j=i+1 ; j<num_elementos ; j++)
            {
// TESTA SE EXISTEM FONTES DE TENSAO LIGADAS EM PARALELO A UMA MESMA REFERENCIA.
// NESTE CASO A TENSAO SERA' INDETERMINADA E O PROGRAMA DEVE SER ABORTADO.
            for (int fonte1=FTI ; fonte1<=VVT ; fonte1++)
                for (int fonte2=FTI ; fonte2<=VVT ; fonte2++)
                    if (testa_elemento(i) == fonte1 && testa_elemento(j) == fonte2)
                        if ((circuito[i].no[0] == circuito[j].no[0] && circuito[i].no[1] == circuito[j].no[1]) ||
                            (circuito[i].no[0] == circuito[j].no[1] && circuito[i].no[1] == circuito[j].no[0]))
                            {
        fprintf (stderr,"\nOs elementos %s e %s estao em curto.", circuito[i].elemento, circuito[j].elemento);
        erro = 1;
                            }
// TESTA SE EXISTEM FONTES DE CORRENTE COM UM NO' EM COMUM E O OUTRO NAO, E SE AO NO'
// COMUM NAO ESTA' CONECTADO NENHUM OUTRO ELEMENTO. NESTE CASO AS FONTES ESTAO EM
// SERIE NUM DETERMINADO RAMO DO CIRCUITO E O PROGRAMA DEVE SER ABORTADO.
            for (fonte1=FCI ; fonte1<=VCT ; fonte1++)
                for (fonte2=FCI ; fonte2<=VCT ; fonte2++)
                    if (testa_elemento(i) == fonte1 && testa_elemento(j) == fonte2)
                        {
                        if ((circuito[i].no[0] == circuito[j].no[0] && circuito[i].no[1] == circuito[j].no[1]) ||
                            (circuito[i].no[0] == circuito[j].no[1] && circuito[i].no[1] == circuito[j].no[0]))
                            {
                            conectividade = testa_conectividade (i, 0, j);
                            if (!conectividade)
                              conectividade = testa_conectividade (i, 1, j);
                            }
                        else if (circuito[i].no[0] == circuito[j].no[0] && circuito[i].no[1] != circuito[j].no[1])
                                  conectividade = testa_conectividade (i, 0, j);
                        else if (circuito[i].no[0] == circuito[j].no[1] && circuito[i].no[1] != circuito[j].no[0])
                                  conectividade = testa_conectividade (i, 0, j);
                        else if (circuito[i].no[1] == circuito[j].no[0] && circuito[i].no[0] != circuito[j].no[1])
                                  conectividade = testa_conectividade (i, 1, j);
                        else if (circuito[i].no[1] == circuito[j].no[1] && circuito[i].no[0] != circuito[j].no[0])
                                  conectividade = testa_conectividade (i, 1, j);
                          if (!conectividade)
                          {
        fprintf (stderr, "\nElementos %s e %s erroneamente em serie.", circuito[i].elemento, circuito[j].elemento);
        erro = 1;
                          }
                        }

// TESTA SE EXISTE ALGUM CAPACITOR CONECTADO, EXCLUSIVAMENTE, ENTRE UMA FONTE DE
// TENSAO ATERRADA E O TERRA DO CIRCUITO.
            if (testa_elemento (i) == CAP)
                {
                if (e_terra (i, 0))
                  if (circuito[i].no[1]==circuito[j].no[0] || circuito[i].no[1]==circuito[j].no[1])
                    if (e_fonte_TA (j))
                        conectividade = testa_conectividade (i, 0, j);
                else if (e_terra (i, 1))
                        if (circuito[i].no[0]==circuito[j].no[0] || circuito[i].no[0]==circuito[j].no[1])
                          if (e_fonte_TA (j))
                            conectividade = testa_conectividade (i, 1, j);
                if (!conectividade)
                  {
                  fprintf (stderr, "\nA ligacao do capacitor %s requer a especificacao ", circuito[i].elemento);
```

```
                    fprintf (stderr, "\nda resistencia interna da fonte %s.", circuito[j].elemento);
                    erro = 1;
                    }
                  }
            }
    if (erro)
        {
        fprintf (stderr, "\nPrograma abortado.\n");
        exit (ERR_AMBIGUOUS_SRC_DEF);
        }
    else return;
    }

void controlador ()
    {
    int i=0, flow, flag;
    float t=dados.t_inicio;
    int *ordem;
    if(!(ordem = new int[num_elementos]))
     {
     fprintf(stderr,"\nMemoria insuficiente.\n");
     exit(ERR_OUT_OF_MEMORY);
     }
    printf ("\n\nProcessando ... \n\n");
    ordena_circuito (ordem);
    carrega_arranjo (ordem);
    while (t <= (dados.t_fim + (dados.delta_t / 2)))
          {
          flow = 1;
          for(cntrl_conv = 0; cntrl_conv < num_elementos; cntrl_conv++)
            {
            mph[cntrl_conv].memoria.flag = -1;
            comunica_host_proc[cntrl_conv]++;
            }
          while (flow == 1)
                {
                printf (".");
                i = pesquisa_tabela ();
                uso_proc[i][(int) ((t/dados.delta_t)+0.5)]++;
                tempo_proc[i]++;
                flag = processador (i, t);
                comunica_proc_host[i]++;
                flow = gerente (flag);
                comunica_host_proc[i]++;
                }
          carrega_saida (t);
          t += dados.delta_t;
          calcula_medias (t);
          limpa_tabela ();
          atualiza_capacitores ();
          printf ("T");
          }
     imprime_saida (ordem);
     return;
     }

 void limpa_tabela ()
     {
     for (int i=0; i<num_elementos; i++)
          tabela_de_eventos[i] = 0.0;
     }
```

```
void ordena_circuito(int *ordem)
    {
    int c_ordem = 0, *elemento;
    if (! (elemento = new int[num_elementos]))
         {
         fprintf (stderr, "\nMemoria insuficiente.\n");
         exit (ERR_OUT_OF_MEMORY);
         }
    for (int i = 0; i < num_elementos; i++)
          if (e_fonte(i) && e_aterrado(i))
             {
             elemento[i] = 0;
             ordem[c_ordem++] = i;
             }
          else elemento[i] = -1;
    int nivel = 1;
    do
         {
         for (i = 0; i < num_elementos; i++)
             if (elemento[i] == -1 && e_ligado(i,nivel-1,elemento))
                {
                elemento[i] = nivel;
                ordem[c_ordem++] = i;
                }
         nivel++;
         }
    while(num_elementos > c_ordem);
    }

int e_fonte(int i)
   {
   switch (testa_elemento(i))
         {
         case FTI: return (1);
         case FTQ: return (1);
         case FTP: return (1);
         case VVT: return (1);
         case CVT: return (1);
         case FCI: return (1);
         case FCQ: return (1);
         case FCP: return (1);
         case CCT: return (1);
         case VCT: return (1);
         default : return (0);
         }
   }

int e_aterrado(int i)
   {
   for (int c_polos = 0; c_polos < NUM_POLOS; c_polos++)
         if (circuito[i].no[c_polos] == 0)
            return !0;
   return 0;
   }

int e_ligado(int i, int nivel, int *elemento)
   {
   for (int c = 0; c < num_elementos; c++)
         if (elemento[c] == nivel && ligados(i, c))
            return 1;
   return 0;   }
```

```
int ligados(int c1, int c2)
   {
   int polo;
   for (int c_polo = 0; c_polo < NUM_POLOS; c_polo++)
        if (circuito[c1].no[c_polo] != 0 && em_curto(c1, c2, c_polo, &polo))
          return !0;
   return 0;
   }

#define o(i)      ordem[(i)]

void carrega_arranjo (int *ordem)
    {
    for (int i=0 ; i<num_elementos ; i++)
         {
         mph[i].ula = testa_elemento (o(i));
        comunica_host_proc[i]++;
         if (mph[i].ula != FTP && mph[i].ula != FCP)
         {
           mph[i].valor = circuito[o(i)].valor;
           comunica_host_proc[i]++;
         }
         if (mph[i].ula == DIO)
         {
           mph[i].I_sat = circuito[o(i)].I_sat;
           comunica_host_proc[i]++;
         }
         else if (mph[i].ula == FTI || mph[i].ula == FCI)
             {
             mph[i].Freq = circuito[o(i)].Freq;
             comunica_host_proc[i]++;
             }
         else if (mph[i].ula == FTQ || mph[i].ula == FCQ)
          {
             mph[i].Per = circuito[o(i)].Per;
             comunica_host_proc[i]++;
             }

         else if (mph[i].ula == CCT || mph[i].ula == CVT)
             {
             for (int j=0 ; j<num_elementos ; j++)
                  if (strcmp (circuito[o(i)].elemento, circuito[j].elemento))
                     {
                     for (int k=0 ; k<num_elementos ; k++)
                          if (o(k) == j)
                             {
                             mph[i].elmto_Cntrl = k;
                      comunica_host_proc[i]++;
                             break;
                             }
                     break;
                     }
             }
         else if (mph[i].ula == FTP || mph[i].ula == FCP)
                 {
                 mph[i].memoria.num_per = 0;
             comunica_host_proc[i]++;
                 for (int j=0 ; j<2 ; j++ )
                     {
                     mph[i].flp.amplitude[j] = circuito[o(i)].flp.amplitude[j];
                     mph[i].flp.t_zero[j] = circuito[o(i)].flp.t_zero[j];
```

```
                    mph[i].flp.t_max[j] = circuito[o(i)].flp.t_max[j];
                    mph[i].flp.subida[j] = circuito[o(i)].flp.subida[j];
                    mph[i].flp.descida[j] = circuito[o(i)].flp.descida[j];
                    mph[i].flp.periodo[j] = circuito[o(i)].flp.periodo[j];
                    if (!mph[i].flp.subida[j])
                        mph[i].memoria.sobe[j] = 0;
                else mph[i].memoria.sobe[j] = mph[i].flp.amplitude[j] / mph[i].flp.subida[j];
                    if (!mph[i].flp.descida[j])
                        mph[i].memoria.desce[j] = 0;
                    else mph[i].memoria.desce[j] = mph[i].flp.amplitude[j] / mph[i].flp.descida[j];
                comunica_host_proc[i] += 8;
                    }
                 }
        else if (mph[i].ula == VCT || mph[i].ula == VVT)
                {
                mph[i].no_Cntrl[0] = circuito[o(i)].no_Cntrl[0];
                mph[i].no_Cntrl[1] = circuito[o(i)].no_Cntrl[1];
            comunica_host_proc[i] += 2;
                }
        solucoes[i].elemento = mph[i].ula;
        solucoes[i].corrente[0] = 0.0;
        solucoes[i].tensao_A[0] = 0.0;
        solucoes[i].tensao_B[0] = 0.0;
        for (int j = 0; j < NUM_POLOS; j++)
            {
            mph[i].fila_saida[j].processador = i;
            mph[i].fila_saida[j].no = circuito[o(i)].no[j];
            mph[i].fila_saida[j].fonteTA = e_fonte_TA (o(i));
            mph[i].fila_saida[j].fonteQQ = e_fonte (o(i));
            mph[i].fila_saida[j].polo = j;
            mph[i].num_entradas[j] = 0;
            comunica_host_proc[i] += 6;
            int j2;
            for (int i2 = 0; i2 < num_elementos; i2++)
                if (i2 != i && em_curto(o(i), o(i2), j, &j2))
                    {
                    mph[i].fila_entrada[j][mph[i].num_entradas[j]++] = &mph[i2].fila_saida[j2];
                    comunica_host_proc[i]++;
                    }
            }
        mph[i].pte = circuito + o(i);
        mph[i].pts = &solucoes[i];
        mph[i].memoria.num_iteracoes = 1;
        mph[i].memoria.tensao_A = 0.0;
        mph[i].memoria.media_tensao_A = 0.0;
        mph[i].memoria.tensao_B = 0.0;
        mph[i].memoria.media_tensao_B = 0.0;
        mph[i].memoria.corrente = 0.0;
        mph[i].memoria.media_corrente = 0.0;
        comunica_host_proc[i] += 7;
        }
    }

int e_fonte_de_tensao (int i)
    {
    switch (mph[i].ula)
            {
            case FTI: return !0;
            case FTQ: return !0;
            case FTP: return !0;
            case VVT: return !0;
```

```
            case CVT: return !0;
            default: return 0;
            }
    }

int e_fonte_de_corrente (int i)
    {
    switch (mph[i].ula)
            {
            case FCI: return !0;
            case FCQ: return !0;
            case FCP: return !0;
            case VCT: return !0;
            case CCT: return !0;
            default: return 0;
            }
    }


int e_fonte_TA (int i)
    {
    switch (testa_elemento(i))
          {
          case FTI: return (testa_terra(i));
          case FTQ: return (testa_terra(i));
          case FTP: return (testa_terra(i));
          default : return (0);
          }
    }

int testa_terra (int i)
    {
    if (e_terra (i,0) || e_terra (i,1))
          return 1;
    return 0;
    }

int em_curto(int c1, int c2, int polo1, int *polo2)
    {
    for (int j = 0; j < NUM_POLOS; j++)
          if (circuito[c1].no[polo1] == circuito[c2].no[j])
            {
            *polo2 = j;
            return !0;
            }
    return 0;
    }

void imprime_saida (int *ordem)
     {
     FILE     *out;
     if((out = fopen("abacus.out","w")) == NULL)
       {
       fprintf(stderr, "Nao pude abrir o arquivo de saida.\n");
       exit(ERR_CANT_OPEN_FILE);
       }
     fprintf (out, "\n\t\t****** Arquivo de entrada  ABACUS.IN ******\n");
     for (int i=0; i<num_elementos; i++)
          {
          fprintf (out, "\n\t\t\t%s", circuito[o(i)].elemento);
          if (mph[i].ula != FTP && mph[i].ula != FCP)
```

```
        fprintf (out, "\t%g", circuito[o(i)].valor);
      else fprintf (out, "\t\t");
      if (mph[i].ula == DIO)
        fprintf (out, "\t%g", circuito[o(i)].I_sat);
      else if (mph[i].ula == FTI || mph[i].ula == FCI)
              fprintf (out, "\t%g", circuito[o(i)].Freq);
      else if (mph[i].ula == FTQ || mph[i].ula == FCQ)
              fprintf (out, "\t%g", circuito[o(i)].Per);
      else if (mph[i].ula == CCT || mph[i].ula == CVT)
              fprintf (out, "\t%10s", circuito[o(i)].elmto_Cntrl);
      else if (mph[i].ula == FTP || mph[i].ula == FCP)
              fprintf (out, "\t\t");
      else if (mph[i].ula == VCT || mph[i].ula == VVT)
              fprintf (out, "\t%d\t%d", circuito[o(i)].no_Cntrl[0], circuito[o(i)].no_Cntrl[1]);
      fprintf (out, "\t%d", circuito[o(i)].no[0]);
      fprintf (out, "\t%d", circuito[o(i)].no[1]);
      if (mph[i].ula == FTP || mph[i].ula == FCP)
        for (int j=0 ; j<2 ; j++)
              {
              if (j)   fprintf (out, "\n\t\t\tno periodo:\n");
              else fprintf (out, "\n\t\t\tno semiperiodo:\n");
              fprintf (out, "\t\t\t\t\t%g", circuito[o(i)].flp.amplitude[j]);
              fprintf (out, "\t%g", circuito[o(i)].flp.t_zero[j]);
              fprintf (out, "\t%g", circuito[o(i)].flp.t_max[j]);
              fprintf (out, "\t%g", circuito[o(i)].flp.subida[j]);
              fprintf (out, "\t%g", circuito[o(i)].flp.descida[j]);
              fprintf (out, "\t%g", circuito[o(i)].flp.periodo[j]);
              }
    }
  fprintf (out, "\n\t\t\t%g", dados.t_inicio);
  fprintf (out, "\n\t\t\t%g", dados.t_fim);
  fprintf (out, "\n\t\t\t%d", dados.amostras);
  fprintf (out, "\n\t\t\t%g", dados.erro);
  fprintf (out, "\n\t\t\t%g\n\n", dados.temperatura);
  fprintf (out, "\n\n\t\t***** Resultados da Simulacao *****\n\n");
  fprintf (out, "\tprocessador (tipo)\t\t  corrente\t  tensao no polo 1\t  tensao no polo 0");
  for ( i=0 ; i<=dados.amostras ; i++ )
      {
      fprintf(out, "\nt: %g:\n", ((float) (dados.t_inicio + i*dados.delta_t)));
      for (int j=0 ; j<num_elementos ; j++ )
          {
          fprintf(out,"\tmph[%d] (%d): %5s", j, mph[j].ula, mph[j].pte->elemento);
          fprintf(out,"\t\t%10g", solucoes[j].corrente[i]);
          fprintf(out,"\t\t%10g", solucoes[j].tensao_A[i]);
          fprintf(out,"\t\t%10g", solucoes[j].tensao_B[i]);
          fprintf(out,"\n");
          }
      }
  fprintf(out,"\n\n\n\t\t***** Estatisticas de utilizacao *****\n\n");
  fprintf(out,"\n\t\tO processador\t\tfoi acionado\t\tvezes no instante\n");
  for (int k=0 ; k<num_elementos ; k++)
      {
      float fator;
      switch (mph[k].ula)     {
              case (FTI): fator =  708; break;
              case (FTQ): fator =  880; break;
              case (RES): fator =  920; break;
              case (CAP): fator = 1020; break;
              case (DIO): fator = 2555; break;
              default: fator = 0;
              }
```

```
        for (int l=0 ; l<=dados.amostras ; l++)
            fprintf(out,"\t\t\t\t%s\t\t\t%d\t\t\t   %g\n", mph[k].pte->elemento, uso_proc[k][l], ((float) (dados.t_inicio +
l*dados.delta_t)));
        fprintf(out,"\t\t\tO tempo de processamento de %s foi de %g us\n", mph[k].pte->elemento,
(tempo_proc[k]*0.05*fator+(dados.amostras*1.25)));
        }
    fprintf(out,"\n\n\n\t\t***** Estatisticas de comunicacao *****\n\n");
    fprintf(out,"\n\t\tO gerente transferiu        informacoes para o processador\n");
    for (k=0 ; k<num_elementos ; k++)
        if (comunica_host_proc[k])
            fprintf(out,"\t\t\t\t\t%d\t\t\t\t\t%s\n", comunica_host_proc[k], mph[k].pte->elemento);
    fprintf(out,"\n\n\t\tO processador\t\ttransferiu\tinformacoes para o gerente\n");
    for (k=0 ; k<num_elementos ; k++)
       if (comunica_proc_host[k])
            fprintf(out,"\t\t\t\t%s\t\t   %d\n", mph[k].pte->elemento, comunica_proc_host[k]);
    fprintf(out,"\n\n\t\tO processador\t\ttransferiu\tinformacoes para o processador\n");
    for (k=0 ; k<num_elementos ; k++)
        for (int l=0 ; l<num_elementos ; l++)
            if (comunica_proc_proc[l][k])
fprintf(out,"\t\t\t\t%s\t\t   %d\t\t\t\t\t%s\n", mph[k].pte->elemento, comunica_proc_proc[l][k], mph[l].pte->elemento);
        if (fclose(out))
          {
          fprintf (stderr, "Nao pude fechar o arquivo de saida.\n");
          exit (ERR_CANT_CLOSE_FILE);
          }
    }

int testa_elemento (int i)
   {
   switch (circuito[i].elemento[0]) {
        case 'R': return (RES);
        case 'r': return (RES);
          case 'C': switch (circuito[i].elemento[1]) {
                    case 'A': return (CAP);
                    case 'a': return (CAP);
                    case 'C': return (CCT);
                    case 'c': return (CCT);
                    case 'V': return (CVT);
                    case 'v': return (CVT);
                    default: break;
                                        }
        case 'c': switch (circuito[i].elemento[1]) {
                    case 'A': return (CAP);
                    case 'a': return (CAP);
                    case 'C': return (CCT);
                    case 'c': return (CCT);
                    case 'V': return (CVT);
                    case 'v': return (CVT);
                    default: break;
                                        }
        case 'D': return (DIO);
        case 'd': return (DIO);
        case 'F': switch (circuito[i].elemento[1])                    {
                        case 'T': switch (circuito[i].elemento[2])      {
                                    case 'I': return (FTI);
                                    case 'i': return (FTI);
                                    case 'Q': return (FTQ);
                                    case 'q': return (FTQ);
                                    case 'P': return (FTP);
                                    case 'p': return (FTP);
                                    default: break;
```

```
                                        }
                    case 't': switch (circuito[i].elemento[2])          {
                                case 'I': return (FTI);
                                case 'i': return (FTI);
                                case 'Q': return (FTQ);
                                case 'q': return (FTQ);
                                case 'P': return (FTP);
                                case 'p': return (FTP);
                                default: break;
                                                }
                    case 'C': switch (circuito[i].elemento[2])    {
                                case 'I': return (FCI);
                                case 'i': return (FCI);
                                case 'Q': return (FCQ);
                                case 'q': return (FCQ);
                                case 'P': return (FCP);
                                case 'p': return (FCP);
                                default: break;
                                                }
                    case 'c': switch (circuito[i].elemento[2])    {
                                case 'I': return (FCI);
                                case 'i': return (FCI);
                                case 'Q': return (FCQ);
                                case 'q': return (FCQ);
                                case 'P': return (FCP);
                                case 'p': return (FCP);
                                default: break;
                                                }
                    default: break;
                                }
case 'f': switch (circuito[i].elemento[1])  {
                    case 'T': switch (circuito[i].elemento[2])    {
                                case 'I': return (FTI);
                                case 'i': return (FTI);
                                case 'Q': return (FTQ);
                                case 'q': return (FTQ);
                                case 'P': return (FTP);
                                case 'p': return (FTP);
                                default: break;
                                                }
                    case 't': switch (circuito[i].elemento[2])    {
                                case 'I': return (FTI);
                                case 'i': return (FTI);
                                case 'Q': return (FTQ);
                                case 'q': return (FTQ);
                                case 'P': return (FTP);
                                case 'p': return (FTP);
                                default: break;
                                                }
                    case 'C': switch (circuito[i].elemento[2])    {
                                case 'I': return (FCI);
                                case 'i': return (FCI);
                                case 'T': return (FCQ);
                                case 't': return (FCQ);
                                case 'P': return (FCP);
                                case 'p': return (FCP);
                                default: break;
                                                }
                    case 'c': switch (circuito[i].elemento[2])    {
                                case 'I': return (FCI);
                                case 'i': return (FCI);
```

```
                                        case 'T': return (FCQ);
                                        case 't': return (FCQ);
                                        case 'P': return (FCP);
                                        case 'p': return (FCP);
                                        default: break;
                                                }
                                default: break;
                                        }
                case 'V': switch (circuito[i].elemento[1]) {
                                case 'C': return (VCT);
                                case 'c': return (VCT);
                                case 'V': return (VVT);
                                case 'v': return (VVT);
                                default: break;
                                        }
                case 'v': switch (circuito[i].elemento[1]) {
                                case 'C': return (VCT);
                                case 'c': return (VCT);
                                case 'V': return (VVT);
                                case 'v': return (VVT);
                                default: break;
                                        }
                default: {
                        fprintf (stderr, "\n\nElemento %s", circuito[i].elemento);
                        fprintf (stderr, " definido erradamente.\007");
                        exit (ERR_UNKNOWN_ELEMENT);
                        }
                                        }
        exit (ERR_ROUTINE_PROBLEM);
        }

int gerente(int flag)
        {
        cntrl_conv -= flag;
        if (cntrl_conv == 0)
          return (0);
        else return (1);
        }

float busca_tempo (int tipo_ula)
        {
        switch (tipo_ula) {
                case RES: return (0.360);
                case CAP: return (0.400);
                case DIO: return (1.000);
                case VCT: return (0.341);
                case CCT: return (0.341);
                case CVT: return (0.283);
                case VVT: return (0.283);
                case FTI: return (0.277);
                case FTQ: return (0.344);
                case FTP: return (0.373);
                case FCI: return (0.335);
                case FCQ: return (0.402);
                case FCP: return (0.431);
                default: {   fprintf (stderr,"\n\nULA %d nao disponivel.\007",tipo_ula);
                             exit (ERR_UNKNOWN_ELEMENT);
                         }
                        }
        exit (ERR_ROUTINE_PROBLEM);
        }
```

```
int pesquisa_tabela ()
   {
   int j, volta = -1;
   static float primeiro=0;
   for (j = num_elementos - 1; j >= 0 ; j--)
        if (tabela_de_eventos[j] <= primeiro)
          {
          primeiro = tabela_de_eventos[j];
          volta = j;
          }
   primeiro = tabela_de_eventos[volta] += busca_tempo(mph[volta].ula);
   return (volta);
   }

int processador (int i, float t)
   {
   switch (mph[i].ula) {
                          case RES: return (resistor (i, t));
                          case CAP: return (capacitor (i, t));
                          case DIO: return (diodo (i));
                          case FTI: return (fonte_tensao (i, t));
                          case FTQ: return (fonteT_quadrada (i, t));
                          case FTP: return (fonteT_pulsada (i, t));
                          case CVT: return (fonteT_controleI (i));
                          case VVT: return (fonteT_controleV (i));
                          case FCI: return (fonte_corrente (i, t));
                          case FCQ: return (fonteI_quadrada (i, t));
                          case FCP: return (fonteI_pulsada (i, t));
                          case CCT: return (fonteI_controleI (i));
                          case VCT: return (fonteI_controleV (i));
                          default:
                               {
                               printf ("\n\nA ULA para o elemento ");
                               printf ("%10s", mph[i].pte->elemento);
                               printf (" nao esta\' disponivel.\n");
                               exit (ERR_UNKNOWN_ELEMENT);
                               }
                          }
   return (0);
   }

float *media_tensoes (int i)
     {
     float *V;
     int fonte[2];
     fonte[0] = fonte[1] = 0;
     if(!(V = new float[NUM_POLOS]))
        {
        fprintf(stderr,"\nMemoria insuficiente.\n");
        exit(ERR_OUT_OF_MEMORY);
        }
     for (int conta_polos = 0; conta_polos < NUM_POLOS; conta_polos++)
          for (int conta_elementos = 0; conta_elementos < mph[i].num_entradas[conta_polos]; conta_elementos++)
             if (mph[i].fila_entrada[conta_polos][conta_elementos]->fonteTA)
                  {
                  V[conta_polos] = mph[i].fila_entrada[conta_polos][conta_elementos]->tensao[1];
                  comunica_proc_proc[i][mph[i].fila_entrada[conta_polos][conta_elementos]->processador]++;
                  fonte[conta_polos] = !0;
                  }
             else if (e_terra(i,conta_polos))
                       {
```

```
                    V[conta_polos] = 0;
                    fonte[conta_polos] = !0;
                    }
    for (conta_polos=0 ; conta_polos<NUM_POLOS ; conta_polos++)
        if (!fonte[conta_polos])
          {
        int aponta;
          V[conta_polos] = 0;
          for (conta_elementos=0 ; conta_elementos<mph[i].num_entradas[conta_polos] ; conta_elementos++)
               {
               aponta = mph[i].fila_entrada[conta_polos][conta_elementos]->processador;
               if (tabela_de_eventos[i] <= tabela_de_eventos[aponta])
                 aponta = 0;
           else aponta = 1;
               V[conta_polos] += mph[i].fila_entrada[conta_polos][conta_elementos]->tensao[aponta];
               comunica_proc_proc[i][mph[i].fila_entrada[conta_polos][conta_elementos]->processador]++;
               }
          V[conta_polos] /= mph[i].num_entradas[conta_polos];
          }
    return V;
    }

float soma_correntes (int i)
    {
    int terra[2];
    float I[2];
    I[0] = I[1] = 0.0;
    terra[0] = terra[1] = 0;
    for (int conta_polos=0 ; conta_polos<NUM_POLOS ; conta_polos++)
        if (e_terra(i,conta_polos))
          terra[conta_polos] = !0;
        else
          {
          int aponta;
          for (int conta_elementos=0 ; conta_elementos<mph[i].num_entradas[conta_polos] ; conta_elementos++)
               {
               aponta = mph[i].fila_entrada[conta_polos][conta_elementos]->processador;
               if (tabela_de_eventos[i] <= tabela_de_eventos[aponta])
                 aponta = 0;
               else aponta = 1;
               if (mph[i].fila_saida[conta_polos].fonteQQ ==
                                              mph[i].fila_entrada[conta_polos][conta_elementos]->fonteQQ)
                 {
                 comunica_proc_proc[i][mph[i].fila_entrada[conta_polos][conta_elementos]->processador]++;
                 if (conta_polos == mph[i].fila_entrada[conta_polos][conta_elementos]->polo)
                   I[conta_polos] -= mph[i].fila_entrada[conta_polos][conta_elementos]->corrente[aponta];
                 else I[conta_polos] += mph[i].fila_entrada[conta_polos][conta_elementos]->corrente[aponta];
                 }
               else
                 {
                 comunica_proc_proc[i][mph[i].fila_entrada[conta_polos][conta_elementos]->processador]++;
                 if (conta_polos == mph[i].fila_entrada[conta_polos][conta_elementos]->polo)
                   I[conta_polos] += mph[i].fila_entrada[conta_polos][conta_elementos]->corrente[aponta];
                 else I[conta_polos] -= mph[i].fila_entrada[conta_polos][conta_elementos]->corrente[aponta];
                 }
          }
          }
    if (terra[0] && terra[1])   {
                    fprintf (stderr, "Elemento %s em curto.", mph[i].pte->elemento);
                    exit (ERR_SHORT_CCT);
                    }
```

```
    else if (!terra[0] && !terra[1])
            return (I[0]+I[1])/2;
    else if (terra[0])
            return I[1];
    else return I[0];
    }

int tramites_finais (float Va, float Vb, float Iab, int i)
   {
   float delta_Va, delta_Vb, delta_Iab, erro_Va, erro_Vb, erro_Iab;
   int flag;
   if (fabs(Va) < fabs(mph[i].memoria.grandeza_tensao_A * dados.erro))
        Va = 0;
   if (fabs(Vb) < fabs(mph[i].memoria.grandeza_tensao_B * dados.erro))
        Vb = 0;
   if (fabs(Iab) < fabs(mph[i].memoria.grandeza_corrente * dados.erro))
        Iab = 0;
   mph[i].fila_saida[1].tensao[0] = mph[i].memoria.tensao_A;
   mph[i].fila_saida[0].tensao[0] = mph[i].memoria.tensao_B;
   mph[i].fila_saida[1].corrente[0] = mph[i].fila_saida[0].corrente[0] = mph[i].memoria.corrente;
   mph[i].fila_saida[1].tensao[1] = Va;
   mph[i].fila_saida[0].tensao[1] = Vb;
   mph[i].fila_saida[1].corrente[1] = mph[i].fila_saida[0].corrente[1] = Iab;
   delta_Va = fabs(Va - mph[i].memoria.tensao_A);          /*Va(t) - Va(t-1)*/
   delta_Vb = fabs(Vb - mph[i].memoria.tensao_B);          /*Vb(t) - Vb(t-1)*/
   delta_Iab = fabs(Iab - mph[i].memoria.corrente);        /*I(t)  - I(t-1) */
   mph[i].memoria.tensao_A = Va;
   mph[i].memoria.tensao_B = Vb;
   mph[i].memoria.corrente = Iab;
   erro_Va = fabs(dados.erro * Va);
   erro_Vb = fabs(dados.erro * Vb);
   erro_Iab = fabs(dados.erro * Iab);
   if(delta_Va <= erro_Va && delta_Vb <= erro_Vb && delta_Iab <= erro_Iab)
         flag = 1;                                 /*convergiu*/
   else flag = -1;                                 /* nao convergiu */
   if (flag > mph[i].memoria.flag)                 /* entrou em convergencia */
     {
     mph[i].memoria.flag = flag;
     return (1);
     }
   else if (flag < mph[i].memoria.flag)            /* saiu da convergencia */
        {
        mph[i].memoria.flag = flag;
        return (-1);
        }
   else return (0);                                /* condicao de convergencia inalterada */
   }

int resistor (int i, float t)
   {
   float Vab, Iab, *V;
   V = media_tensoes (i);
   Vab = V[1] - V[0];
   Iab = soma_correntes (i);
   if (inicio(t) && !mph[i].memoria.corrente && !mph[i].memoria.tensao_A && !mph[i].memoria.tensao_B)
     {
     if (Vab && !Iab)
          Iab = Vab / mph[i].valor;
     else if (!Vab && Iab)
               Vab = Iab * mph[i].valor;
     }
```

```
  Vab += Iab * mph[i].valor;
  Vab /= 2;
  Iab = Vab / mph[i].valor;
  if (e_terra(i,0))
    {
    V[1] = Vab;
    V[0] = 0.0;
    }
  else if (e_terra(i,1))
          {
          V[1] = 0.0;
          V[0] = -Vab;
          }
  else V[1] = V[0] + Vab;
  return (tramites_finais (V[1], V[0], Iab, i));
  }

int capacitor (int i, float t)
  {
  float Iab, *V, Vab;
  V = media_tensoes(i);
  Iab = soma_correntes (i);
  mph[i].memoria.Vf = busca_fonte_noCAP (i, V);
  if (inicio (t) || transicao (i, t))
    {
    if (inicio (t))
          Vab = 0;
    else  Vab = carga_armazenada (i, t);
    if (e_terra(i, 0) || e_terra(i, 1))
        {
        Iab = Vab = ((V[1] - V[0] + Iab) / 2) + Vab;
        if (e_terra(i,0))
          {
          mph[i].memoria.Vf = V[1] = Vab;
          V[0] = 0.0;
          }
        else if (e_terra(i,1))
                {
                V[1] = 0.0;
                mph[i].memoria.Vf = V[0] = -Vab;
                }
        }
    else if (fonteTA_RECADI (i, V, Vab))
            mph[i].memoria.Vf = busca_fonte_noCAP (i, V);

    else
        {
        V[0] = (V[1] + V[0]) / 2;
        mph[i].memoria.Vf = V[1] = V[0] + Vab;
        }
    mph[i].memoria.Io = Iab;
    mph[i].memoria.transicao = t;
    mph[i].memoria.carga = Vab;
    if (Iab && inicio (t))
        mph[i].memoria.tau = fabs((mph[i].memoria.Vf/mph[i].memoria.Io)*mph[i].valor);
    }
  else
      {
      if (!mph[i].memoria.tau)
        {
        fprintf (stderr, "Problema no calculo de tau no elemento %s.", mph[i].pte->elemento);
```

```
            exit (ERR_ROUTINE_PROBLEM);
            }
          Iab = mph[i].memoria.Io * exp (-(t-mph[i].memoria.transicao)/mph[i].memoria.tau);
          Vab = mph[i].memoria.Vf * (1 - exp (-(t-mph[i].memoria.transicao)/mph[i].memoria.tau));
          Vab += mph[i].memoria.carga * exp (-(t-mph[i].memoria.transicao)/mph[i].memoria.tau);
          if (e_terra(i,0))
            {
            V[1] = Vab;
            V[0] = 0.0;
            }
          else if (e_terra(i,1))
                  {
                  V[1] = 0.0;
                  V[0] = -Vab;
                  }
          else if (fonteTA_RECAD! (i, V, Vab))
                  ;
          else V[1] = V[0] + Vab;
          }
      return (tramites_finais (V[1], V[0], Iab, i));
      }

int inicio (float t)
      {
      if (t == dados.t_inicio)
            return !0;
      return 0;
      }

int transicao (int i, float t)
      {
      float Vab[4], dt[2];
      int antes = (int) ((t-dados.delta_t) / dados.delta_t + 0.5);
      if (t<=(dados.t_inicio + (2*dados.delta_t))) return 0;
      Vab[0] = mph[i].memoria.Vf * (1 - exp (-t/mph[i].memoria.tau));
      Vab[1] = mph[i].memoria.Vfo * (1 - exp (-(t-(0.01*dados.delta_t))/mph[i].memoria.tau));
      Vab[2] = mph[i].memoria.Vfo * (1 - exp (-(t-(0.99*dados.delta_t))/mph[i].memoria.tau));
      Vab[3] = solucoes[i].tensao_A[antes] - solucoes[i].tensao_B[antes];
      dt[0] = (Vab[0] - Vab[1]) / (0.01 * dados.delta_t);
      dt[1] = (Vab[2] - Vab[3]) / (0.01 * dados.delta_t);
      if (fabs(dt[0])==0 && fabs(dt[1])==0)
        return 0;
      else if (fabs(dt[1])==0)
              return !0;
      else if (fabs(dt[0]/dt[1]) > 1)
              return !0;
      return 0;
      }

float carga_armazenada (int i, float t)
      {
      int antes = (int) ((t-dados.delta_t) / dados.delta_t + 0.5);
      return (solucoes[i].tensao_A[antes] - solucoes[i].tensao_B[antes]);
      }

float busca_fonte_noCAP (int i, float *V)
      {
      for (int polo=0 ; polo<NUM_POLOS ; polo++)
            for (int elemento=0 ; elemento<mph[i].num_entradas[polo] ; elemento++)
                  if (mph[i].fila_entrada[polo][elemento]->no && mph[i].fila_entrada[polo][elemento]->fonteTA)
                        return (mph[i].fila_entrada[polo][elemento]->tensao[1]);
```

```
    return (V[1]);
    }

int fonteTA_RECADI (int i, float *V, float Vab)
    {
    for (int polo=0 ; polo<NUM_POLOS ; polo++)
          for (int elemento=0 ; elemento<mph[i].num_entradas[polo] ; elemento++)
             if (mph[i].fila_entrada[polo][elemento]->fonteTA)
                {
                V[polo] = mph[i].fila_entrada[polo][elemento]->tensao[1];
                if (!polo)
                  V[1-polo] = V[polo] + Vab;
                else V[1-polo] = V[polo] - Vab;
                return !0;
               }
    return 0;
    }

void atualiza_capacitores (void)
     {
     for (int i=0 ; i<num_elementos ; i++)
          if (mph[i].ula == CAP)
             mph[i].memoria.Vfo = mph[i].memoria.Vf;
     }

int diodo (int i)
    {
    float *V, Vab, Iab;
    float Vt = 86.168e-6 * (273 + dados.temperatura);
    V = media_tensoes (i);
    Vab = V[1] - V[0];
    if (Vab >= 0.65)
      {
      Iab = soma_correntes (i);
      if (Iab == 0)
           {
           Vab = 0.65;
           Iab = mph[i].I_sat * (exp (Vab / Vt) - 1) + (Vab * 1e-12);
           }
      else if (Iab > 0)
             Vab = Vt * log (Iab / mph[i].I_sat + 1);
      else Vab = -Vt * log (fabs(Iab / mph[i].I_sat + 1));
      }
    else if (Vab < 0)
          {
          Iab = -mph[i].I_sat + (Vab * 1e-12);
          }
    else
          {
          Iab = soma_correntes (i);
          if (Iab > 0)
             Vab = Vt * log (Iab / mph[i].I_sat + 1);
          else Vab = -Vt * log (fabs(Iab / mph[i].I_sat + 1));
          }
    if (e_terra(i,0))
      {
      V[1] = Vab;
      V[0] = 0.0;
      }
    else if (e_terra(i,1))
            {
```

```
            V[1] = 0.0;
            V[0] = -Vab;
            }
    else if (fonteTA_RECADI (i, V, Vab))
            ;
    else V[1] = V[0] + Vab;
    return (tramites_finais (V[1], V[0], Iab, i));
    }

int fonte_tensao (int i, float t)
    {
    float Vab, Iab, *V;
    if (!(V = new float[NUM_POLOS]))
      {
      fprintf(stderr,"\nMemoria insuficiente.\n");
      exit(ERR_OUT_OF_MEMORY);
      }
    if (mph[i].Freq == 0)
      Vab = mph[i].valor;
    else
        {
        Vab = mph[i].valor * cos (2*PI*mph[i].Freq*t);
        if (fabs(Vab) < fabs(mph[i].valor*dados.erro))
          Vab = 0;
        }
    Iab = soma_correntes (i);
    if (e_terra(i,0) && e_terra(i,1))
      {
      fprintf (stderr, "Elemento %s em curto.", mph[i].pte->elemento);
      exit (ERR_SHORT_CCT);
      }
    else if (e_terra(i,0))
            {
            V[0] = 0.0;
            V[1] = Vab;
            }
    else if (e_terra(i,1))
            {
            V[0] = Vab;

            V[1] = 0.0;
            }
    else
      {
      V[0] = 0;
      int aponta;
      for (int j=0 ; j<mph[i].num_entradas[0] ; j++)
            {
            aponta = mph[i].fila_entrada[0][j]->processador;
            if (tabela_de_eventos[i] <= tabela_de_eventos[aponta])
              aponta = 0;
            else aponta = 1;
            V[0] += mph[i].fila_entrada[0][j]->tensao[aponta];
            }
      V[0] /= mph[i].num_entradas[0];
      V[1] = V[0] + Vab;
      }
    return (tramites_finais (V[1], V[0], Iab, i));
    }
```

```
int fonte_corrente (int i, float t)
   {
   float *V, Iab;
   V = media_tensoes (i);
   if (mph[i].Freq == 0)
     Iab = mph[i].valor;
   else
        {
        Iab = mph[i].valor * cos (2*PI*mph[i].Freq*t);
        if (fabs(Iab) < fabs(mph[i].valor*dados.erro))
          Iab = 0;
        }
   if (e_terra(i,0))
     {
     V[1] = V[1] - V[0];
     V[0] = 0;
     }
   else if (e_terra(i,1))
            {
            V[0] = V[0] - V[1];
            V[1] = 0.0;
            }
   return (tramites_finais (V[1], V[0], Iab, i));
   }

int fonte1_controleV (int i)
   {
   int num_tensoes=0;
   float Iab, *V, Tcntrl, Vcntrl[2];
   Vcntrl[0] = Vcntrl[1] = 0;
   V = media_tensoes(i);
   for (int m=0; m<2; m++)
        {
      num_tensoes = 0;
        for (int k=0; k<num_elementos; k++)
           for (int l=0; l<NUM_POLOS; l++)
                if (mph[k].fila_saida[l].no == mph[i].no_Cntrl[m])
                  {
                  Vcntrl[m] += mph[k].fila_saida[l].tensao[1];
                  num_tensoes++;
                  Vcntrl[m] /= num_tensoes;
                  }
        }
   Tcntrl = Vcntrl[0] - Vcntrl[1];
   Iab = mph[i].valor*Tcntrl;
   if (e_terra(i,0))
     {
     V[1] = V[1] - V[0];
     V[0] = 0.0;
     }
   else if (e_terra(i,1))
            {
            V[0] = V[0] - V[1];
            V[1] = 0.0;
            }
   return(tramites_finais(V[1], V[0], Iab, i));
   }

int fonte1_controle1 (int i)
   {
   float Iab, *V;
```

```
  V = media_tensoes (i);
  for (int k=0 ; k<num_elementos ; k++)
       if (mph[i].elmto_Cntrl == k)
          {
          Iab = mph[i].valor * mph[k].memoria.corrente;
          break;
          }

  if (e_terra (i, 0))
    {
    V[1] = V[1] - V[0];
    V[0] = 0.0;
    }
  else if (e_terra (i, 0))
       {
       V[0] = V[0] - V[1];
       V[1] = 0.0;
       }
  return (tramites_finais (V[1], V[0], Iab, i));
  }

int fonteT_controleI (int i)
  {
  float Iab, Vab, *V;
  if (! (V = new float[NUM_POLOS]))
    {
    fprintf (stderr, "\nMemoria insuficiente.\n");
    exit (ERR_OUT_OF_MEMORY);
    }
  Iab = soma_correntes (i);
  for (int k=0 ; k<num_elementos ; k++)
       if (mph[i].elmto_Cntrl == k)
          {
          Vab = mph[i].valor * mph[k].memoria.corrente;
          break;
          }
  if (e_terra (i,0))
    {
    V[1] = Vab;
    V[0] = 0.0;
    }
  else if (e_terra (i, 1))
       {
       V[1] = 0.0;
       V[0] = Vab;
       }
  else
       {
       V[0] = 0;
       for (int j=0 ; j<mph[i].num_entradas[0] ; j++)
        V[0] += mph[i].fila_entrada[0][j]->tensao[1];
       V[0] /= mph[i].num_entradas[0];
       V[1] = V[0] + Vab;
       }
  return (tramites_finais (V[1], V[0], Iab, i));
  }

int fonteT_controleV (int i)
  {
  int num_tensoes;
  float Iab, Tcntrl, Vcntrl[2], *V;
```

```
  if (! (V = new float[NUM_POLOS]))
    {
    fprintf (stderr, "\nMemoria insuficiente.\n");
    exit (ERR_OUT_OF_MEMORY);
    }
  Vcntrl[0] = Vcntrl[1] = 0;
  Iab = soma_correntes (i);
  for (int m=0; m<2; m++)
       {
       num_tensoes = 0;
       for (int k=0 ; k<num_elementos ; k++)
          for (int l=0 ; l<NUM_POLOS ; l++)
               if (mph[k].fila_saida[l].no == mph[i].no_Cntrl[m])
                  {
                  Vcntrl[m] += mph[k].fila_saida[l].tensao[1];
                  num_tensoes++;
                  Vcntrl[m] /= num_tensoes;
                  }
    }
  Tcntrl = Vcntrl[0] - Vcntrl[1];
  if (e_terra (i, 0))
    {
    V[1] = mph[i].valor * Tcntrl;
    V[0] = 0;
    }
  if (e_terra (i, 1))
    {
    V[1] = 0;
    V[0] = -Tcntrl * mph[i].valor;
    }
  else
       {
       V[0] = 0;
       for (int j=0 ; j<mph[i].num_entradas[0] ; j++)
          V[0] += mph[i].fila_entrada[0][j]->tensao[1];
       V[0] /= mph[i].num_entradas[0];
       V[1] = V[0] + (mph[i].valor * Tcntrl);
       }
  return (tramites_finais (V[1], V[0], Iab, i));
  }

int fonteT_quadrada (int i, float t)
  {
  float Vab, Iab, *V;
  if (! (V = new float[NUM_POLOS]))
    {
    fprintf (stderr, "\nMemoria insuficiente.\n");
    exit (ERR_OUT_OF_MEMORY);
    }
  Iab = soma_correntes (i);
  for (int n=1 ; n<=(int) (dados.t_fim / mph[i].Per + 0.5) + 1 ; n++)
       if ( t>=((n-1)*mph[i].Per) && t<((2*n-1)*mph[i].Per/2) )
          {
          Vab = mph[i].valor;
          break;
          }
       else if ( t<(n*mph[i].Per) && t>=((2*n-1)*mph[i].Per/2) )
               {
               Vab = -mph[i].valor;
               break;
               }
```

```
  if (e_terra (i, 0))
    {
    V[1] = Vab;
    V[0] = 0.0;
    }
  else if (e_terra (i, 1))
        {
        V[1] = 0.0;
        V[0] = Vab;
        }
  else
        {
        V[0] = 0;
        for (int j=0 ; j<mph[i].num_entradas[0] ; j++)
        V[0] += mph[i].fila_entrada[0][j]->tensao[1];
        V[0] /= mph[i].num_entradas[0];
        V[1] = V[0] + Vab;
        }
  return (tramites_finais (V[1], V[0], Iab, i));
  }

int fonte1_quadrada (int i, float t)
  {
  float Iab, *V;
  V = media_tensoes (i);
  for (int n=1 ; n<=(int) (dados.t_fim / mph[i].Per + 0.5) + 1 ; n++)
        if ( t>=((n-1)*mph[i].Per) && t<((2*n-1)*mph[i].Per/2) )
          {
          Iab = mph[i].valor;
          break;
          }
        else if ( t<(n*mph[i].Per) && t>=((2*n-1)*mph[i].Per/2) )
                {
                Iab = -mph[i].valor;
                break;
                }
  if (e_terra (i, 0))
    {
    V[1] = V[1] - V[0];
    V[0] = 0;
    }
  else if (e_terra (i, 1))
        {
        V[0] = V[0] - V[1];
        V[1] = 0;
        }
  return (tramites_finais (V[1], V[0], Iab, i));
  }

int fonteT_pulsada (int i, float t)
  {
  float Iab, Vab, *V, tmp[11];
  if (! (V=new float[NUM_POLOS]))
    {
    fprintf (stderr, "\nMemoria insuficiente.\n");
    exit (ERR_OUT_OF_MEMORY);
    }
  Iab = soma_correntes (i);
  while ((tmp[10] = (1+mph[i].memoria.num_per) * mph[i].flp.periodo[1]) < t)
        mph[i].memoria.num_per++;
  tmp[0] = mph[i].memoria.num_per * mph[i].flp.periodo[1];
```

```
  tmp[1] = tmp[0] + mph[i].flp.t_zero[0];
  tmp[2] = tmp[1] + mph[i].memoria.sobe[0];
  tmp[3] = tmp[2] + mph[i].flp.t_max[0];
  tmp[4] = tmp[3] + mph[i].memoria.desce[0];
  tmp[5] = tmp[0] + mph[i].flp.periodo[0];
  tmp[6] = tmp[5] + mph[i].flp.t_zero[1];
  tmp[7] = tmp[6] + mph[i].memoria.sobe[1];
  tmp[8] = tmp[7] + mph[i].flp.t_max[1];
  tmp[9] = tmp[8] + mph[i].memoria.desce[1];
  if ( tmp[4]>tmp[5] || tmp[9]>tmp[10] )
     {
     fprintf (stderr, "Erro na definicao de %s.", mph[i].pte->elemento);
     exit (ERR_AMBIGUOUS_SRC_DEF);
     }
  if ( (t>=tmp[0] && t<=tmp[1]) || (t>tmp[4] && t<=tmp[6]) || (t>tmp[9] && t<=tmp[10]) )
     Vab = 0;
  else if ( t>tmp[1] && t<=tmp[2] )
        Vab = mph[i].flp.subida[0] * (t - tmp[1]);
  else if ( t>tmp[2] && t<=tmp[3] )
        Vab = mph[i].flp.amplitude[0];
  else if ( t>tmp[3] && t<=tmp[4] )
        Vab = mph[i].flp.amplitude[0] - (mph[i].flp.descida[0] * (t - tmp[3]));
  else if ( t>tmp[6] && t<=tmp[7] )
        Vab = mph[i].flp.subida[1] * (t - tmp[6]);
  else if ( t>tmp[7] && t<=tmp[8] )
        Vab = mph[i].flp.amplitude[1];
  else Vab = mph[i].flp.amplitude[1] - (mph[i].flp.descida[1] * (t - tmp[8]));
  if (e_terra (i, 0))
     {
     V[1] = Vab;
     V[0] = 0.0;
     }
  else if (e_terra (i, 1))
        {
        V[1] = 0.0;
        V[0] = Vab;
        }
  else
        {
        V[0] = 0;
        for (int j=0 ; j<mph[i].num_entradas[0] ; j++)
        V[0] += mph[i].fila_entrada[0][j]->tensao[1];
        V[0] /= mph[i].num_entradas[0];
        V[1] = V[0] + Vab;
        }
  return (tramites_finais (V[1], V[0], lab, i));
  }

int fontei_pulsada (int i, float t)
  {
  float lab, *V, tmp[11];
  V = media_tensoes (i);
  while ((tmp[10] = (1+mph[i].memoria.num_per) * mph[i].flp.periodo[1]) < t)
        mph[i].memoria.num_per++;
  tmp[0] = mph[i].memoria.num_per * mph[i].flp.periodo[1];
  tmp[1] = tmp[0] + mph[i].flp.t_zero[0];
  tmp[2] = tmp[1] + mph[i].memoria.sobe[0];
  tmp[3] = tmp[2] + mph[i].flp.t_max[0];
  tmp[4] = tmp[3] + mph[i].memoria.desce[0];
  tmp[5] = tmp[0] + mph[i].flp.periodo[0];
  tmp[6] = tmp[5] + mph[i].flp.t_zero[1];
```

```
  tmp[7] = tmp[6] + mph[i].memoria.sobe[1];
  tmp[8] = tmp[7] + mph[i].flp.t_max[1];
  tmp[9] = tmp[8] + mph[i].memoria.desce[1];
  if ( tmp[4]>tmp[5] || tmp[9]>tmp[10] )
    {
    fprintf (stderr, "Erro na definicao de %s.", mph[i].pte->elemento);
    exit (ERR_AMBIGUOUS_SRC_DEF);
    }
  if ( (t>=tmp[0] && t<=tmp[1]) || (t>tmp[4] && t<=tmp[6]) || (t>tmp[9] && t<=tmp[10]) )
    Iab = 0;
  else if ( t>tmp[1] && t<=tmp[2] )
        Iab = mph[i].flp.subida[0] * (t - tmp[1]);
  else if ( t>tmp[2] && t<=tmp[3] )
        Iab = mph[i].flp.amplitude[0];
  else if ( t>tmp[3] && t<=tmp[4] )
        Iab = mph[i].flp.amplitude[0] - (mph[i].flp.descida[0] * (t - tmp[3]));
  else if ( t>tmp[6] && t<=tmp[7] )
        Iab = mph[i].flp.subida[1] * (t - tmp[6]);
  else if ( t>tmp[7] && t<=tmp[8] )
        Iab = mph[i].flp.amplitude[1];
  else Iab = mph[i].flp.amplitude[1] - (mph[i].flp.descida[1] * (t - tmp[8]));
  if (e_terra (i, 0))
    {
    V[1] = V[1] - V[0];
    V[0] = 0;
    }
  else if (e_terra (i, 1))
       {
       V[0] = V[0] - V[1];
       V[1] = 0;
       }
  return (tramites_finais (V[1], V[0], Iab, i));
  }

void calcula_medias (float t)
  {
  float sinal;
  int iteracoes = (int) (t/dados.delta_t+0.5);
  if (iteracoes)
       for (int i=0 ; i<num_elementos ; i++)
          {
          mph[i].memoria.media_corrente = 1;
          mph[i].memoria.media_tensao_A = 1;
          mph[i].memoria.media_tensao_B = 1;
          }
  else for (i=0 ; i<num_elementos ; i++)
           {
           mph[i].memoria.num_iteracoes = iteracoes;
           if (fabs(solucoes[i].corrente[iteracoes]) > fabs(solucoes[i].corrente[iteracoes-1]))
               mph[i].memoria.media_corrente = solucoes[i].corrente[iteracoes];
           if (fabs(solucoes[i].tensao_A[iteracoes]) > fabs(solucoes[i].tensao_A[iteracoes-1]))
               mph[i].memoria.media_tensao_A = solucoes[i].tensao_A[iteracoes];
           if (fabs(solucoes[i].tensao_B[iteracoes]) > fabs(solucoes[i].tensao_B[iteracoes-1]))
               mph[i].memoria.media_tensao_B = solucoes[i].tensao_B[iteracoes];
           if (!mph[i].memoria.media_corrente)
               mph[i].memoria.grandeza_corrente = 1;
           else
               {
               sinal = mph[i].memoria.media_corrente/fabs(mph[i].memoria.media_corrente);
               mph[i].memoria.grandeza_corrente = pow (10, (int) (log10 (sinal*mph[i].memoria.media_corrente)));
               }
```

```
            if (!mph[i].memoria.media_tensao_A)
                mph[i].memoria.grandeza_tensao_A = 1;
            else
                {
                sinal = mph[i].memoria.media_tensao_A/fabs(mph[i].memoria.media_tensao_A);
                mph[i].memoria.grandeza_tensao_A =
                                        pow (10, (int) (log10 (sinal*mph[i].memoria.media_tensao_A)));
                }
            if (!mph[i].memoria.media_tensao_B)
                mph[i].memoria.grandeza_tensao_B = 1;
            else
                {
                sinal = mph[i].memoria.media_tensao_B/fabs(mph[i].memoria.media_tensao_B);
                mph[i].memoria.grandeza_tensao_B =
                                        pow (10, (int) (log10 (sinal*mph[i].memoria.media_tensao_B)));
                }
            }
    }

void carrega_saida (float t)
    {
    int i;
    for (i=0; i<num_elementos; i++)
      {
      solucoes[i].corrente[(int) (t/dados.delta_t+0.5)] = mph[i].memoria.corrente;
      solucoes[i].tensao_A[(int) (t/dados.delta_t+0.5)] = mph[i].memoria.tensao_A;
      solucoes[i].tensao_B[(int) (t/dados.delta_t+0.5)] = mph[i].memoria.tensao_B;
      comunica_proc_host[i] += 3;
      }
    return;
    }
```