SISTEMA CONFIGURÁVEL PARA SIMULAÇÃO DE CIRCUITOS EVENT-DRIVEN

Pedro Manuel Reis dos Santos

INESC, IST

SUMÁRIO

A tecnologia dos circuitos integrados tem evoluído muito rapidamente nos últimos anos. É agora possível desenhar circuitos integrados com centenas de milhar de transistores. É da maior importância para o projectista a avaliação das capacidades do projecto, a correcção dos seus erros ainda durante a fase desenvolvimento e a verificação incremental das decisões tomadas em face de outras possíveis soluções.

Nestes últimos anos têm surgido muitos simuladores para a resolução do problema da verificação de circuitos. Grande número destes simuladores recorre, no entanto, a máquinas de simulação controladas por acontecimentos - *event-driven*. Este facto não surpreende pois este tipo de simulação reduz drasticamente o processamento face à simulação por saltos no tempo - *time-step*.

Neste artigo pretende-se descrever um sistema para simulação de circuitos incluindo uma estrutura que se adapta aos vários tipos e níveis de descrição. O sistema procura ser, numa primeira fase, um ambiente de desenvolvimento de novos simuladores e de integração de algorítmos de simulação já existentes. O ambiente de integração, permite uma sistematização no desenvolvimento dos simuladores propriamente ditos e, oferece uma interface utilizador uniforme facilitando a tarefa do programador. O sistema pretende integrar vários simuladores *event-driven* e incluir mecanismos de simulação mista. Numa fase posterior serão estendidos os mecanismos de simulação mista a simuladores paralelos e *pipelined*.

1. INTRODUÇÃO

Com o objectivo de avaliar as capacidades e o comportamento de um circuito, no seu todo ou apenas de alguns blocos funcionais, o projectista é cada vez mais forçado a recorrer a simuladores. Esta necessidade resulta de os projectos serem cada vez mais complexos e consequentemente mais sujeitos a erros. Também a avaliação do desempenho, a detecção dos erros de especificação e de desenho se tornam mais difíceis com o aumento da complexidade do projecto.

Uma observação do código da maioria dos simuladores existentes evidencia uma grande percentagem dedicada à leitura de descrições de circuitos e estímulos, expansão de hierarquia e manutenção de filas de espera face ao algorítmo de simulação propriamente dito. A definição de novas primitivas de descrição de circuitos origina, quando é abordada, sequências de comandos longas e elaboradas[18].

Sistema para Simulação

Com o objectivo de oferecer os mecanismos necessários ao desenvolvimento de simuladores de circuitos capazes de analisar a crescente complexidade dos circuitos, está em fase de desenvolvimento um sistema para a simulação. O sistema em desenvolvimento funcionará como ambiente de teste dos simuladores propriamente ditos. A estrutura de dados adapta-se aos vários tipos de descrição de circuitos, permitindo que mais de um simulador possa ser invocado sobre uma mesma descrição. Além dos mecanismos habituais de gestão de filas de espera, o sistema incluirá procedimentos para simulação mista ou paralela.

A integração de novos elementos de circuito, modelos de atraso e mesmo simuladores é, no sistema em desenvolvimento, simples de realizar. A inclusão de bibliotecas, simuladores ou elementos é feita pelo programador no momento da ligação dos vários módulos - *linking*. Este facto permite grande flexibilidade no teste do desempenho de novos blocos, bastando para tal criar um binário com o novo código e a biblioteca de rotinas fornecida.

A interface para o preenchimento das estruturas de dados é procedimental - classes e métodos. Serão desenvolvidos *parsers* para os formatos comuns de dados: SPICE[8] e outros. Esta aproximação permite a invocação directa dos simuladores por ferramentas que lhes estão associadas. O recurso a uma interface procedimental garante grande eficiência na invocação do simulador permitindo a visualização dos resultados e controlo da simulação por parte da ferramenta de projecto[1].

A Ferramenta de Simulação

Um simulador é essencialmente uma ferramenta de correcção de erros - *debug* - pelo que se pretende incluir um conjunto de comandos para visualização interactiva, colocação de *break points* em ligações, elementos e instantes de tempo. Notar que o simulador *Cinnamon*[3] já inclui uma versão interactiva para a visualização dos resultados da simulação.

Foram inclusivamente efectuadas experiências com o objectivo de interromper a simulação em determinado ponto e podendo iniciar outras simulações do mesmo circuito a partir desse ponto.

O sistema encontra-se organizado em torno de bibliotecas. Cada biblioteca contém descrições de máquinas de simulação para determinado nível e de elementos de circuito. Assim, o utilizador pode, mediante a indicação do nível de descrição do circuito que dispõe - eléctrico, lógico, comportamental - , obter uma lista de elementos primitivos e simuladores capazes de processar o seu circuito.

2. SIMULAÇÃO

A existência de uma gama muito diversificada de simuladores cobrindo os vários tipos de representação é essencial para garantir um acompanhamento contínuo de um projecto ao longo das suas várias fases. Uma classificação possível para os simuladores baseada no tipo de análise que prefazem e no tipo de primitivas que aceitam é a seguite:

- Simuladores a nível de comportamento ou de sistema[9].
- Simuladores a nível de tranferência entre registos.
- Simuladores lógicos[10,14,11].
- Simuladores de comutação (*switch level*)[12,19].
- Simuladores a nível de circuito eléctrico[8,3,13].

Verifica-se, cada vez mais, um aumento da diversificação de simuladores para um mesmo domínio de representação. Esta diversificação é resultado de uma procura de velocidade e eficiência crescente. Para citar apenas a simulação lógica, existem simuladores que trabalham com três[12], oito[10] ou mesmo quinze níveis lógicos. Quanto aos mecanismos de aceleramento introduzidos podem-se referir simuladores para máquinas paralelas ou *pipelined*[15], simuladores paralelos para máquinas convencionais que usam os processadores de computadores numa rede local para obter o paralelismo[11], além de inúmeras diferenças na realização dos algorítmos[14,2]. Por fim é de referir que simulações de nível eléctrico - mais dispendiosas - estão a recorrer a macromodelos[17] e à utilização de valores tabelados[16] para descrever o comportamento de elementos com características não lineares.

É de notar que a simulação de circuitos pode apresentar grandes semelhanças mesmo em domínios de representação que pouco têm em comum. A título de exemplo veja-se o caso da descrição do circuito: um circuito é em geral composto por elementos com determinado comportamento e um conjunto de interligações; mesmo a simulação (*event-driven*) propriamente dita é baseada na troca de acontecimentos e na gestão de filas de espera dos mesmos para posterior processamento.

Tipos de Simulação

A simulação é uma representação de um sistema de objectos cujo objectivo é compreender o seu comportamento sob determinados estímulos. Os objectos envolvidos numa simulação operam de uma forma mais ou menos independente um dos outros. Assim, torna-se necessário desenvolver mecanismos que coordenem a sincronização das actividades dos diversos objectos. De uma forma geral esta sincronização traduz-se no envio e recepção de acontecimentos ou *events*.

Um aspecto importante das simulações reside no facto de estas modelarem situações que evoluem no tempo. Existem diversas formas de representar as acções dos objectos em simulação face à sua evolução no tempo[5]. Uma aproximação utiliza num relógio central que opera com uma determinada unidade de tempo. Nesta situação, em cada intervalo de tempo é dada a oportunidade a todos os objectos de realizarem determinada actividade. O relógio funciona como dispositivo de sincronização aguardando que todos os objectos realizem as suas acções antes de evoluir para o instante de tempo seguinte. Este tipo de simulação origina, frequentemente, a simulação de tempos mortos - tempos em que nenhum dos objectos tem nenhuma operação a realizar.

Numa outra aproximação o relógio avança até ao instante de tempo em que algum objecto mostrou interesse em realizar certa actividade. Neste caso, o sistema é controlado pelo acontecimento agendado - *event-driven*. A realização deste tipo de simulações exige a gestão de uma fila de espera de acontecimentos agendados pelos objectos. Sempre que um acontecimento acaba de ser processado, o seguinte é retirado da fila de espera, o relógio actualizado para o novo instante e, finalmente, é processado. Desta forma, simulações controladas por acontecimentos reduzem o esforço computacional e consequentemente a duração da simulação, simulando apenas os instantes com actividade.

Este sistema aborda apenas os simuladores *event-driven*. Estes simuladores apresentam grande número de características comuns permitindo a sistematização e generalização pretendida com este trabalho.

3. REALIZAÇÃO

A base do sistema inclui um conjunto de classes abstractas básicas que definem as mensagens que cada tipo de objecto – simulador, elemento de circuito, ligação – deve suportar e um conjunto de serviços mínimo. Existem, ainda no sistema base, o conjunto de mecanismos para declaração de novos objectos e seu armazenamento em bibliotecas especificadas. É também fornecido um conjunto de funções para gestão de filas de espera sequênciais com mecanismos de ordenação, tabelas de *hashing*, cadeias de caracteres, *etc*. Os serviços atrás indicados constituem a infraestrutura – *framework* – para o programador de simuladores e para a criação de novas bibliotecas de componentes.

Podem ser desenvolvidos simuladores controlados por acontecimentos sobre esta infraestrutura. Para tal basta ao programador derivar da classe abstracta fornecida, ou de qualquer outra que a partir dela tenha sido desenvolvida, e reespecificar o algorítmo ou partes do algorítmo que pretende desenvolver. A criação de novos elementos de circuitos e modelos segue um processo idêntico ao descrito para os simuladores.

Uma linguagem orientada por (ou para) objectos é a forma natural de realizar tal máquina de simulação. Como um simulador é uma ferramenta muito exigente em termos de eficiência optou-se por C++ [5,6,4] conseguindo-se assim obter as vantagens oferecidas por uma linguagem orientada para objectos quase sem perda de eficiência.

É de notar que a simulação de circuitos apresenta grandes semelhanças mesmo em níveis de representação que pouco

têm em comum. São igualmente fornecidos elementos de desacopulamento. Estes elementos permitem a comunicação entre blocos descritos em níveis diferentes permitindo a simulação simultânea de circuitos com descrições heterogéneas - simulação mista. Redefinindo os elementos de desacopolamento torna-se possível a realização de simulações paralelas mais ou menos complexas.

Classes Utilitárias

O sistema básico de simulação recorre a um conjunto de classes base. Estas classes base têm uma função semelhante às classes do *Smalltalk* e oferecem ao C++ um nível de abstração mínimo para uma programção orientada para objectos.

Algumas destas classes têm origem em programas anteriormente desenvolvidos tendo sido adaptadas e melhoradas dadas as exigências de uma máquina de simulação. No entanto, a grande maioria destas classes foi realizada de origem podendo vir a ser utilizadas para futuras bases de trabalho dada a sua generalidade.

List Esta classe reune as funcionalidades de algumas das *Collections* do *smalltalk*. Isto deve-se ao facto de embora a interface apresentada poder ser interpretada como uma lista ligada, uma *array* ou um *stack*.

Os objectos que armazena são apenas blocos de memória sem qualquer semântica - *void**, apontador para qualquer coisa. Não oferece nem mesmo o conceito de *Object* do *smalltalk*, permitindo, por isso, armazenar informação que não esteja sobre a forma de objectos.

SortedCltn A *SortedCltn* é uma versão reduzida da classe com o mesmo nome do *smalltalk*. Tem como classes base a *List* e redefine apenas uma função de *insert* - uma forma de *add* com ordem.

Esta classe foi criada apenas como base para a gestão da fila de espera dos acontecimentos. A função de remoção não foi realizada pois o acontecimento que se retira da lista é sempre o primeiro.

StrDict A classe *StrDict* é uma tabela de *hashing* em que a chave é uma cadeia de caracteres.

Estas tabelas são largamente utilizadas no sistema de simulação já que todos os objectos acessíveis pelo utilizador são referidos por nome, existindo, portanto, em tabelas deste tipo.

String Esta classe pretende facilitar a manipulação de cadeias de caracteres e a sua realização justificou-se pelas razões apresentadas na secção anterior.

Table e Ptable Para que os objectos definidos pelos vários simuladores presentes no sistema possam ser acedidos de qualquer ponto existem estas duas tabelas. Estas tabelas ao contrário das tabelas de *hashing* só podem ser declaradas uma vez e cada instância adiciona a uma tabela, que é variável de classe, a própria instância.

A classe *Table* contém a lista das bibliotecas presentes no sistema enquanto a *Ptable* inclui o conjunto dos *parsers* que o utilizador pode referir para carregar a descrição dos circuitos. O funcionamento específico de cada uma destas classes será explicado em maior detalhe nas classes *SimulationTab* e *ParserTab* respectivamente.

Classes do Núcleo

As classes do núcleo ou classes base do sistema são em número de dezoito e dividem-se em três grupos:

- as tabelas, descrevem o comportamento de outras classes e tal como as *Metaclasses* em *Smalltalk* existe uma por cada classe, ou mais precisamente, uma por cada grupo de objectos que exibe um comportamento diferente - por exemplo, *and* e *or* são instâncias de **Gate** mas são referidas independentemente pois apresentam comportamentos físicos (não em termos de objecto) diferentes.
- as classes abstractas que, não oferecendo qualquer algorítmo de processamento dos dados que suportam, limitam-se a garantir um acesso uniforme às classes definidas pelo programador de simuladores quando este as redefine.
- as classes completamente realizadas, exibem um comportamento bem definido e não serão necessariamente redefinidas pelos simuladores, podendo, no entanto, estes fazê-lo sempre que o comportamento desejado não possa ser obtido pela funcionalidade oferecida.

Observer Começando por descrever as classes de interligação do mundo exterior com o sistema de simulador, referirei o *Observer*. Esta classe tem como função fornecer os resultados que vão sendo simulados ao utilizador. Como existe uma grande diversidade de editores e visualizadores de formas onda, cada um suportando o seu formato, é responsabilidade do programador redefinir a forma como cada acontecimento será impresso ou enviado para um processo na mesma máquina ou numa máquina remota, *etc.* Garante-se assim, de uma forma simples, a compatibilidade com todos os sistemas que se pretenda utilizar mediante a redefinição do formato de saída de dados.

Parser O *Parser* representa a outra interface do sistema com o mundo exterior e oferece um conjunto de métodos a ser utilizados pelos vários simuladores bem como suporte para a análise da biblioteca corrente, definição de *Observers*, e circuitos. Esta classe é totalmente abstracta devendo ser totalmente definida pelo programador. Este pode, no entanto, utilizar *Parsers* já desenvolvido utilizando mecanismos que estes ofereçam, como seja, expansão de subcircuitos, inclusão de ficheiros, *etc.*

ParserTab A classe *ParserTab* exporta os nomes dos *Parsers* definidos no sistema. Assim, sempre que um novo *Parser* é realizado deve ser criada uma instância estática de *ParserTab*. Refira-se que uma instância estática é uma instância que é criada pelos mecanismos de C++ no início da execução do programa, mas antes que o main seja invocado.

A declaração de uma instância estática de *ParserTab* é feita por recurso a uma macro definição fornecida. Nesta macro definição deve apenas ser indicado o nome pelo qual se pretende que o novo *parser* passe a ser conhecido pelo utilizador e o nome da classe para que instâncias dessa classe possam criadas sempre que o nome do *parser* seja invocado.

Library Toda a informação estática do sistema, excluindo *Observers* e *Parser* que são considerados interface, encontra localizada em bibliotecas. Todas as bibliotecas estão agrupadas numa lista comum e são acedidas por nome.

Cada biblioteca contém listas de descrições de elementos de circuito - *ElementTab* -, de ligações - *Connection* -, de modelos de representação de elementos - *ModelTab* -, de redes para suporte dos elementos - *Network* - e de simuladores capazes de manipular os circuitos descritos à custa desta informação.

Assim existirá, em princípio, uma biblioteca por domínio de simulação, podendo existir mais que uma biblioteca num mesmo domínio sempre que conflitos de nomes possam surgir - *nand* em **CMOS** ou **TTL** - ou representações incompatíveis dentro de um mesmo domínio de simulação - simulação lógica em três, oito ou quinze níveis.

Atom Todo o objecto passível de ser agendados de alguma forma deriva da classe *Atom*. Assim, são átomos os elementos de circuito - *Element* -, os elementos de ligação - *Connection* - e os elementos de conversão entre representações - *Converter*. Todos estes elementos são passíveis de ser processados pelo que um método *process* com funções idênticas aos métodos *tasks* e *activate* é incluído. Como são estes elementos que constituem o circuito a simular, um conjunto de funções para os interligar é igualmente fornecido.

Connection, Converter e Element As classes *Connection*, *Converter* e *Element* são classes derivadas de *Atom* e apresentam grandes semlhanças. As diferenças entre elas residem mais na utilização que se espera que o programador lhes dará que na funcionalidade intrínseca fornecida pelo núcleo do sistema.

Excetue-se o caso do *Element* que inclui manipulação de modelos, ou seja, inclui métodos que permitem associar um determinado modelo com o elemento de circuito em causa.

Acrescente-se, por fim, que a semelhança entre o *Converter* e a *Connection* não é um mero acaso ou fruto de qualquer decisão furtuita. Qualquer dos elementos em questão tem como função interligar elementos de circuito, a complexidade adicional do *Converter* reside no facto deste ligar elementos entre circuitos descritos em domínios ou tipos de simulação diversos.

Model O modelo tem como função centralizar características comuns a certo número de elementos e que de outra forma teriam que estar repetidos em cada um dos elementos de circuito em uso.

A utilização de modelos é vulgar em quase todo o tipo de simuladores e inclui, normalmente, parâmetros que condicionam a resposta do elemento a determinadas excitações - atrasos na resposta a desterminados estímulos ou amplitudes de resposta a esses mesmos estímulos, só para exemplificar algumas das aplicações mais correntes.

A classe *Model* é derivada de *List* pois cada um dos seus parâmetros é referido pela posição onde se encontra na lista da tabela do modelo - *ModelTab* -, evita-se, assim, a duplicação da informação relativa à identificação de cada parâmetro.

ModelTab A classe *ModelTab* inclui, além dos mecanismos para criar instâncias da classe *Model*, um nome - o nome do modelo - e uma lista de nomes dos parâmetros.

O funcionamento do *ModelTab* resume-se a, dados o nome do parâmetro, indicar qual a sua posição na lista para que o *Model* possa colocar o seu valor na mesma posição da sua lista.

A lista de parâmetros do *ModelTab* não é uma *hash table* pois a posição de cada parâmetro na lista tem de ser conhecida *à priori* pelo elemento de circuito que a vai usar para o acesso ao ser valor possa ser feito por índice. O acesso por índice é indispensável pois cada vez que o elemento é activado, as suas saídas devem ser recalculadas em função destes parâmetros, sendo o tempo de processamento altamente dependente deste tipo de acessos.

ElementTab Cada instância desta classe - *ElementTab* - refere um elemento de circuito que pode ser directamente instanciado pelo utilizador. Convém repetir que não existe uma relação de um para um entre as instâncias de *ElementTab* e as classes de *Element* pois uma mesma classe pode ser utilizada como elementos diferentes de circuito desde que inicializada com configurações diferentes. Acrescente-se que é responsabilidade do *ElementTab* manter, por instância um apontador e um inteiro que cada classe *Element* pode utilizar como lhe aprouver para se configurar de diferentes formas. Consegue-se, assim, obter no caso das porta lógicas, diferentes resultados pela simples alteração de uma tabela. As aplicações deste recurso são vastas já que o apontador pode indicar uma função definida pelo programador e onde esteja concentrada toda a semântica do elemento.

ConverterTab Para estabelecer a comunicação entre dois circuitos com descrições de alguma forma incompatíveis criou-se a classe *Converter*. A classe *ConverterTab* controla não só a criação das instâncias de *Converter* como, quais os tipos ou domínios de descrição que é capaz de converter. A definição das descrições a converter é feita, através da indicação das respectivas bibliotecas, no momento da criação da instância de *ConverterTab*, ou seja, na definição de um novo conversor para o sistema.

Network e NetworkCall A classe *Network* pretende agrupar todos os elementos que descrevem um circuito ou subcircuito. A grande utilidade desta classe surge na construção do circuito a na atribuição de máquinas de simulação. Cada subcircuito pode ser simulado de forma independente, sendo então necessária a introdução de *NetworkCalls* no circuito base, ou expandido nos seus elementos.

No caso de simulações mistas os subcircuitos não podem ser expandidos e além da adição de uma *NetworkCall* por circuito devem igualmente ser inseridos conversores, devendo por fim ser criada uma instância da classe *Simulation* conveniente e ser-lhe atribuido o circuito respectivo.

Notar que mesmo a utilização de *NetworkCalls* não evita a duplicação do subcircuito pelo número de vezes que este é referido. Isto deve-se ao facto de os elementos conterem não só informação das ligações a que estão sujeitos mas o estado em que se encontram. Como o estado de um mesmo elemento em duas instâncias pode ser diferente este tem que ser duplicado. Estudos no sentido de introduzir um novo nível de

descrição com a separação da informação estrutural da informação de estado estão a ser feitos e serão introduzidos em futuras versões do sistema se tal se justificar.

Event O acontecimento - *Event* - é o centro da simulação *Event-Driven* e é da sua responsabilidade transportar ao longo do circuito os estímulos que provocam a evolução do comportamento do mesmo. De uma forma muito resumida podemos descrever simulação *Event-Driven* como a geração, transformação e aramzenamento de acontecimentos e da informação neles contida.

Muito embora cada instância da classe *Event* inclua apenas uma referência para o átomo que a criou, sendo por isso uma classe abstracta, contém funcionalidade suficiente para garantir a transferência da informação entre os vários elementos do circuito e entre estes e o seu simulador.

Simulações em que apenas o acontecimento seja relevante, e não o tipo de fenómeno verificado, podem utilizar a classe *Event* sem a redefinir. No entanto, como tal não é o caso da esmagadora maioria dos simuladores pelo que esta classe deverá ser redefinida pelo programador do simulador directamente a partir da classe *Event* ou de outra qualquer que tenha sido definida à custa desta.

Simulation e SimulationTab Temos por fim as classes que permitem definir e realizar máquinas de simulação. A relação existente entre a classe *Simulation* e *SimulationTab* é, como em casos semelhantes já apresentados, a existência de uma instância de da classe *SimulationTab* por cada classe *Simulation* definida.

A classe *SimulationTab* inclui o nome pelo qual deve ser referido e a biblioteca em que se insere. Como informação adicional, e além dos mecanismos para criar instâncias de *Simulation*, contém um apontador e um inteiro que cada instância de *Simulation* pode utilizar como parâmetros de configuração.

A classe *Simulation* é, nesta versão do núcleo, um mero sequenciador de eventos com algum controlo sobre o estado geral do circuito. Este controlo é necessário para poder interromper e acompanhar a evolução passo a passo do circuito, se tal for desejado.

Como é a simulação a única que, uma vez introduzido o circuito, continua a ter controlo sobre este, suporta mecanismos de acesso aos vários elementos para a alteração ou simples inspecção do seu estado. Permite, ainda, a alteração das ligações do circuito durante uma interrupção da simulação, sendo a responsabilidade da manutenção da consistência totalmente alheia à máquina de simulação, como é evidente.

4. mini: UM SIMULADOR LÓGICO DE 3 NÍVEIS

Para efectuar os primeiros testes do sistema foi desenvolvido um simulador com três níveis lógicos e um modelo de atraso simples - cada elemento apresenta apenas um valor de atraso independetemente do tipo de transição que sofreu.

Este simulador inclui apenas 450 linhas de código das quais 200 referem-se ao *parsing* da informação. Na representação utilizada apenas um terminal de saída pode estar ligado a uma *Connection*. Esta simplificação impede a utilização de elementos de circuito bidireccionais e de portas lógicas com saídas *open collector* ou *open emmiter*, o que não é relevante já que o simulador é apenas experimental e foi desenvolvido com o objectivo de testar o sistema e não as opções da máquina de simulação. O eixo dos tempos utilizado é inteiro da mesma forma que os valores dos modelos, que neste caso são apenas modelos de atraso. Para a representação dos três níveis lógicos optou-se por 0, 1 e 2 sobre um **int** de C++, permite-se assim a reutilização de algumas classes ou métodos em simuladores com oito ou quinze níveis.

Resultados

Nesta primeira fase, é apenas possível apresentar simulações de pequenos circuitos, não porque o simulador desenvolvido não tenha capacidades para isso mas, porque o formato de entrada de dados não é compatível com as descrições disponíveis. Acrescente-se que já está a ser desenvolvido trabalho com vista à integração do próprio *parser* do **Logdet**[10], bastando para tal a adaptação das suas rotinas ao ambiente proporcionado pela classe *Parser*.
Seguidamente apresentam-se descrições e resultados de simulação de dois circuito distintos. O primeiro, a realização de uma porta lógica **exor** à custa de portas **nand**, utiliza atrasos unitários em todas as suas portas. O segundo é constituído por quatro **buffers** cujas saídas se encontram ligadas a uma porta **and** de quatro entradas. O objectivo deste último circuito, sem função bem definida, foi constituir um primeiro teste aos modelos de atraso e à ordenação dos acontecimentos na fila de espera do simulador. Para isso cada um dos quatro **buffers** apresenta tempos de atraso diferentes, permitindo agendar, sempre que as suas entradas se alteram, quatro acontecimentos em simultâneo para quatro instantes de tempo distintos.

EXOR COM NANDS

```
#! logdet
nand nand1 in1 in2 nand
nand nand2 in1 nand tmp1
nand nand3 in2 nand tmp2
nand nand4 tmp1 tmp2 exor

pulse In1 in1 0 1 10 0
pulse In2 in2 0 1 5 0 15 1
```

MODELOS DE ATRASO

```
#! logdet
buf buf1 in i1
buf buf2 in i2
buf buf3 in i3
buf buf4 in i4
and out i1 i2 i3 i4 out

model buf1 1 buf2 2 \
 buf3 3 buf4 4 out 3

pulse in in 0 1 10 0
```

Figura 1: Descrição dos circuitos simulados.

Figura 2: Resultados da simulação da EXOR.

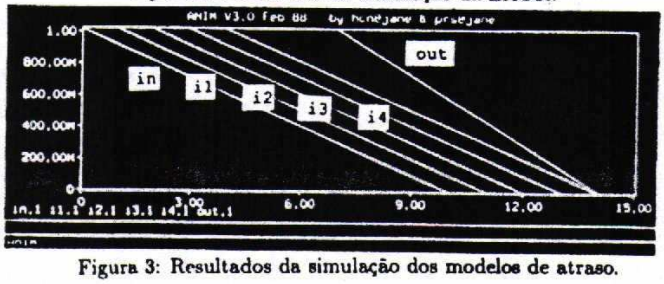

Figura 3: Resultados da simulação dos modelos de atraso.

5. CONCLUSÃO

Se o projecto de circuitos pretende manter-se a par com a tecnologia de fabricação, aproximações completamente novas de desenho assistido por computador terão que ser encontradas. As ferramentas de simulação existentes estão a atingir a exaustão, e em breve provar-se-ão incapazes de garantir os requisitos exigidos pelo projectista.

Um dos grandes óbices à completa automatização da tarefa de verificação do projecto ao longo das suas várias fases reside na enorme diversidade dos formatos de entrada (e de saída) dos dados e de produção de resultados dos programas disponíveis para os diferentes níveis de representação. Este facto exige um grande esforço de reconversão em cada mudança de nível de abstração, tarefa que é extremamente vulnerável à introdução de erros.

Mediante o recurso a um sistema integrado para ferramentas de simulação de circuitos de simulação prevê-se que seja possível testar mais facilmente novos algorítmos bem como integrar os simuladores desenvolvidos noutras fases do projecto de circuitos. Dotando os simuladores de mecanismos de extensão, permite-se sua fácil modificação e actualização das suas bibliotecas.

No que se refere ao simulador **mini** não entrarei em detalhes de realização deste simulador, muito menos em descrições do seu funcionamento. Assim, referirei apenas que o maior esforço foi despendido no *parser* para a leitura do ficheiro com descrição do circuito e estímulos a aplicar. O trabalho desenvolvido nas restantes classes foi muito reduzido não só pela pequena dimensão e simplicidade do simulador como pelo apoio das classes do núcleo. Foi assim possível verificar a facilidade de desenvolvimento de simuladores sobre esta infraestrutura, tendo ainda ficado por testar a flexibilidade adicional fornecida ao utilizador quando este disposer de vários simuladores capazes de simular um mesmo circuito de diversas formas - onde ressalta a simulação mista pelo aumento de velocidade e rigor que introduz.

REFERÊNCIAS

[1] Mário Silva, Helena Sarmento, "PACE - A New Approach to Solve the Design Integration Problem", INESC, Relatório Interno, Maio 1988

[2] Horácio C Neto, Luís M Vidigal, "CINNAMON: New Results and Improvements", European Conference on Circuit Theory and Design, Paris, França, Setembro1987

[3] L. M. Vidigal, S. R. Nassif, S. W. Director, "CINNAMON: Coupled INtegration and Nodal Analysis of MOS Networks", 23rd ACM/IEEE Design Automation Conference, 1986

[4] B. Stroustroup, "The C++ Programming Language", Addison Wesley, 1978

[5] Adele Goldberg, David Robson, "Smalltalk-80: The Language and its Implementation", Addison-Wesley Publishing Company, 1984

[6] Bjarne Stroustrup, "What is Object-Oriented Programming", European Conference on Object-Oriented Programming, 1987

[7] Keith E. Gorlen, "Object Oriented Program Support", Computer Systems Laboratory, Division of Computer Research and Technology, Natio nal Institutes of Health, 1986

[8] A Vladimirescu, A. R. Newton, D. O. Pederson, "SPICE Version 2G5 User's Guide", Department of Electrical Engineering and Computer Sciences, University of California, Berkeley

[9] Pedro M.B. Veiga, "Uma Linguagem de Projecto para Especificação e Simulação Hierárquica Multinível de Sistema Digitais", Instituto Superior Técnico, 1984

[10] Carlos F. T. Almeida, "Simulação Lógica por Oito Valores para Verificação do Projecto de Circuitos Digitais Incluindo Elementos Bidireccionais", Instituto Superior Técnico, 1984

[11] Jeffrey M. Arnold, "Parallel Simulation of Digital LSI Circuits", Massachussetts Institute of Technology, 1985

[12] R. E. Bryant, "A Switch-level Simulation Model for Integrated Logic Circuits", Massachusetts Institute of Technology, Laboratory for Computer Science TR-259, 1981

[13] A. R. Newton, A. L. Sangiovanni-Vincentelli, "Relaxation-Based Electrical Simulation", IEEE Transactions on Electronic Design, ED-30:9, September 1983

[14] William J. Dally, "Lazy Event-Driven Simulation", Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 1986

[15] Prathima Agrawal, William J. Dally, Raffi Tutundjian, "Logic Simulation Algorithms for Pipelined Hardware Architectures", AT&T Bell Laboratories, 1987 1986

[16] André Zúquete, Isabel Teixeira, "Modelação com Tabelas: Métodos de Interpolação", 3º Simpósio de Electrónica das Telecomunicações, 1988

[17] A. Zúquete, A. Raposo, P. Teixeira, I. Teixeira, "Macromodelos de Circuitos CMOS: Nível de Circuito e Lógicos", 3º Simpósio de Electrónica das Telecomunicações, 1988

[18] J. Guimarães, H. Neto, L. Vidigal, "Uma Estrutura de Dados Flexível para Modelação de Transistores MOS", 3º Simpósio de Electrónica das Telecomunicações, 1988

[19] H. Neto, L. Vidigal, "A C Implemented Switch Level Simulator", MWSPA Proceedings, 1984