Análise de *Timing* Funcional de Circuitos VLSI Contendo Portas Complexas

José Luís A. Güntzel 1

Ricardo Reis 1

Resumo: Este artigo apresenta um conjunto de modelos e procedimentos que permitem a análise de *timing* funcional de circuitos que contenham portas complexas. Inicialmente, é apresentada uma revisão dos modelos computacionais e dos algoritmos utilizados na análise de *timing* de circuitos VLSI. A seguir, é proposta uma taxonomia que permite classificar os algoritmos de análise de *timing* funcional existentes segundo o número de caminhos simultaneamente tratados e segundo o método utilizado para testar a sensibilização dos caminhos. Finalmente, a análise de *timing* funcional de circuitos que contenham portas complexas é abordada.

Abstract: This article presents a set of models and procedures for performing functional timing analysis of circuits containing complex gates. Initially, a review on computational models and algorithms for timing analysis is presented. After then, it is proposed a new taxonomy for classifying the existing functional timing analysis algorithms. Such taxonomy classifies the algorithms according to the number of paths simultaneously treated and according to the method used for testing path sensitizability. Finally, the functional timing analysis of circuits containing complex gates is addressed.

1 Instituto de Informática, Universidade Federal do Rio Grande do Sul
{guntzel, reis @inf.ufrgs.br}

1 Introdução

A **verificação de *timing*** tem como objetivo determinar se as restrições temporais impostas ao projeto podem ser satisfeitas ou não. De modo mais objetivo, a verificação de *timing* está relacionada com a estimativa do **atraso crítico** dos circuitos e com a **máxima freqüência de operação**. A precisão da verificação de *timing* depende da precisão dos modelos adotados. Por modelos entende-se não apenas os modelos físicos de atraso usados para quantificar o atraso de cada componente do circuito, mas também os modelos computacionais para os atrasos dos componentes e do circuito como um todo. Estes últimos estão relacionados ao modelo de operação do circuito, isto é, se o circuito opera no modo síncrono ou no modo assíncrono

A maioria das técnicas de verificação de *timing* são orientadas a circuitos seqüenciais síncronos. Desta forma, consideramos os aspectos mais importantes ao estimar a máxima freqüência de operação de um circuito seqüencial que pode ser representado pelo modelo de Mealy. O modelo de Mealy, mostrado na figura 1, divide o bloco combinacional em dois sub-blocos distintos: a lógica de próximo estado e a lógica de saída. Considerando-se que os elementos de armazenamento são flip-flops disparados pela borda, então, a cada borda ativa do relógio, o próximo estado é carregado nos flip-flops, tornando-se o estado atual. Neste momento, o novo próximo estado começa a ser computado pela lógica de próximo estado. As saídas podem mudar como conseqüência de uma mudança no estado atual ou como conseqüência de uma mudança nas entradas ou como conseqüência de ambos.

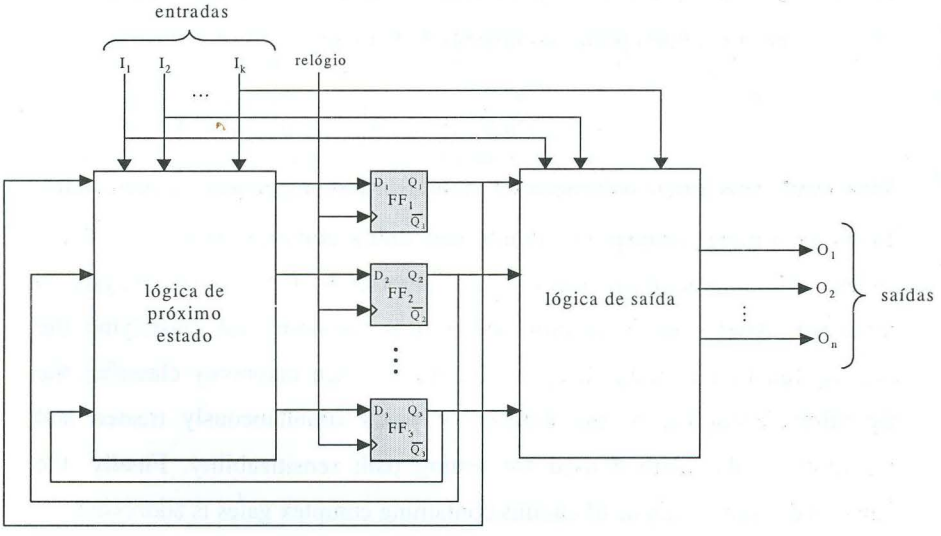

Figura 1 - Modelo de circuito seqüencial síncrono.

Considere que a lógica de próximo estado possui atraso de propagação máximo e mínimo T_{next} e t_{next}, respectivamente, e que a lógica de saída possui atraso de propagação máximo e mínimo T_{out} e t_{out}, respectivamente. Considere também que os flip-flops apresentam máximo atraso de propagação igual a T_{ff}, tempo de preparação (*setup time*) igual a t_s e tempo de manutenção (*hold time*) igual a t_h. Então, para que o circuito opere corretamente, deve-se assegurar as seguintes condições:

- $> \max\{ (T_{ff} + T_{next} + t_s) , (T_{ff} + T_{out}) \}$, onde τ é o período do relógio
- $t_{next} > t_h$

as saídas do circuito devem estar estáveis e válidas durante um tempo maior do que $T_{next} + t_s$ antes de cada borda ativa do relógio.

As condições anteriores são bastante conservadoras, porém asseguram uma operação síncrona correta. A primeira condição assegura que o período do relógio seja longo o suficiente para acomodar o pior caso em termos de atraso do laço de realimentação do próximo estado ($T_{ff} + T_{next} + t_s$) e o pior caso de atraso para a lógica de saída ($T_{ff} + T_{out}$). A segunda condição assegura que o período do relógio é longo o suficiente para que os flip-flops possam amostrar um novo estado. A terceira condição assegura que os sinais de entrada da lógica de próximo estado sejam computados a tempo, de modo que todas as saídas deste bloco estejam estáveis e válidas por um tempo maior ou igual a t_s antes da próxima borda ativa do relógio.

Considere que se adote o modelo de operação descrito anteriormente e que as entradas sejam sincronizadas por flip-flops. Se não houver variação significativa nos tempos de propagação dos flip-flops, pode-se assumir que cada bloco combinacional opera de maneira completamente síncrona, de modo que o atraso de propagação é conseqüência da aplicação de dois vetores de entrada consecutivos. Entretanto, caso os tempos de propagação dos flip-flops sejam significativamente diferentes, os blocos combinacionais operarão de maneira assíncrona, como se seqüências rápidas de vetores fossem aplicadas às suas entradas.

Dentre as técnicas existentes para a estimativa do atraso crítico de blocos combinacionais, a **simulação elétrica** é a que fornece os resultados mais precisos. Por outro lado, o esforço computacional requerido por esta técnica limita seu uso prático para circuitos de tamanho moderado segundo padrões atuais de complexidade. A **simulação de *timing*** é uma técnica similar à simulação elétrica, mas que utiliza modelos de atraso simplificados. Em função disso, a simulação de *timing* pode ser até duas ordens de grandeza mais rápida que a simulação elétrica, ao custo de uma redução de precisão da estimativa de atraso.

Existem, entretanto, três dificuldades sérias com relação ao uso de simulação para se determinar o atraso crítico de circuitos complexos. A primeira é o longo tempo de execução. A segunda diz respeito ao esforço necessário para a preparação do conjunto de estímulos a serem usados na simulação. A terceira, e provavelmente a mais séria, reside em garantir que o conjunto de estímulos propostos exercita a situação na qual o atraso crítico do circuito se manifesta. Sem dúvida, somente a simulação exaustiva nos ofereceria tal garantia. No entanto, tomando-se, por exemplo, um bloco combinacional com 100 entradas, existem 2^{100}

possíveis vetores. Restringindo-se a simulação exaustiva a apenas pares de vetores (modo síncrono) seria necessário simular 2^{100x99} situações diferentes.

Em função das dificuldades citadas anteriormente, a técnica denominada **análise de *timing*** tem sido extensivamente utilizada pelos projetistas na estimativa do atraso crítico de circuitos VLSI estado-da-arte.

A seção 2 apresenta os modelos de cálculo de atraso que conduzem à divisão da análise de *timing* em topológica e funcional. A seção 3 introduz uma nova taxonomia que permite uma classificação didática dos algoritmos de análise de *timing* funcional. A análise de *timing* funcional de circuitos contendo portas complexas é abordada na seção 4. A seção 5 apresenta algumas conclusões.

2 A Análise de *Timing*

A análise de *timing* é uma técnica para estimar-se o atraso crítico de circuitos que dispensa a aplicação de estímulos às entradas do circuito. Para que isso seja possível, cada bloco combinacional é representado como um grafo acíclico direto (DAG) no qual os nodos representam as portas e as arestas representam as conexões. Associado às arestas e às conexões existem pesos que representam os atrasos de portas e conexões, respectivamente. As entradas e as saídas primárias do circuito são representadas por nodos de entrada e de saída (i.e., com peso zero), respectivamente. Freqüentemente, o grafo é polarizado pela inclusão de um nodo fonte e de um nodo terminal. Ao nodo fonte estão ligados todos os nodos de entrada, ao passo que os nodos de saída se ligam ao nodo terminal. Num DAG polarizado, um **caminho completo** assume a forma (**s**, e_s, v_0, e_0, v_1, e_1, ..., v_n, e_n, v_{n+1}, e_t, **t**), onde v_i é um nodo e e_i é uma aresta. Em especial, v_0 e v_{n+1} representam a entrada primária e a saída primária, respectivamente, **s** e **t** são os nodos fonte e terminal e e_s e e_t são arestas sem atraso. Um **caminho parcial** é um caminho que ou não termina com **t** ou não começa com **s**.

A **análise de *timing* topológica** (*topological timing analysis* - TTA) assume que o atraso crítico de um bloco combinacional corresponde ao atraso do caminho mais longo, o qual é encontrado com o uso do algoritmo *topological sort*. Este algoritmo apresenta custo computacional muito reduzido e que aumenta de maneira linear com relação ao tamanho do grafo. Entretanto, o caminho topologicamente mais longo ou **caminho crítico topológico** pode não permitir a propagação de transições. Um caminho que não permite a propagação de transições é denominado **caminho falso** ou **não-sensibilizável**.

O primeiro estudo sistematizado sobre falsos caminhos se deve a Hrapcenko. Considere o circuito da figura 2, retirada de [1]. Suponha que neste circuito cada porta tem atraso unitário e as conexões possuem atraso igual a zero. Este circuito possui dois caminhos críticos topológicos: P_1=(i_1, G_1, G_2, G_3, G_4, G_6, G_7, G_8, h) e P_2=(i_2, G_1, G_2, G_3, G_4, G_6, G_7, G_8, h), ambos com atraso 7. Entretanto, enquanto i_3=0, nenhuma transição consegue se propagar de a para b. Similarmente, enquanto i_3=1, nenhuma transição consegue se propagar de d para f. Desde que um sinal não pode assumir ambos valores lógicos ao mesmo tempo, a única maneira de sensibilizar P_1 e P_2 é aplicar uma seqüência de vetores nas entradas do circuito de

modo que i_3=1 no tempo 1 e i_3=0 no tempo 4. Entretanto, se apenas pares de vetores são permitidos, então P_1 e P_2 são declarados falsos e o atraso do circuito é menor do que 7.

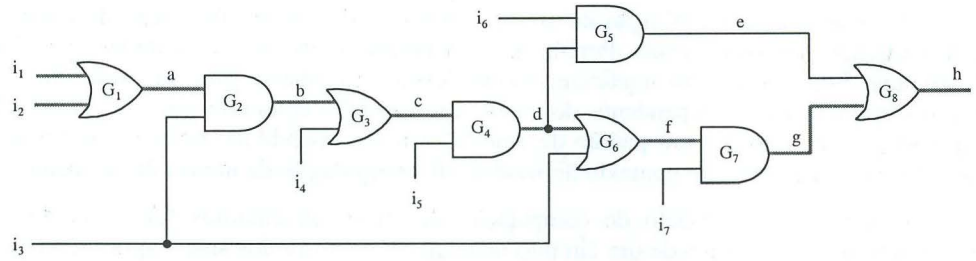

Figura 2 - Exemplo de caminho falso: circuito de Hrapcenko.

Algumas técnicas de síntese lógica são responsáveis pela introdução de redundâncias nos circuitos, as quais são sabidamente uma fonte de falsos caminhos [2]. Além disso, algumas classes de circuitos são projetados de maneira que os caminhos topologicamente mais longos são propositadamente tornados falsos. Este é o caso, por exemplo, dos somadores tipo carry skip [3]. Conclui-se, então, que a ocorrência de falsos caminhos é um fato bastante comum, principalmente devido ao uso extensivo de ferramentas de síntese automática no fluxo de projeto de circuitos VLSI. Por outro lado, caso os caminhos topológicos mais longos sejam falsos, o atraso topológico poderá representar uma estimativa deveras pessimista para o atraso crítico do circuito. Assim sendo, é altamente desejável que um analisador de *timing* seja capaz de considerar o fenômeno dos falsos caminhos.

A fim de considerar o fenômeno dos falsos caminhos, faz-se necessário que se revise o conceito de atraso crítico de blocos combinacionais. A definição mais tradicional é a **baseada em caminhos**: "o **atraso crítico** de um bloco combinacional corresponde ao atraso do caminho sensibilizável mais longo", acrescentando ainda que "pode existir mais de um caminho crítico" [4]. Apesar de correta, tal definição foi derivada considerando apenas as técnicas baseadas em caminho, as quais determinam o atraso crítico avaliando a sensibilização de cada caminho, iniciando pelo mais longo. Uma definição mais geral pode ser derivada notando-se que a essência da análise de *timing* deve ser a captura do exato instante em que a(s) saída(s) mais lenta(s) do circuito estabiliza-se(zam-se) com seu(s) valor(es) final(is). Assim sendo, pode-se redefinir o atraso de um circuito combinacional: o **atraso de um circuito** sob um dado padrão de entradas corresponde à mínima quantidade de tempo após a qual todas as saídas são garantidas estar estáveis. Por extensão, o **atraso crítico do circuito** corresponde ao maior atraso, considerando todos os padrões possíveis de entradas. Por uma questão de simplicidade, o atraso crítico será referenciado apenas por **atraso do circuito**, uma vez que esse é o parâmetro de maior interesse neste trabalho. No contexto da nova definição, a técnica baseada em caminhos é uma solução possível, porém já há um certo tempo não corresponde ao estado-da-arte. Na atualidade, as técnicas de análise de *timing* ("estado-da-arte") operam sobre conjuntos de caminhos, o que resulta em significativa redução do tempo de execução. De um modo geral, qualquer técnica (algoritmo)

de análise de *timing* que leve em consideração os falsos caminhos recai na categoria da **análise de *timing* funcional** (*functional timing analysis* - FTA).

Note-se que na redefinição de atraso e atraso crítico, a sensibilização de caminhos está implicitamente considerada. Porém, não é detalhado como são constituídos os padrões de entrada. Esta questão foi intencionalmente deixada em aberto para que a definição de atraso crítico ficasse independente do modo de operação assumido para o circuito. Na realidade, o que constitui um padrão de entradas válido depende do modo de operação do circuito, e é considerado no contexto do **modelo de computação de atraso de circuitos**.

O conceito de modelo de computação de atraso de circuitos foi motivado pela observação de que o atraso de um circuito depende da natureza dos sinais aplicados às suas entradas, isto é, se as entradas são assumidas como sendo **pares de vetores** ou **seqüências de vetores** [3]. Para ilustrar isso, considere o circuito teste da figura 3a, onde o atraso de cada porta está assinalado no interior da mesma. O caminho mais longo deste circuito é (a, c, f, y), com atraso 5. Se considerarmos pares de vetores sendo aplicados às entradas do circuito, então a última transição de saída ocorre em t=1. A figura 3b mostra um dos possíveis pares de vetores capazes de gerar uma transição em t=1. Por Outro lado, se forem aplicadas seqüências de vetores às entradas do circuito, então a última transição de saída ocorre em t=3. A figura 3c mostra uma possível combinação de entradas do tipo seqüência de vetores capaz de provocar uma transição em t=3. Estes resultados revelam que o atraso de um circuito combinacional depende da maneira como vetores são aplicados às entradas.

O cálculo do atraso para pares de vetores assume que um vetor v_1 é aplicado em $t=-\infty$ e um segundo vetor v_2 é aplicado em t=0. O atraso do circuito é então definido como sendo o pior caso dentre os tempos de estabilização das saídas, considerando-se todos os pares de vetores possíveis. Por essa definição, fica claro que todos os nós do circuito são assumidos como estando estáveis sob v_1 quando v_2 é aplicado.

O atraso para pares de vetores é equivalente a assumir-se que o circuito opera de modo totalmente síncrono. Este tipo de operação é referenciado como **modo de transição** (*transition mode*) e o atraso por ele obtido é chamado **atraso de transição** (*transition delay*) [5].

Alguns autores, dentre os quais Silva et al. [6], sustentam que o atraso de transição de um circuito corresponde ao seu atraso exato. De fato, tal conjectura seria correta caso se pudesse sempre garantir uma operação totalmente síncrona. Entretanto, os elementos de memória, tais como *latches* e *flip-flops*, apresentam tempos de propagação cuja discrepância de valores pode causar um desalinhamento de sinais nas entradas dos blocos combinacionais. Tal desalinhamento tem o efeito de uma seqüência rápida de vetores, caracterizando assim um funcionamento assíncrono. Outro suposto benefício de estimar o atraso usando o modo de transição seria a possibilidade de identificar o par de vetores responsável pelo atraso crítico, o qual poderia ser usado numa simulação para a certificação da estimativa. Entretanto, os resultados mostrados pelos algoritmos que computam o atraso de transição (e.g., [5]) revelam o alto custo computacional, uma vez que todos se baseiam em simulação de pares de vetores.

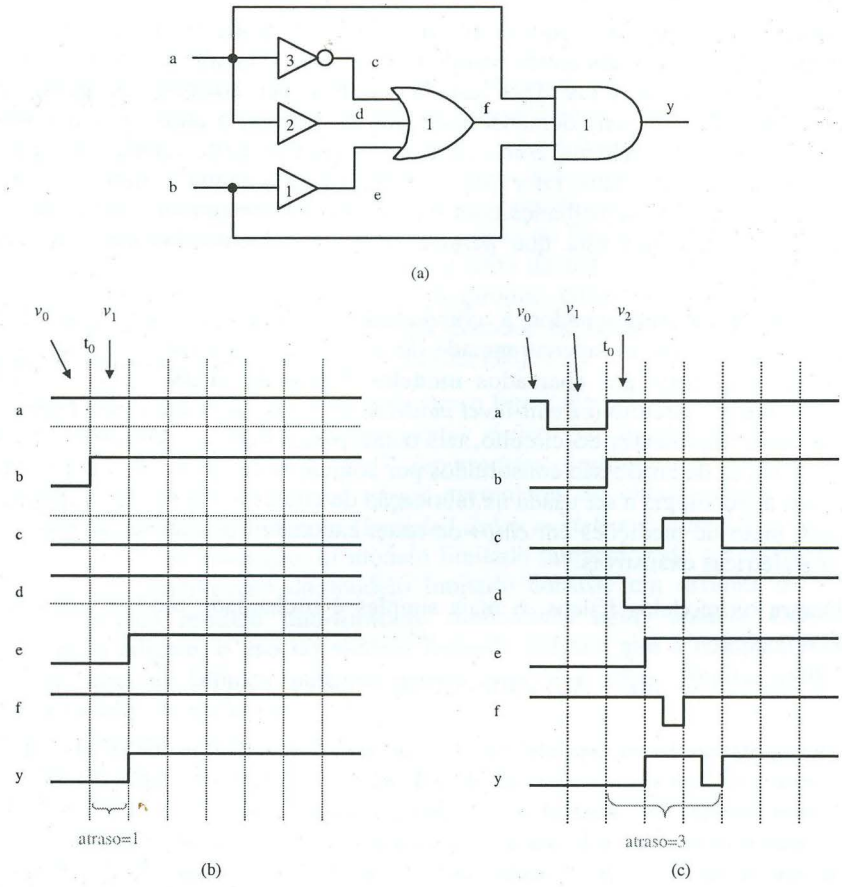

Figura 3 - O atraso dos circuitos depende da maneira como vetores são aplicados às entradas.

A dificuldade encontrada na implementação de algoritmos eficientes para a computação do atraso de transição motivou a adoção da abordagem de **um vetor único**. Esta abordagem é uma aproximação do atraso para seqüência de vetores. A abordagem do vetor único assume valores lógicos arbitrários para os nós do circuito, como se estes estivessem "flutuando", até que a aplicação de um único vetor de entrada v determine os valores estáveis finais. O atraso do circuito é definido como sendo o maior tempo de estabilização das saídas, considerando-se todos os possíveis vetores únicos v_i. A assertiva referente aos nós estarem flutuando decorre do fato de que o circuito pode ainda estar propagando vetores de entrada aplicados antes de v. Na literatura, a abordagem do vetor único é mais conhecida por **modo de operação flutuante** (*floating mode*) [7] e o atraso computado segundo esta é denominado de **atraso flutuante** (*floating delay*).

Apesar do atraso para seqüências de vetores ser teoricamente o método exato, a inexistência de algoritmos eficientes resultou na adoção generalizada do método do vetor único nos algoritmos de FTA. Esta escolha vem a ser corroborada pelos resultados apresentados em [3], os quais demonstraram que na prática, o atraso por um vetor único (atraso flutuante) ou é coincidente com o atraso para seqüências de vetores, ou representa um limite superior para este. Outro fator que contribuiu para a ampla utilização do método do vetor único reside na sua semelhança com o problema da geração automática de vetores de teste (ATPG), semelhança esta que permite o aproveitamento das diversas técnicas de aceleração existentes.

Os problemas relacionados à computação do atraso de blocos combinacionais constituem, a grosso modo, apenas metade do problema da estimativa de atraso. A outra metade está relacionada aos chamados **modelos físicos de atraso** (*physical models*) ou modelos no nível de circuito (*circuit-level models*), os quais são usados para estimar o atraso individual dos componentes do circuito, tais como portas lógicas e conexões. Tipicamente, os modelos físicos de atraso são constituídos por conjuntos de equações com parâmetros que caracterizam a tecnologia a ser usada na fabricação do circuito. Alguns destes parâmetros são obtidos por meio de medições em *chips* de teste, enquanto que outros são provenientes de simulações elétricas exaustivas.

Dentre os modelos físicos, o mais simples é o chamado modelo linear, dado pela equação:

$$d(i) = A(i) + B(i) \times C_L(i) \qquad (A3.1) \qquad (1)$$

onde $A(i)$ é o **atraso de transporte** com relação à saída i, $B(i)$ é o inverso da capacidade de carga da porta e $C_L(i)$ é a carga capacitiva total da rede i, concentrada na saída da porta. Apesar deste modelo não levar em consideração características importantes da tecnologia CMOS corrente (e.g., efeito do canal curto, rampas de entrada lentas), ele ainda é bastante utilizado pela maioria das ferramentas de síntese lógica. Por outro lado, sua simplicidade acarreta uma baixa precisão que o torna pouco útil no contexto de ferramentas de análise de *timing*.

Diversos modelos físicos mais sofisticados têm sido desenvolvidos desde o início dos anos 80. Dentre estes, os pioneiros foram os modelos apresentados no contexto dos simuladores de *timing* **Crystal** [8] e **TV** [9], os quais levam em consideração atraso de portas e de conexões de maneira simultânea. Após, diversos outros trabalhos surgiram, porém, concentrando-se ou no atraso das portas, ou no das conexões.

Dada a existência de diversas possibilidades de modelos físicos de atraso, outro aspecto importante diz respeito a como os atrasos individuais dos componentes serão considerados no contexto da computação do atraso do circuito. Este aspecto é considerado pelo **modelo de computação de atraso de porta**. O modelo mais simples é o **modelo de atraso fixo**, o qual assume um valor fixo d por porta. Uma extensão natural do modelo fixo reside em considerar um atraso d_i para cada entrada i da porta. Outra possibilidade é considerar o atraso de descida e o de subida em separado, podendo ser um par descida/subida por porta $[df,dr]$ ou um par descida/subida por entrada i $[df_i,dr_i]$. O assinalamento de um

valor (fixo) de atraso ou de um par de valores (fixos) de atraso por entrada é referenciado por atraso **pino-a-pino**.

Os modelos citados no parágrafo anterior assumem valores fixos que normalmente correspondem ao máximo atraso individual dos componentes. Tais valores, quando usados no contexto da análise de *timing* topológica (TTA), jamais fornecerão uma subestimativa do atraso do circuito. Porém, é bastante comum que forneçam uma sobrestimativa, cuja gravidade dependerá da diferença de atraso entre o caminho crítico topológico e o caminho crítico real (sensibilizável). Por outro lado, em função do fenômeno dos falsos caminhos, o uso de valores (fixos) máximos de atraso no contexto da análise de *timing* funcional (FTA) pode conduzir a uma subestimativa do atraso do circuito, constituindo assim uma estimativa errônea inaceitável. Este último fenômeno foi chamado de **falha da aceleração monótona** (*monotone speedup failure*) por McGeer e Brayton [10].

A fim de garantir que a estimativa de atraso fornecida pela FTA seja segura (i.e., não seja uma subestimativa), os atrasos das portas devem ser especificados por intervalos limitados do tipo $[d^{min},d^{max}]$, onde d^{min} e d^{max} representam o mínimo e o máximo atrasos que uma dada porta pode apresentar nos vários exemplares do circuito fabricado. Este é o chamado **modelo de atraso limitado** (*bounded delay model*) e é o modelo implicitamente assumido pelo modo de transição. O modelo limitado também pode ser usado no formato pino-a-pino. Uma modificação do modelo limitado consiste em assumir $d^{min}=0$, dando origem ao chamado **modelo não-limitado** (*unbounded delay model*). Conforme será comentado mais adiante, o uso do modelo limitado permite que a computação do atraso flutuante do circuito forneça uma estimativa segura e mais precisa, pois leva em consideração a falha da aceleração monótona.

Uma vez detalhados os modelos de computação de atraso, tem-se os subsídios básicos a análise dos algoritmos de FTA existentes, bem como para determinar os requisitos básicos no desenvolvimento de novos algoritmos. Neste sentido, é importante ressaltar que o objetivo da FTA é fornecer uma estimativa segura de máxima precisão do atraso crítico do circuito. Por estimativa segura de máxima precisão entende-se um valor que represente uma sobrestimativa mais justa possível para o atraso de todos os exemplares fabricados do circuito. Para cumprir este objetivo, um algoritmo de FTA deve satisfazer a duas propriedades. São elas, a **robustez** e a **corretude**. Diz-se que um algoritmo de FTA é robusto se ele assegura uma estimativa de atraso válida para qualquer um dos exemplares fabricados do circuito. Em outras palavras, um algoritmo robusto é capaz de levar em conta a aceleração monótona e por isso, também é denominada propriedade da aceleração monótona [10]. Já a propriedade da corretude diz que o conjunto de testes de sensibilização não devem subestimar o atraso crítico do circuito.

As diversas possibilidades para o desenvolvimento de algoritmos de FTA se diferenciam pelos seguintes aspectos:

- os modelos de computação de atraso para as portas e para o circuito;

- o conjunto de condições usadas para testar a sensibilização dos caminhos. Este conjunto de condições é normalmente referenciado por **critério de sensibilização**;
- o método usado para testar se as condições de sensibilização são satisfeitas ou não.

À medida em que eram desenvolvidos os primeiros trabalhos em FTA, diversos critérios de sensibilização foram propostos. A razão para isso era a busca de um bom compromisso entre a precisão da estimativa de atraso e o tempo de execução do algoritmo usado no teste de sensibilização. Com efeito, este compromisso foi extremamente importante para as primeiras ferramentas de FTA, as quais testavam a sensibilização caminho por caminho, a partir dos caminhos topologicamente mais longos. Entretanto, com a evoluação dos algoritmos de FTA, o **critério de sensibilização exato do modo flutuante** (ou simplesmente **critério exato**) passou a ser amplamente utilizado, em função de sua precisão.

Seja um caminho P= (v_0, e_0, v_1, e_1, ..., v_n, e_n, v_{n+1}) em um circuito *C*. P é dito **exatamente sensibilizável** ou **verdadeiro** (sob o modo flutuante) se e somente se existe ao menos um vetor de entrada *w* tal que, para cada porta v_i ao longo de P, com $1 \leq i \leq n$, uma das condições que seguem é satisfeita:

- se e_{i-1} apresenta o valor controlante de v_i, então cada entrada lateral *e* de v_i que apresentar o valor controlante de v_i deve ser, no máximo, tão rápida quanto e_{i-1}.
- se e_{i-1} apresenta o valor não-controlante de v_i, então cada entrada lateral *e* de v_i deve ser, no mínimo, tão rápida quanto e_{i-1}, e deve apresentar o valor não-controlante de v_i.

Assim sendo, para um caminho ser responsável pelo atraso do circuito sob o modo flutuante é necessário que, para cada porta *g* ao longo do caminho:

- se a saída de *g* apresentar o valor controlado, então sua entrada principal deve apresentar o valor controlante e as entradas laterais que também apresentarem o valor controlante não devem ser mais rápidas que a entrada principal.
- se a saída de *g* apresentar o valor não-controlado, então todas as entradas devem apresentar o valor não-controlante de *g* e sua entrada principal não deve ser mais rápida que as entradas laterais.

3 Algoritmos de Análise de *Timing* Funcional

Apesar do grande número de trabalhos sobre critérios de sensibilização e algoritmos de FTA publicados desde o fim da década dos 80, a ausência de uma terminologia padrão é uma séria dificuldade para o desenvolvimento de atividades de pesquisa nestes temas, sobretudo para principiantes. Nesta seção, propõe-se uma taxonomia para a classificação dos algoritmos de computação de atraso.

Os primeiros algoritmos de FTA testavam a sensibilização dos caminhos, um após o outro, usando adaptações do algoritmo D [11]. Este era o caso dos trabalhos apresentados em [12], [13], [7] e [4]. Nos dois últimos, os critérios de sensibilização usados eram dependentes de atraso. Esta característica de testar um caminho por vez corresponde ao conceito de **sensibilização individual de caminhos** existente em ATPG. Porém, logo alguns autores reconheceram ser bastante comum a ocorrência de circuitos com centenas de milhares de falsos caminhos com atraso maior do que o atraso do caminho crítico (verdadeiro) [14][15][3], de modo que o teste de sensibilização caminho por caminho foi reconhecido ser de uso limitado, na prática. Por outro lado, surgiram diversas propostas de algoritmos de FTA capazes de tratar simultaneamente a sensibilização de conjuntos de caminhos, habilidade esta que em ATPG é conhecida por **sensibilização concorrente de caminhos**. Dentre tais algoritmos merecem destaque o **procedimento de geração de teste temporal** (*timed-test generation procedure*), de Devadas et al. [16] e o trabalho de Silva e Sakallah [17]. Existe ainda uma abordagem **mista**, proposta em [18], na qual um conjunto de potenciais caminhos críticos é identificado usando sensibilização concorrente de caminhos. Após, sensibilização individual é aplicada ao conjunto.

Outra questão, a princípio independente do número de caminhos tratados, diz respeito ao método usado para determinar se as condições de sensibilização podem ser satisfeitas ou não. Nos algoritmos citados nos dois parágrados anteriores, valores lógicos são assinalados a alguns dos nós do circuito e então, implicados para os demais nós. Tal procedimento é derivado dos algorimos tradicionais de ATPG, sendo portanto referenciados por **baseados em ATPG**.

Em 1989 McGeer e Brayton chamaram a atenção para o fato de que as condições de sensibilização poderia ser formalmente expressas por uma função de sensibilização de caminho [19]. Assim, o problema de sensibilização de caminho poderia ser resolvido usando algum método mais formal, baseado em programação dinâmica ou em BDDs, tal como o próprio McGeer propôs em [19] e [10], respectivamente. Naturalmente, é de se imaginar que tal formulação possa também ser aplicada aos algoritmos que realizam sensibilização concorrente de caminhos, obviamente com um aumento significativo na complexidade. Neste sentido, dois trabalhos representam marcas importantes. O primeiro foi o de Larrabee [20], que utilizou **solvabilidade** (*satisfiability*) Booleana para resolver uma formulação que expressava a diferença Booleana entre circuitos com falhas e sem falhas. O segundo, apresentado por McGeer e Brayton [21], propôs as chamadas **funções recursivas**, as quais proveram o formalismo necessário para a aplicação de solvabilidade Booleana para FTA. Nasciam assim os algoritmos de FTA **baseados em solvabilidade** (SAT). Outros algoritmos de FTA baseados em SAT que merecem citação são apresentados em [22], [23], [24] and [6].

Sumarizando a taxonomia apresentada juntamente com a revisão histórica anterior, as soluções possíveis para FTA podem ser classificadas conforme os seguintes quesitos:

1. o critério de sensibilização usado;
2. o número de caminhos simultaneamente tratados e
3. o método usado para determinar se as condições de sensibilização são satisfeitas ou não.

Com relação ao número de caminhos simultaneamente tratados, um algoritmo de FTA pode utilizar uma das seguintes estratégias:

- sensibilização individual de caminhos;
- sensibilização concorrente de caminhos;
- abordagem mista

Finalmente, os métodos possíveis para o teste de sensibilização são:

- baseado em ATPG,
- baseado em solvabilidade (baseado em SAT)
- outros (e.g., BDDs)

A tabela 1 classifica alguns dos algoritmos de FTA mais importantes encontrados na literatura, fazendo uso da taxonomia aqui proposta.

Tabela 1 - Classificação dos principais algoritmos de FTA encontrados na literatura.

algoritmo	sensibilização	n° de caminhos	teste de sensibilização
SLOCOP [12]	estática	individual	ATPG
ACPA [4]	aproximada	individual	ATPG
LLLAMA [19][10]	viabilidade	individual	SAT ou BDDs
XBD0 [21][22]	exata	concorrente	SAT
VIPER [18]	vigorosa	mista	ATPG
TrueD-F [16]	exata	concorrente	ATPG
TA-LEAP [17]	estática "segura"	concorrente	ATPG
STA [23]	estática	concorrente	SAT
GRASP [24]	estática ou viabilidade	concorrente	SAT
CGRASP [6]	estática ou viabilidade	concorrente	SAT

Faz-se necessária, ainda, uma observação final sobre terminologia. Na literatura, a designação "baseada em ATPG" está associada aos algoritmos baseados em ATPG que usam sensibilização concorrente. Isto ocorre por razões históricas, pois que na realidade a sensibilização individual, que é derivada do algoritmo D, surgiu antes da sensibilização concorrente. De modo similar, a designação "baseado em SAT" assume implicitamente a sensibilização concorrente. Embora tal técnica pudesse ser aplicada à sensibilização individual, seu poder reside justamente em tratar os caminhos de maneira concorrente. Por razões didáticas, este texto utiliza a taxonomia proposta anteriormente.

4 Análise de *Timing* Funcional de Circuitos Contendo Portas Complexas

As primeiras técnicas de FTA realizavam sensibilização individual de caminhos utilizando variações do algoritmo D e critérios de sensibilização simplificados, buscando reduzir o tempo execução. Entretanto, logo a sensibilização individual mostrou-se impraticável mesmo para circuitos de complexidade moderada, e a sensibilização concorrente tomou-lhe o lugar. Com a sensibilização concorrente, as técnicas de FTA passaram a utilizar uma abordagem baseada em "enumeração de atraso", ao invés de uma abordagem baseada em "enumeração de caminhos". Além de dispensar a enumeração de caminhos, a sensibilização concorrente também permite a adoção do critério de sensibilização exato do modo flutuante, o qual é o único capaz de fornecer o atraso exato sob o modo flutuante.

Entretanto, toda a teoria que embasa os métodos de teste de sensibilização de caminhos e também os próprios critérios de sensibilização foram desenvolvidos assumindo circuitos compostos por portas simples, isto é, portas E/NÃO-E, OU/ NÃO -OU e inversores. Conseqüentemente, se um circuito combinacional que contém portas mais complexas deve ser analisado, a ferramenta de FTA a ser utilizada deve estar apta não somente a reconhecer tais portas mas também de tratar coerentemente o circuito de acordo com o modelo computacional de atraso adotado. Isto pode ser realizado ou pela inclusão de uma fase de pré-processamento ou pela extensão do algoritmo/método de computação do atraso do circuito e das condições de sensibilização do critério adotado.

A disponibilização de geradores de macrocélulas CMOS eficientes tais como os apresentados em [25] e [26], e de ferramentas de mapeamento tecnológico independentes de bibliotecas [27] tornou possível o uso extensivo de portas complexas (sobretudo portas CMOS estáticas) no projeto físico de grandes blocos combinacionais. Deste modo, a capacidade de tratar circuitos que contenham portas complexas passa a ser altamente desejável para novas ferramentas de FTA.

Antes de iniciar uma discussão sobre a análise de *timing* funcional de circuitos que contenham portas complexas, é importante prover definições para portas simples e portas complexas. Uma **porta simples** corresponde a uma implementação física de um operador básico da álgebra Booleana. Assim, são portas simples as portas E, OU, NÃO-E, NÃO-OU (com qualquer número de entradas) e o inversor. Particularmente, **portas simples CMOS** são portas simples que podem ser implementadas diretamente em tecnologia CMOS, ou seja, portas NÃO-E, NÃO-OU e inversor. Uma porta complexa corresponde a uma implementação física de qualquer função Booleana de complexidade maior do que os operadores Booleanos básicos. Especificamente, uma porta complexa CMOS estática (*static* CMOS *complex gate* – SCCG) corresponde a uma porta complexa implementada em tecnologia CMOS.

Uma porta CMOS estática é implementada por uma rede de transistores CMOS conectados segundo uma topologia de "restauração completa", composta por uma rede PMOS e uma rede NMOS. A rede PMOS é capaz de prover um caminho entre a saída da

porta e a massa (Vdd), enquanto que a rede NMOS é capaz de prover um caminho entre a saída da porta e a terra (Gnd). As redes NMOS e PMOS possuem um mesmo número de transistores. Por uma questão de simplicidade, iremos assumir que uma SCCG é uma porta CMOS estática em que ambas redes de transistores apresentam apenas associações série/paralelo e que a rede NMOS é o dual da rede PMOS, em termos de associação de transistores. A figura 4 mostra um exemplo de SCCG. Note-se que, para uma porta CMOS estática de n entradas, há n pares NMOS/PMOS conectados pela grade, formando cada par uma entrada da porta.

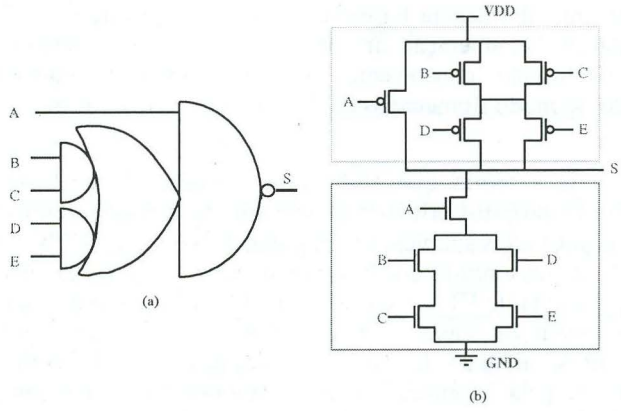

Figura 4 - Exemplo de SCCG.

As portas CMOS estáticas, incluindo as SCCGs, podem ser classificadas de acordo com o número de transistores série/paralelo existentes nas redes NMOS e PMOS. O conjunto das portas CMOS estáticas que apresentam não mais do que n (p) transistores NMOS (PMOS) em série é definido como sendo uma "biblioteca virtual" [28] que pode ser designada por SCG(n,p). Pode-se também utilizar a notação SCCG(n,p) para designar o subconjunto de SCG(n,p) composto apenas por SCCGs. A tabela 2, retirada de [29], detalha o número de SCGs existentes para bibliotecas virtuais de até 5 transistores série.

Tabela 2 - Número de elementos para várias bibliotecas virtuais [29].

		número de transistores PMOS em série				
		1	**2**	**3**	**4**	**5**
Número de transistores NMOS em série	**1**	1	2	3	4	5
	2	2	7	18	42	90
	3	3	18	87	396	1677
	4	4	42	396	3503	28435
	5	5	90	1677	28435	425803

Muitos algoritmos de FTA foram desenvolvidos na década dos 90. Entretanto, a maioria destes não considera a possibilidade de tratar circuitos que contenham portas complexas. No caso dos algoritmos baseados em ATPG, tal limitação deve-se ao fato da teoria de sensibilização de caminhos ter sido desenvolvida unicamente para portas simples. Obviamente, a extensão de um critério de sensibilização para considerar portas complexas resulta em regras mais complicadas. O uso de tais regras por um algoritmo de sensibilização individual de caminhos tende a piorar significativamente o desempenho deste. Em uma primeira análise, os algoritmos baseados em SAT parecem ser mais promissores, uma vez que as equações características são potencialmente capazes de representar qualquer função Booleana. Entretanto, a fim de reduzir a complexidade das instâncias de SAT a serem resolvidas, alguns algoritmos baseados em SAT assumem que os circuitos combinacionais são compostos unicamente de portas simples.

A solução mais simples para realizar-se FTA de circuitos contendo portas complexas consiste em substituir-se cada porta complexa por um subcircuito equivalente composto de portas simples. Esta técnica é conhecida como macroexpansão [10][30] e, quando utilizada a título de pré-processamento, viabiliza o uso de qualquer ferramenta de FTA que tenha sido desenvolvida para tratar circuitos constituídos unicamente por portas simples. Entretanto, a macroexpansão apresenta dois inconvenientes [30]. Em primeiro lugar, é muito difícil modelar com precisão o atraso das portas complexas macroexpandidas. Em segundo lugar, a macroexpansão cria novos nós no circuito. Estes novos nós representam um potencial aumento no número de linhas que devem ser justificadas, no caso de FTA baseada em ATPG, ou num aumento do número de equações características, no caso de FTA baseada em SAT. Em qualquer um destes casos, o aumento no tempo de execução dependerá da complexidade das portas complexas e dos modelos de atraso utilizados para estas.

Uma segunda solução reside em modificar os testes de sensibilização de modo a torná-los aptos a tratar de circuitos que contenham portas complexas. Tal modificação refere-se não apenas ao critério de sensibilização, mas também ao algoritmo que testa a sensibilização. Desde que há mais de um algoritmo para testar a sensibilização de caminhos, esta solução pode ser desdobrada em várias soluções.

Em [30], por exemplo, é apresentado um algoritmo de FTA capaz de operar diretamente sobre circuitos com portas complexas. Com o intuito de evitar a macroexpansão, as condições para sensibilização exata no modo flutuante são estendidas, de modo a considerar portas complexas. Estas condições de sensibilização estendidas são utilizadas por um algoritmo baseado em sensibilização individual derivado do algoritmo D. Os resultados mostrados em [30] permitem comparar modelos de atraso para macroexpansão, bem como comparar o uso da macroexpansão com o algoritmo que testa diretamente portas complexas. Por outro lado, todos os resultados foram obtidos pelo uso de algoritmos baseados em sensibilização individual de caminhos, a qual, sabidamente, não representa o estado-da-arte por exibir o problema conhecido como "explosão de caminhos".

Em [31] é proposta uma extensão do procedimento de geração de teste temporal de Devadas et al. [16]. O procedimento de geração de teste temporal é um algoritmo de sensibilização concorrente de caminhos baseado em ATPG derivado do algoritmo PODEM [32]. No procedimento de geração de teste temporal, o problema de calcular o atraso de um

circuito é transformado num conjunto de geração de testes para falhas de colagem "temporais" imaginadas como ocorrendo nas saídas primárias do circuito. O procedimento tem então o objetivo de justificar tais falhas de colagem. Este algoritmo procura responder à seguinte pergunta: **o atraso do circuito é maior ou igual a δ?** O valor δ é inicializado com $T-\varepsilon_0$, onde T é o atraso topológico do circuito e ε_0 é um pequeno valor maior que zero. Para responder à pergunta, é empregado o método de simulação de cubos de entrada (*input cube simulation*) por meio de uma versão modificada de algoritmo PODEM [32], denominada de **procedimento de geração de teste temporal** (*timed-test generation procedure*) [16]. Enquanto a resposta à pergunta for "não", para cada uma das saídas primárias, o procedimento de geração de teste temporal é sucessivamente invocado para $\delta_i = T-\varepsilon_i$, com $i=0,1,2,....$, $\varepsilon_{i+1} > \varepsilon_i$. Se a resposta for "sim" para uma saída, então esta saída apresenta um atraso entre o valor corrente de δ (digamos $T-\varepsilon_k$) e o valor anterior ($T-\varepsilon_{k-1}$). Assim, $T-\varepsilon_{k-1}$ representa um limite superior seguro para o máximo atraso do circuito. Caso se deseje um valor mais preciso para o atraso, pode-se aplicar pesquisa binária sobre o intervalo [$T-\varepsilon_k$, $T-\varepsilon_{k-1}$].

Como o circuito pode estabilizar com o valor lógico 0 ou com o valor lógico 1, para cada valor δ_i, o procedimento de geração de teste temporal deve ser aplicado duas vezes a cada saída primária (exceto no caso em que a resposta for "sim" para o primeiro valor lógico testado). A figura 5 ilustra o procedimento básico do algoritmo TrueD-F.

A fim de determinar se o atraso máximo (atraso crítico) numa saída primária do circuito é maior ou igual a δ_i para o valor lógico **lvalue**, o procedimento de geração de teste temporal tenta justificar **lvalue** na saída considerada com atraso maior ou igual a δ_i. Isto é feito por meio de simulação de cubos, de maneira similar à realizada pelo algoritmo PODEM. Como o PODEM permite uma exploração sistemática e exaustiva do espaço das entradas, caso ele falhe em encontrar um cubo de entrada capaz de justificar lvalue na saída do circuito para o valor de atraso δ_i considerado, então a resposta à pergunta será "não" (para a saída testada e para o valor lógico **lvalue** considerado). Em outras palavras, o problema de determinar o atraso do circuito é transformado num problema de geração de teste para uma falha de colagem simples localizada na saída primária do circuito.

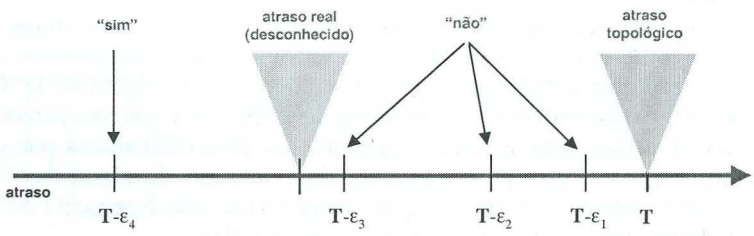

Figura 5 - Procedimento de geração de teste temporal aplicado a um circuito de uma única saída.

A chamada de mais alta hierarquia do procedimento de geração de teste temporal (timed_test) é descrita pelo pseudocódigo que segue (figura 6). Tal qual no algoritmo

PODEM puramente lógico, há uma lista de justificação **jlist** armazenando as linhas do circuito que precisam ser justificadas. O procedimento inicia pela inserção de uma dada saída primária po em **jlist** com valor lógico **lvalue** (0 ou 1) e assumindo *delay* como sendo o limite inferior do atraso do circuito a ser testado (δ_i). Então, o procedimento SEARCH_1 é chamado.

```
1       timed_test(po,delay,lvalue)
2       {
3          v.value = lvalue;
4          v.lower = delay;
5          v.upper = INFINITY;
6          modify_jlist (po,v,jlist);
7          backward schedule po;
8          status = SEARCH_1(jlist);
9          return(status);
10      }
```

Figura 6 - Pseudocódigo para a chamada de mais alta hierarquia do procedimento de geração de teste temporal.

As funções de busca são similares àquelas encontradas no algoritmo PODEM e estão descritas pelos pseudocódigos das figuras 7 e 8. A função SEARCH_1 chama a função BACKTRACE para, partindo da saída primária po, encontrar uma entrada primária cujo valor lógico ainda é desconhecido. A entrada primária identificada é inicialmente ajustada ao valor lógico 1 e então a função IMPLY é chamada. Esta, por sua vez, leva a cabo uma rodada de simulação de cubos para teste temporal considerando as condições de sensibilização do critério exato do modo flutuante (no PODEM original, que a partir daqui será referido por PODEM lógico, a função IMPLY corresponde à simulação de cubos com três valores lógicos e sem informação de atrasos). A função de implicação pode levar a uma situação de conflito, o qual pode ser de duas naturezas distintas: lógica ou temporal. Se não ocorrer conflito, SEARCH_1 será chamada recursivamente. O procedimento termina com sucesso em SEARCH_1 se a lista de justificação tornar-se vazia. No caso de ocorrência de conflito em SEARCH_1, o algoritmo retrocede ao assinalamento de entrada primária mais recente, assinalando-lhe o valor lógico 0. Então, a função SEARCH_2, mostrada na figura 8, é chamada. O procedimento BACKTRACE falha se suas funções forem incapazes de encontrar uma entrada primária livre de assinalamento ou se o espaço das entradas tiver sido completamente explorado sem sucesso em SEARCH_2. Ocorre conflito se não for possível atribuir a uma linha do circuito o valor lógico que lhe é requerido com o período de tempo necessário. Em caso de conflito ou falha, é necessário desfazer todos os assinalamentos originados do assinalamento de entradas que causou o conflito ou falha.

```
1       SEARCH_1(jlist)
2       {
3          if (length of jlist is zero) return SUCCEED;
4          if(BACKTRACE(gate,value,delay,&pi,&pi_value)==FALSE)
5             return(FAILED);
6          if(IMPLY(pi,pi_value,jlist)!=IMPLY_CONFLICT)
```

```
7       {
8           search_status = SEARCH_1(jlist);
9           if(search_status == FAILED)
10          {
11              restore the state of the network;
12              search_status = SEARCH_2(jlist,pi,1-pi_value);
13          }
14      }
15      else
16      {
17          restore the state of the network;
18          search_status = SEARCH_2(jlist,pi,1-pi_value);
19      }
20      return(search_status);
21  }
```

Figura 7 - Pseudocódigo para o primeiro procedimento de busca.

O procedimento de geração de teste temporal assume que cada porta do circuito (e também cada aresta) possui uma "variável lógico-temporal" formada por três campos: um limite inferior e um limite superior de atraso da porta (atrasos no contexto do circuito como um todo) e um valor lógico. Um passo de pré-processamento inicializa os campos de valores lógicos com o valor 2 (o qual indica que o valor lógico ainda não está determinado) e os limites inferior e superior de atraso com os mínimos e máximos atrasos topológicos computados a partir das entradas primárias.

```
1   SEARCH_2(jlist,pi,pi_value)
2   {
3       backtracks = backtracks + 1 ;
4       if(backtracks > BACKTRACK_LIMIT) return(ABORTED);
5       if(IMPLY(pi,pi_value,jlist)!=IMPLY_CONFLICT)
6       {
7           search_status = SEARCH_1(jlist);
8           if(search_status == FAILED)
9               restore the state of the network;
10      }
11      else
12      {
13          search_status = FAILED;
14          restore the state of the network;
15      }
16      return(search_status);
17  }
```

Figura 8 - Pseudocódigo para o segundo procedimento de busca.

Na fase de geração de teste, ao serem assinalados valores lógicos conhecidos às entradas primárias, os limites inferior e superior de atraso vão sendo aproximados para diante durante a simulação, devido à sensibilização ou bloqueio dos caminhos. Os limites inferior e

superior são também modificados pela retro-implicação, no momento em que novos valores são inferidos em algumas portas, como decorrência dos valores lógicos e respectivos tempos requeridos para as saídas primárias.

A chave para determinar se é possível ou não justificar um valor lógico numa saída do circuito para dado tempo reside na adoção de um cálculo temporal de três valores que leva em consideração as condições de sensibilização do critério exato do modo flutuante. Considere uma porta E de duas entradas com atraso d. Cada uma das entradas desta porta i_i pode apresentar um valor lógico pertencente ao conjunto {0,1,2} com limites inferior e superior de atraso dados por l_i e u_i, respectivamente. O termo "atraso do sinal" será utilizado, ao invés de "tempo de estabilização" [4], pois mesmo portas que apresentam valor lógico não assinalado (i.e., 2) em suas saídas, apresentam valores coerentes para os limites inferior e superior do atraso.

A fim de poder estender o procedimento de geração de teste temporal para circuitos que contenham portas complexas, é necessário generalizar o cálculo temporal para portas E/OU de n entradas. Para tanto, lança-se mão dos conceitos de valor controlante e valor não-controlante, conforme definidos no escopo de teste. Assim, dada uma porta g tipo E/OU de n entradas, podemos classificar seu estado lógico conforme os seguintes casos:

- Casos em que ao menos uma das entradas de g apresenta o valor controlante (c(g)). As demais entradas podem apresentar ou o valor não-controlante (nc(g)) ou o valor 2;
- Caso em que todas as entradas de g apresentam o valor não-controlante (nc(g));
- Casos em que ao menos uma das entradas de g apresenta o valor 2, mas nenhuma entrada apresenta o valor controlante (c(g)). As demais entradas podem apresentar o valor não-controlante (nc(g)).

A partir da identificação dos casos possíveis chega-se à generalização das regras, conforme mostrado na tabela 3. Também é possível expandir tais regras de modo a considerar a polaridade de saída das portas, o que permite tratar portas NÃO-E e NÃO-OU.

Tabela 3 -Regras generalizadas para o cálculo temporal de três valores para portas simples de *n* entradas.

grupo		regras	*Lv*
1	l_o	$\min\{ l_i \mid i=$ c(g) or $i=2 \} + d$	pol(g) ⊕ c(g)
	u_o	$\min\{ u_j \mid j=$ c(g) $\} + d$	
2	l_o	$\max\{ l_i \} + d$	pol(g) ⊕ nc(g)
	u_o	$\max\{ u_i \} + d$	
3	l_o	$\min\{ l_i \mid i=2 \} + d$	2
	u_o	$\max\{ u_i \} + d$	

As regras resumidas na tabela 3 podem ainda ser representadas de maneira gráfica, conforme mostram as figuras 9, 10 e 11. Por uma questão de simplicidade, tanto na tabela 3

como nas figuras 9, 10 e 11 assumiu-se um atraso único por porta. Modelos computacionais de atraso mais sofisticados serão discutidos mais adiante.

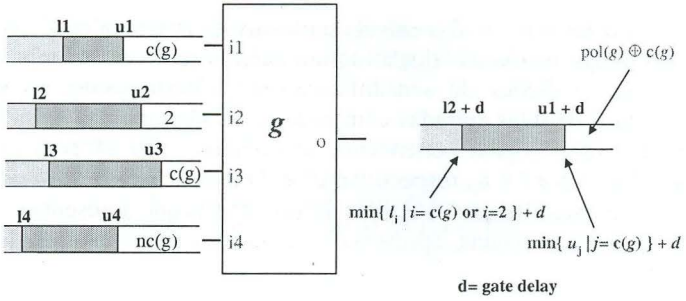

Figura 9 - Cálculo temporal de três valores para o grupo 1.

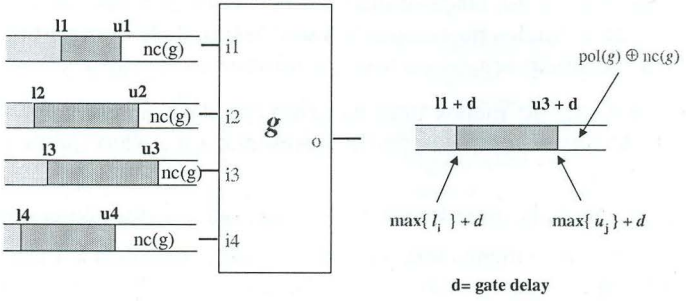

Figura 10 - Cálculo temporal de três valores para o grupo 2.

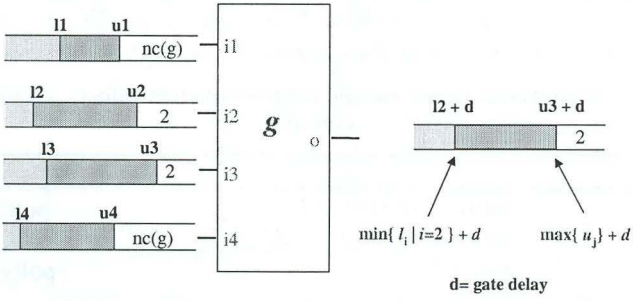

Figura 11 - Cálculo temporal de três valores para o grupo 3.

Assumindo-se que a função lógica de uma porta qualquer *g* esteja na forma fatorada e utilizando-se uma estrutura de dados apropriada para representá-la, a aplicação das regras generalizadas é direta. Considere, por exemplo, a SCCG mostrada na figura 12a. A função

lógica desta porta é dada por $S = \overline{A \cdot ((B \cdot C) + (D \cdot E))}$ e pode ser representada por uma "árvore da função" (figura 12c). Examinando-se esta árvore da função nota-se que, dado um assinalamento de valores temporais de entrada (valores lógicos e respectivos limites inferior e superior de atraso), os valores temporais na saída podem ser obtidos mediante a aplicação sucessiva das regras da tabela 3 para cada subárvore, iniciando-se pela subárvore mais próxima da base, desde que sejam feitas as seguintes assertivas:

1. Todas as subárvores, exceto a raiz, apresentam atraso de propagação zero e polaridade igual a zero
2. atraso de propagação da porta é aplicado somente sobre os limites inferior e superior de atraso da saída da porta, i.e., sobre os valores temporais resultantes da avaliação da subárvore de maior hierarquia.
3. A polaridade da porta é tratada somente quando a subárvore de maior hierarquia é processada.

A primeira e a segunda assertiva permitem que se divida a avaliação temporal da SCCG em dois passos independentes. No primeiro passo, o valor lógico da saída e os limites inferior e superior de atraso são computados para um atraso de propagação da porta igual a zero. Chamaremos este intervalo de atrasos de "limites de atraso de primeira ordem". Num segundo passo os limites inferior e superior reais são computados por meio da adição do atraso de propagação da porta aos limites de atraso de primeira ordem. Este segundo passo leva em conta o modelo computacional de atraso adotado para as portas.

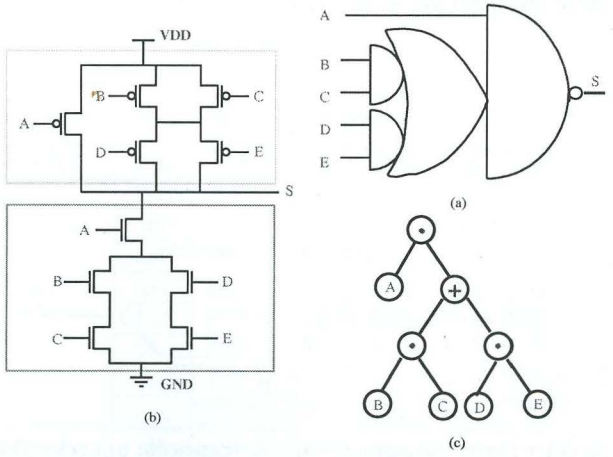

Figura 12 - Exemplo de SCCG: símbolo para o nível lógico (a), esquemático de transistores (b) árvore da função (c).

A figura 13 ilustra a aplicação das regras do cálculo temporal aplicadas à SCCG da figura 12.

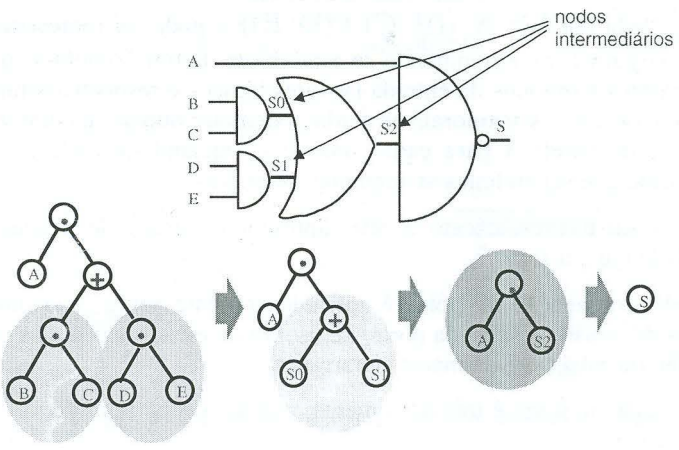

Figura 13 - Uso do cálculo temporal de três valores para avaliar uma SCCG.

Uma vez proposto um procedimento para determinar os limites de atraso de primeira ordem na saída de uma porta complexa, faz-se necessário investigar modelos computacionais de atraso que sejam válidos para o cálculo do atraso flutuante de um circuito. A fim de revelar a relação que existe entre o modelo computacional de atraso de circuitos e o modelo computacional de atraso de portas (e também, o modelo físico de atraso), consideremos a porta NÃO-E de 3 entradas mostrada na figura 14.

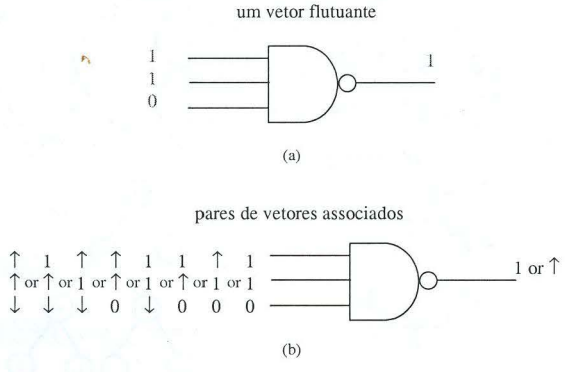

Figura 14 - Relação entre modo flutuante e modo de transição: um vetor flutuante aplicado a uma porta NÃO-E de 3 entradas (a) e os 8 pares de vetores associados (b).

Considere também que o vetor v=(a=1;b=1;c=0) seja aplicado a suas entradas. A fim de determinar o atraso flutuante desta porta NÃO-E sob o vetor v, é necessário examinar todas as combinações de entrada contidas em tal vetor, encontrando o pior atraso. Para iniciar, é necessário lembrar que no modo flutuante, um valor 0 representa tanto a transição

de descida quanto o valor 0 estático, enquanto que um valor 1 representa tanto a transição de subida quanto o valor 1 estático. Então, substituindo-se 0 por ↓ ou por 0 e substituindo-se 1 por ↑ ou por 1, obtém-se um total de 8 possibilidades de pares de vetores (figura 14b). Desta forma, estabelece-se uma relação entre o modo flutuante e o modo de transição, no que se refere ao modelo de atraso de portas lógicas.

Antes de continuar a discussão, é conveniente definir-se vetor flutuante e atraso flutuante. Um **vetor flutuante** é um assinalamento de valores lógicos aplicado às entradas de uma porta ou às entradas do circuito, quando assume-se que o circuito está operando no modo flutuante. O atraso computado para a porta (para o circuito) assumindo-se tal modo de operação é chamado **atraso flutuante** da porta (do circuito).

Dada a definição para vetor flutuante, vale lembrar que um vetor flutuante de n variáveis contém 2^n pares de vetores, dentre os quais um é composto por dois vetores iguais. A tabela 4 mostra todos os vetores flutuantes possíveis e os pares de vetores associados para uma porta NÃO-E de 3 entradas. Os pares hachureados em cinza claro são responsáveis por uma única transição de saída, enquanto que os pares hachureados em cinza escuro podem provocar uma transição espúria (*glicth*) na saída da porta.

Indo um pouco além, é possível criar-se uma tabela de equivalência entre modo flutuante e modo de transição como a tabela 4 para cada uma das 256 funções Booleanas de 3 variáveis. Desde que a associação entre vetores flutuantes e pares de vetores é fixa, as funções lógicas de 3 entradas se distinguiriam pelos valores lógicos resultantes na saída e também pelos pares de vetores que influenciam a computação do atraso da porta. Potencialmente, pode-se construir tabelas de equivalência para qualquer função lógica de n entradas.

Tabela 4 - Relação entre vetores do modo flutuantes e vetores do modo de transição.

vetor de entrada	000	001	010	011	100	101	110	111
pares de vetores associados	000	001	010	011	100	101	110	111
	↓00	↓01	↓10	↓11	↑00	↑01	↑10	↑11
	0↓0	0↓1	0↑0	0↑1	1↓0	1↓1	1↑0	1↑1
	00↓	00↑	01↓	01↑	10↓	10↑	11↓	11↑
	↓↓0	↓↓1	↓↑0	↓↑1	↑↓0	↑↓1	↑↑0	↑↑1
	↓0↓	↓0↑	↓1↓	↓1↑	↑0↓	↑0↑	↑1↓	↑1↑
	0↓↓	0↓↑	0↑↓	0↑↑	1↓↓	1↓↑	1↑↓	1↑↑
	↓↓↓	↓↓↑	↓↑↓	↓↑↑	↑↓↓	↑↓↑	↑↑↓	↑↑↑
valor lógico da saída	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**0**

Uma contribuição da tabela de equivalência reside no fato de que ela permite a identificação dos modelos computacionais de atraso para portas a serem usados para calcular o atraso flutuante de um circuito. Mais especificamente, a tabela diz que uma porta pode apresentar ao menos um atraso distinto por vetor flutuante de entrada. De maneira mais conservadora, pode-se considerar que o modelo computacional de atraso de porta mais

detalhado para o modo flutuante corresponde a assumir-se um valor de atraso por vetor flutuante. Tal modelo será referenciado por **modelo de atraso de vetor**.

Um modelo menos preciso, porém ainda correto do ponto de vista da análise de *timing*, corresponde a agrupar os vetores flutuantes de entrada de acordo com o tipo de transição de saída da porta. Assim, a cada porta é assinalado um atraso de descida, correspondendo ao máximo atraso dentre os vetores flutuantes que provocam uma transição de descida, e um atraso de subida, correspondendo ao máximo atraso dentre os vetores flutuantes que provocam uma transição de subida. Tal modelo será chamado **modelo de atraso de subida/descida**.

Um terceiro modelo corresponde a assinalar um único valor de atraso por porta, chamado de **modelo de atraso único**. Naturalmente, este é o modelo menos preciso, embora ainda correto.

É importante ressaltar que, para obter-se um cálculo robusto de atraso flutuante do circuito, os valores de atraso de portas a serem usados em cada um dos três modelos deve corresponder ao limite superior para o atraso da porta, considerando-se todas as instâncias do circuito fabricado. Isto significa que o modelo físico de atraso usado ou o método de estimativa de atraso deve ser capaz de fornecer o valor correspondente ao pior caso individualmente para cada componente.

Uma análise mais cuidadosa da tabela 4 permite ainda concluir que o formato de atraso pino-a-pino, normalmente utilizado na caracterização de *standard-cells*, não é apropriado para a computação do atraso flutuante desde que considera somente parte de todos os vetores flutuantes de entrada. Por exemplo, no caso de uma porta de 3 entradas, o formato pino-a-pino cobre apenas seis pares de vetores: $\downarrow 11$, $1\downarrow 1$, $11\downarrow$, $\uparrow 11$, $1\uparrow 1$, $11\uparrow$. Ou seja, apenas as situações em que ocorre uma única transição de entrada.

Uma segunda contribuição da tabela de equivalência entre modo flutuante e modo de transição é a própria associação entre um dado vetor flutuante e os pares de vetores subscritos. Tal informação é útil quando deseja-se caracterizar o atraso de uma porta, sobretudo se o método de caracterização for simulação elétrica. Neste caso, apenas os pares de vetores que podem produzir uma transição de saída devem ser simulados, o que pode reduzir significativamente o número de casos. Por outro lado, no caso de caracterização por meio de formulação analítica, a informação da tabela de equivalência pode ser usada para ajustar ou adaptar o conjunto de equações de modo a cobrir apenas os pares de vetores de interesse.

Tendo derivado regras seguras para determinar o atraso das portas sob qualquer um dos três modelos computacionais de atraso de porta, faz-se necessário considerar a aplicação de tais regras no cálculo dos limites inferior e superior de atraso de uma porta usando o procedimento de implicação para diante. Retomando as regras do cálculo temporal de três valores descritas na tabela 3, as situações possíveis de vetores flutuantes podem ser classificadas nos três grupos que seguem:

1 - Casos em que ao menos uma das entradas de g apresenta o valor controlante ($c(g)$). As demais entradas podem apresentar ou o valor não-controlante ($nc(g)$) ou o valor 2;

2 - Caso em que todas as entradas de g apresentam o valor não-controlante (nc(g));

3 - Casos em que ao menos uma das entradas de g apresenta o valor 2, mas nenhuma entrada apresenta o valor controlante (c(g)). As demais entradas podem apresentar o valor não-controlante (nc(g)).

Os grupos 1 e 3 referem-se aos casos onde uma das entradas da porta apresenta o valor 2. No contexto do cálculo de três valores, um 2 representa a existência ou do valor lógico 0 ou do valor lógico 1. Em decorrência disto podemos introduzir a definição de cubo flutuante: dado o cálculo de três valores, um **cubo flutuante** é um assinalamento de valores lógicos tal que ao menos um destes corresponde ao valor 2. O número de vetores flutuantes contidos em um cubo flutuante é igual a 2^m, onde m é o número de posições que apresentam o valor 2.

Com efeito, um cubo flutuante contém ao menos dois vetores flutuantes e desta forma, pode ser visto como uma variação do modelo de atraso de subida/descida. A fim de garantir uma máxima precisão, é necessário considerar o máximo e o mínimo atrasos da porta sob um dado cubo. A partir de tal assertiva, o atraso de uma porta sob um cubo terá uma forma invariante, independentemente do modelo computacional de atraso de porta. Para efeitos ilustrativos, considere a aplicação do cubo flutuante 022 à SCCG da figura 15, a qual implementa a função lógica $S = \overline{A + B \cdot C}$.

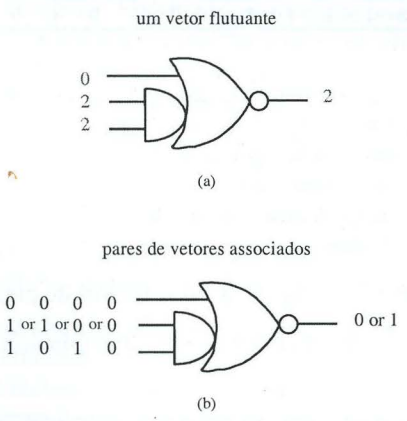

Figura 15 - Atraso de uma SCCG sob um cubo flutuante.

A tabela 5 mostra a equivalência entre o modo flutuante e o modo de transição para tal porta. Note que a saída desta SCCG será 1 quando o vetor flutuante aplicado as suas entradas pertencer a {000, 001, 010} e será 0 quando o vetor flutuante aplicado as suas entradas pertencer a {011, 100, 101, 110, 111}. Se o procedimento de cálculo do atraso assumir o modelo de atraso de vetor, então o atraso desta SCCG para o cubo 022 terá máximo e mínimo limites de atraso dados por max{d(000),d(001),d(010),d(011)} e min{d(000),d(001),d(010),d(011)}, respectivamente, onde d(000), d(001), d(010) e d(011)

são os atrasos sob os vetores flutuantes 000, 001, 010 e 011. Por outro lado, se o procedimento de cálculo do atraso assumir o modelo de atraso de subida/descida, então o máximo e o mínimo limites de atraso serão dados por $\max\{tp_{LHma}, tp_{HLmax}\}$ e $\min\{tp_{LHmin}, p_{HLmin}\}$. Entretanto, para esta SCCG, $tp_{LHmax}=\max\{d(000),d(001),d(010)\}$ e $tp_{LHmin}=\min\{d(000),d(001),d(010)\}$, enquanto que $tp_{HLmax}=tp_{HLmin}=d(011)$, resultando nos mesmos valores de máximo e mínimo atrasos fornecidos pelo modelo de atraso de vetor. De forma análoga, pode-se declarar que o atraso flutuante deste cubo para o modelo de atraso subida/descida irá resultar nos mesmos valores de máximo e mínimo atrasos calculados pelos outros dois modelos de atraso de porta.

Tabela 5 - Equivalência entre modo flutuante e modo de transição para a SCCG da figura 15.

vetor de entrada	000	001	010	011	100	101	110	111
pares de vetores associados	000	001	010	011	100	101	110	111
	↓00	↓01	↓10	↓11	↑00	↑01	↑10	↑11
	0↓0	0↓1	0↑0	0↑1	1↓0	1↓1	1↑0	1↑1
	00↓	00↑	01↓	01↑	10↓	10↑	11↓	11↑
	↓↓0	↓↓1	↓↑0	↓↑1	↑↓0	↑↓1	↑↑0	↑↑1
	↓0↓	↓0↑	↓1↓	↓1↑	↑0↓	↑0↑	↑1↓	↑1↑
	0↓↓	0↓↑	0↑↓	0↑↑	1↓↓	1↓↑	1↑↓	1↑↑
	↓↓↓	↓↓↑	↓↑↓	↓↑↑	↑↓↓	↑↓↑	↑↑↓	↑↑↑
valor lógico da saída	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**0**

Analisando novamente as tabelas de equivalência, é possível afirmar que o atraso de qualquer porta depende apenas do cubo (ou vetor) aplicado às suas entradas. Particularmente no caso do modelo subida/descida, pode-se afirmar que o atraso de uma porta fica perfeitamente definido pelos grupos aos quais o vetor (ou cubo) flutuante de entrada pertence. Tal conclusão ratifica a decisão de dividir o cálculo do valor temporal de uma porta em duas subtarefas independentes:

- Cálculo do valor lógico da saída e dos limites de primeira ordem
- Identificação do atraso flutuante da porta e aplicação deste no cálculo dos limites de atraso reais.

A partir desta verificação, torna-se claro que os recursos necessários para realizar-se a computação do valor temporal para portas complexas arbitrárias já foram propostos. Estes recursos são:

- A árvore da função e os procedimentos necessários para computar o valor lógico na saída das portas e os limites de atraso de primeira ordem;
- O uso de um dentre os três modelos de cálculo de atraso de porta (atraso de vetor, atraso de subida/descida ou atraso único) para calcular os limites inferior e superior do atraso das portas.

Uma vez detalhado o procedimento para realizar-se a implicação para diante, resta-nos investigar o procedimento de retro-implicação.

Dada uma porta e um valor lógico desejado para a sua saída, a retro-implicação consiste em encontrar um assinalamento de entradas capaz de produzir tal valor. Obviamente, o valor lógico desejado na saída da porta é um valor conhecido, ou seja, 0 ou 1. Ocorre conflito quando a saída de alguma porta apresenta valor lógico oposto ao que se deseja.

No caso da retro-implicação temporal, não basta verificar a ocorrência de conflitos lógicos. Para cada entrada de cada porta, é necessário verificar se os limites inferior e superior de atraso requeridos estão dentro dos limites inferior e superior de atraso pré-existentes. Dada uma porta simples e um valor lógico desejado para sua saída, o procedimento original de geração de teste temporal procura pelo cubo capaz de produzir tal valor lógico explorando os conceitos de valor controlante/valor não controlante. Considere que se deseje colocar a saída de uma porta E no valor lógico 0. Como 0 é o valor controlante da porta E, existe mais de uma possibilidade de vetores capazes de satisfazer a condição desejada. Assim, o procedimento procura pela entrada capaz de produzir um 0 na saída da porta e que seja a mais rápida. Se tal entrada não existe, então não é possível implicar para trás o valor lógico 0 pela porta considerada.

No caso das portas complexas, os conceitos de valor controlante/valor não controlante não podem ser aplicados diretamente. Entretanto, percebe-se que implicar para trás um valor lógico por uma porta pode corresponder a um "processo de adivinhação", no qual se tenta descobrir um assinalamento de entradas capaz de produzir o valor lógico de saída desejado. Neste sentido, o procedimento de retro-implicação para portas complexas vem a ser uma generalização do caso das portas simples. A heurística usada pelo procedimento de geração de teste temporal na retro-implicação com portas simples reside em "setar" o menor número de entradas da porta, pois isto resultará num menor número de linhas do circuito necessitando justificação. Tal procedimento pode ser visto como uma exploração sistemática do subespaço Booleano local (i.e., espaço Booleano associado à função realizada pela porta considerada), ao mesmo tempo em que uma variável por vez é "setada".

O raciocínio desenvolvido acima sugere o uso de um procedimento de enumeração sistemática do espaço de entradas juntamente com o cálculo temporal de três valores apresentado na tabela 3 para realizar a retro-implicação lógica em portas complexas. A figura 16 exemplifica o procedimento de retro-implicação quando se deseja o valor lógico 0 na saída da SCCG da figura 15. Para tanto, inicia-se colocando o valor lógico 0 na entrada superior. Implicando-se para diante descobre-se que a saída da SCCG mantém-se com valor 2 (figura 16a). Isto significa que é necessário "setar" uma ou mais entradas. Assinalando-se o valor 0 à entrada do centro resulta que a saída da SCCG terá valor lógico 1 (figura 16b), o que corresponde a um conflito. Então, retrocede-se ao estado anterior, substituindo o valor da entrada central pelo seu oposto, i.e., 1. A implicação desta nova situação de entradas resulta no valor 2 na saída da SCCG (figura 16c). Colocando o valor 0 na entrada inferior resulta no valor lógico 1 na saída (figura 16d), o que corresponde a um conflito. Finalmente, trocando-se o valor da entrada inferior para 1 obtém-se o valor lógico 0 desejado para a saída.

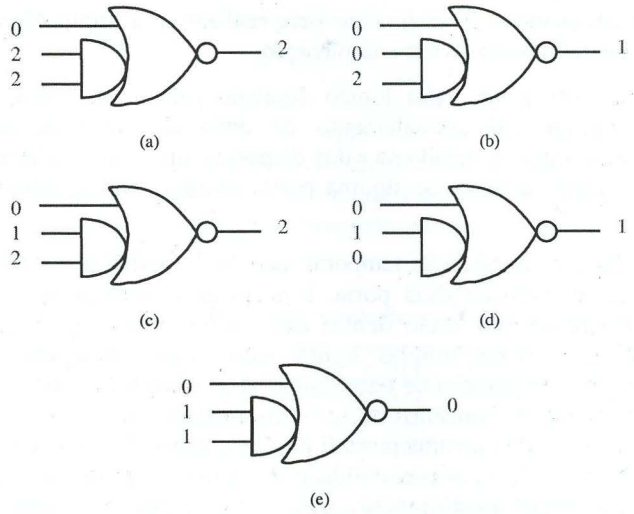

Figura 16 - Retro-implicação aplicada à SCCG da figura 15.

A solução para retro-implicação com portas complexas descrita acima equivale a aplicar o algoritmo PODEM a uma única porta, o que exige tempo de execução reduzido desde que se restrinja o número de entradas da porta. Entretanto, o aspecto mais importante de tal solução reside no fato dela permitir o uso das regras do cálculo temporal de três valores e da árvore da função propostos neste trabalho. O único recurso extra de que se necessita é um mecanismo de enumeração que permita o controle do subespaço referente à porta cuja saída se deseja retro-implicar.

Uma vez encontrado um procedimento para realizar a retro-implicação lógica em portas complexas, deve-se pensar em como estendê-lo à retro-implicação temporal. Similarmente à implicação temporal para diante, pode-se compor tal procedimento assumindo-se os seguintes passos:

1. Retro-implicação lógica
2. Cálculo dos limites de atraso de primeira ordem e detecção de conflito.

O primeiro passo já foi descrito nos parágrafos anteriores. O segundo passo pode ainda ser subdividido em:

1. Determinar o atraso da porta sob o cubo de entrada k selecionado (i.e., sob o assinalamento escolhido para as entradas da porta)
2. A partir dos limites inferior e superior da saída, calcular os limites inferior e superior de primeira ordem para cada entrada da porta usando as seguintes fórmulas: $l_o' = l_o - d(k_{max})$ e $u_o' = u_o - d(k_{min})$, onde $d(k_{max})$ e $d(k_{min})$ correspondem ao máximo e mínimo atrasos para o cubo k assinalado à entrada da porta, respectivamente.

3. Para cada entrada i da porta, calcular a intersecção entre os limites inferior e superior de atraso da entrada e os limites inferior e superior de primeira ordem calculados no passo anterior.

No passo 3, um conflito é detectado se $l_o' < l_i$ ou se $u_o' < u_i$. Note que pode ocorrer conflito ao se verificar a entrada de uma porta (passo 3 anterior) ainda que a mesma apresente valor lógico 2. Neste caso, seria um conflito temporal.

Uma vez propostos procedimentos que realizam a implicação para diante e a retro-implicação sobre portas complexas, o procedimento de geração de teste temporal foi suficientemente generalizado de modo a permitir a análise de circuitos que contenham portas complexas.

É importante notar que o método proposto para realizar o cálculo temporal de três valores sobre portas complexas não irá representar significativo aumento no tempo total de execução uma vez que, por razões tecnológicas, os projetistas costumam limitar a 4 o número de transistores em série nas portas CMOS.

5 Conclusão

Este artigo apresentou uma revisão sistemática sobre modelos e algoritmos de análise de *timing*. Com o intuito de facilitar a compreensão dos diversos aspectos da FTA e foi proposto uma nova taxonomia para a classificação dos algoritmos de FTA.

Ainda com relação à teoria de FTA, foi proposto que os circuitos síncronos implementados com o uso da tecnologia CMOS corrente podem apresentar um comportamento assíncrono que não é apropriadamente capturado pelo modelo de atraso de transição. Isto significa que, a fim de evitar uma subestimativa do atraso do circuito, seria preferível calcular o atraso para seqüências de vetores do que para pares de vetores. Por outro lado, na ausência de métodos eficientes capazes de utilizar o modelo de atraso para seqüências de vetores, o modelo de atraso flutuante seria o único capaz de fornecer uma estimativa segura para o atraso dos circuitos. Esta argumentação também justifica o uso da análise de *timing* funcional, uma vez que esta baseia-se essencialmente no modelo de atraso flutuante.

No que se refere à FTA com portas complexas, escolheu-se o algoritmo TrueD-F, proposto por Devadas et al., para servir de base para um algoritmo de sensibilização concorrente baseado em ATPG e que fosse capaz de tratar circuitos que contenham portas complexas, sem requerer macro-expansão. Esta escolha foi motivada pela existência de inúmeras técnicas de aceleração de algoritmos de ATPG. Outro argumento que sustenta a escolha de um algoritmo baseado em ATPG advém dos problemas de desempenho apresentados pelos algoritmos baseados em SAT, quando modelos de atraso mais realistas são adotados.

Também foi proposta uma extensão do algoritmo TrueD-F a fim de permitir o tratamento de circuitos contendo portas complexas, sem o uso de marco-expansão. De

acordo com o método proposto, o cálculo dos valores temporais de uma porta complexa é levado a cabo por um procedimento recursivo que avalia cada subárvore, iniciando pelas folhas. Para um máximo de 4 transistores em série na rede PMOS e na rede NMOS, o número máximo de subárvores que qualquer SCCG pode apresentar é 11, incluindo a subárvore raiz. Considerando-se que a avaliação de uma única subárvore consiste de um cálculo do valor lógico e dos intervalos de atraso, pode-se esperar que operando-se diretamente sobre portas complexas resulte em tempos de execução que serão, na pior das hipóteses, equivalentes aos tempos de execução necessários para um método que fizesse uso de macro-expansão. Entretanto, vale mencionar que a aplicação de um algoritmo baseado em ATPG a uma rede macro-expandida pode levar a um aumento significativo no tempo de execução, uma vez que haverá forte tendência de aumento no número de linhas do circuito, e portanto, aumento no número de linhas que devem ser justificadas.

Finalmente, a extensão proposta para o cálculo temporal, juntamente com os modelos de atraso válidos para o modo flutuante, formam um conjunto de regras de macro-modelamento que abrem a possibilidade para realizar-se uma "computação hierárquica do atraso flutuante" de circuitos. Esta é a conseqüência de que o conceito de porta complexa constitui-se na representação genérica de qualquer porção possível de circuitos CMOS.

Referências

[1] HRAPCENKO, V. Depth and Delay in a Network. **Soviet Math. Dokl.** v.19, n.4, p.1006-1009, 1978.

[2] KEUTZER, K.; MALIK, S.; SALDANHA, A. Is Redundancy Necessary to Reduce Delay? **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, Los Alamitos, California, v.10, n4, p.427-435, April 1991.

[3] LAM, W.; BRAYTON, R. **Timed Boolean Functions:** A Unified Formalism for Exact Timing Analysis. Norwell, MA: Kluwer Academic Publishers, 1994. 273p.

[4] CHEN, H.-C.; DU, D. Path Sensitization in Critical Path Problem **IEEE Transanctions on CAD of Integrated Circuits and Systems**, Los Alamitos, California, v.12, n.2, p.196-207, February 1993.

[5] DEVADAS, S.; KEUTZER, K.: MALIK, S.; WANG, A. Certified Timing Verification and the Transition Delay of a Logic Circuit. In: ACM/IEEE DESIGN AUTOMATION CONFERENCE, 29., 1992, Anaheim, California. **Proceedings...** Los Alamitos, California: IEEE Computer Society Press, 1992. p.549-555.

[6] SILVA, Luís Jorge B. M. G. e. **Models and Algorithms for Timing Analysis of Combinational Circuits.** Lisboa: Instituto Superior Técnico (IST), Universidade Técnica de Lisboa, 1999. 84p. Dissertação de mestrado.

[7] CHEN, H.-C.; DU, D. Path Sensitization in Critical Path Problem. In: IEEE/ACM INTERNATIONAL CONFERENCE ON COMPUTER-AIDED DESIGN, 1991, Santa Clara, California. **Proceedings...** Los Alamitos, California: IEEE Computer Society Press, 1991. p. 208-211.

[8] OUSTERHOUT, John K. A Switch-Level Timing Verifier for Digital MOS VLSI, **IEEE Transactions on Computer-Aided Design**, Los Alamitos, California, v. CAD-4, n. 3, July 1985, p.336-349.

[9] JOUPPI, Norman P. Timing Analysis and Performance Improvement of MOS VLSI Designs. **IEEE Transactions on Computer-Aided Design**, Los Alamitos, California, v.CAD-6, n.4, p.650-665, July 1987.

[10] MCGEER, P.; BRAYTON, R. **Integrating Functional and Temporal Domains in Logic Design**: The False Path Problem and its Implications. Norwell, MA: Kluwer Academic Publishers, 1991. 212p.

[11] ROTH, J.P. Diagnosis of Automata Failures: A Calculus and a New Method In: **IBM Journal of Research and Development**, p.278-281, Oct.1966.

[12] BENKOSKI, J.; VAN DEN MEERSCH, E.; CLAESEN, L.; DE MAN, H. Timing Verification Using Statically Sensitizable Paths. **IEEE Transanctions on CAD of Integrated Circuits and Systems**, Los Alamitos, CA, v. 9, n. 10, p. 1073-1084, Oct. 1990.

[13] BRAND, D.; IYENGAR, V. Timing Analysis Using Functional Analysis **IEEE Transactions on Computers** v.37 n.10 p.1309-1314, October 1988

[14] KEUTZER, K.; MALIK, S.; SALDANHA, A. Is Redundancy Necessary to Reduce Delay? **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, Los Alamitos, California, v.10, n4, p.427-435, April 1991.

[15] DEVADAS, S.; GHOSH, A.; KEUTZER, K. **Logic Synthesis.** New York: McGraw-Hill, 1994. 404p.

[16] DEVADAS, S.; KEUTZER, K.: MALIK, S.; WANG, A. Computation of Floating Mode Delay in Combinational Circuits: Practice and Implementation. **IEEE Transactions on Computed-Aided Design of Integrated Circuits and Systems**, Los Alamitos, California, v.12, n.12, p.1924-1936, December 1993.

[17] SILVA, João P.M.; SAKALLAH, Karem. Efficient and Robust Test Generation-Based Timing Analysis. In: IEEE INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS, 1994, London. **Proceedings...** Piscataway: IEEE 1994. p. 303-306 v.1.

[18] CHANG, Hoon; ABRAHAM, Jacob A. VIPER: An Efficient Vigorously Sensitizable Path Extractor. In: ACM/IEEE DESIGN AUTOMATION CONFERENCE, 30., 1993, Dallas, Texas. **Proceedings....** Los Alamitos, California: IEEE Computer Society Press, 1993. p.112-117.

[19] MCGEER, P.; BRAYTON, R. Efficient Algorithms for Computing the Longest Viable Path in a Combinational Circuit In: ACM/IEEE DESIGN AUTOMATION CONFERENCE, 26., 1989, Las Vegas, Nevada. **Proceedings...** Los Alamitos, California: IEEE Computer Press, 1989. p.561-567.

[20]LARRABEE, T. Test Pattern Generation Using Boolean Satisfiability. **IEEE Transanctions on CAD of Integrated Circuits and Systems**, Los Alamitos, California, v.11, n.1, p. 4-15, Jan. 1992.

[21] MCGEER, P. et al. Timing Analysis and Path Delay-Fault Test Generation using Path-Recursive Functions. In: IEEE/ACM INTERNATIONAL CONFERENCE ON COMPUTER-AIDED DESIGN, 1991, Santa Clara, California. **Proceedings...** Los Alamitos, California: IEEE Computer Society Press, 1991. p.180-183.

[22]MCGEER, P.; et al. Delay Models and Exact Timing Analysis. In: SASAO, T. (Ed.). **Logic Synthesis and Optimization**. Norwell, MA: Kluwer Academic Publishers, 1993. p. 167-189.

[23] SILVA, João P.M.; SAKALLAH, Karem. Concurrent Path Sensitization in Timing Analysis. In: THE EUROPEAN CONFERENCE ON DESIGN AUTOMATION, 1993, Paris, France. **Proceedings...** Los Alamitos, California: IEEE Computer Society Press, 1993.

[24] SILVA, João P.M.; SAKALLAH, Karem. GRASP – A New Search Algorithm for Satisfiability. In: IEEE INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS, 1996, San Jose, California. **Proceedings...** Los Alamitos, California: IEEE Computer Society Press, 1996. p. 220-227.

[25] CADABRA. **CLASSIC-SC Automated Transistor Layout Tool**. Disponível em http://www.cadabradesign.com (24/9/1999).

[26] MORAES, F.; REIS, R.; LIMA F. An Efficient Layout Style for Three-Metal CMOS Macro-Cells. In: Reis, R. and Claesen, L. (Editors) **VLSI: Integrated Systems on Silicon**. Chapman & Hall, 1997. p. 415-426.

[27] REIS, André I. et al. Library Free Technology Mapping. In: Reis, R. and Claesen, L. (Editors) **VLSI: Integrated Systems on Silicon**. Chapman & Hall, 1997. p. 303-314.

[28] REIS, André I. **Assignation Technologique sur Biblioteques Virtuelles de Portes Complexes CMOS**. Montpellier: Université Montpellier II, 1998. 122p. Tese de doutorado.

[29] DETJENS, E.; GANNOT, G.; RUDELL, R. L. Technology Mapping in MIS. In: IEEE/ACM INTERNATIONAL CONFERENCE ON COMPUTER-AIDED DESIGN, 1987. **Proceedings...** Los Alamitos, California: IEEE Computer Society Press, 1987. p.116-119.

[30] HSU, Y.-C.; CHEN, H.-C.; SUN, S.; DU, D. Timing Analysis of Combinational Circuits Containing Complex Gates. In: INTERNATIONAL CONFERENCE ON COMPUTER DESIGN: VLSI IN COMPUTERS AND PROCESSORS, 1998, Austin, Texas. **Proceedings...** Los Alamitos, California: IEEE, 1998. p.407-412

[31] GÜNTZEL, José L. **Análise de *Timing* Funcional de Circuitos VLSI Contendo Portas Complexas.** Porto Alegre, PGCC da UFRGS, novembro de 2000. 182p. Tese de doutorado.

[32] GOEL, Prabhakar. An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits. **IEEE Transactions on Computers**, Los Alamitos, California, v.C-30, n.3, p. 215-222, Mar. 1981.