Capitolo 8

Simulazione di guasti in circuiti digitali

8.1 Introduzione

La "simulazione di guasti" (*Fault Simulation* - FS) rappresenta una delle due operazioni[1] fondamentali del CAD dedicato al collaudo (*testing*) dei circuiti integrati digitali [1]: in particolare, essa consente di valutare la qualità delle sequenze di stimoli da applicare ai circuiti per verificarne il corretto funzionamento.

Come noto, a causa dell'impossibilità di una verifica esaustiva della funzionalità di un circuito digitale, si assume che ogni deviazione dal comportamento corretto sia dovuta a malfunzionamenti interni al circuito tipicamente dovuti a guasti di tipo fisico o ad errori nel processo di fabbricazione e produzione del dispositivo. La FS consiste, appunto, nella simulazione di un circuito in presenza di un insieme prefissato di guasti e da uno di stimoli di ingresso (vettori di *test*) allo scopo di determinare quali guasti diano luogo a differenze (errori) nel comportamento osservabile del circuito rispetto al caso corretto. Qualora uno di tali guasti sia effettivamente presente, esso verrà quindi rivelato durante il collaudo proprio dalla presenza di di almeno un errore nella risposta del circuito.

Le considerazioni che precedono definiscono implicitamente alcuni aspetti fondamentali della FS: a) la FS richiede lo sviluppo di "modelli di guasto" (*fault models*) che debbono essere necessariamente congruenti con il livello (elettrico, logico, ...) di astrazione utilizzato per descrivere il circuito e con la tecnologia con cui questo è stato realizzato; b) le deviazioni dal comportamento corretto utilizzabili per la rivelazione dei guasti possono riguardare diversi aspetti del funzionamento del circuito (valore logico delle uscite, assorbimento di corrente, prestazioni dinamiche del circuito, ...).

[1] L'altra operazione fondamentale è quella della "generazione automatica" dei vettori di collaudo (*Automatic Test Pattern Generation* - ATPG).

Ciascuno di questi aspetti, in generale, è adatto per trattare tipi diversi di guasti e, almeno in linea di principio, richiede modelli e simulazioni specifiche. In particolare, ad esempio, guasti che si manifestano come eccessivi ritardi nella propagazione dei segnali richiedono simulatori in grado di tenere in conto le temporizzazioni all'interno del circuito, mentre quelli che cambiano il valore logico delle uscite richiedono solamente di conoscere il valore delle linee di segnale nel loro stato stabile.

8.2 Scopi della simulazione di guasti

Per un assegnato insieme di guasti (diversi per modello e/o posizione nel circuito considerato), la FS consente di determinare quale frazione di essi possa essere rivelata mediante un assegnato insieme di vettori di collaudo. In altri termini, essa consente di ottenere la "copertura" (*fault coverage*) dell'insieme di guasti considerato, quantificando così l'efficacia dell'insieme di vettori di collaudo. In particolare, partendo da questa cifra di merito è possibile determinare la "resa" del processo di collaudo [2], ovvero la frazione di circuiti effettivamente guasti riconosciuti come tali in fase di collaudo (infatti, se la copertura non è quella massima, è possibile che un circuito guasto passi il collaudo).

Inoltre, durante la generazione deterministica[2] dei vettori di *test*, la conoscenza di quali guasti sono rivelati ad ogni passo del processo è estremamente importante allo scopo di evitare la ricerca di vettori ridondanti. Infatti, un vettore di collaudo generato per uno specifico guasto è in grado, in generale, di rivelarne anche altri, che possono essere determinati mediante la FS e così eliminati dal processo di ATPG.

Nel caso della generazione di vettori di *test* di tipo (pseudo-)casuale[3] [3], invece, la FS viene utilizzata per determinare quando i prefissati obbiettivi di copertura sono stati raggiunti e, di conseguenza, non è più necessario generare ulteriori vettori di collaudo.

Infine, vale la pena di osservare come la FS svolga un ruolo importante anche per quanto riguarda la diagnosi di circuiti malfunzionanti, in quanto rappresenta lo strumento essenziale per studiare la relazione fra gli effetti (errori) riscontrati nel comportamento del circuito e le possibili cause (cioè guasti opportunamente modellati).

[2] Ovvero basata su programmi di ATPG che, assegnato un guasto, siano in grado di generare (se esiste) un vettore di collaudo in grado di rivelarlo.

[3] Con questo metodo le sequenze di vettori di collaudo con proprietà statistiche che approssimano quelle di una distribuzione uniforme, in cui, in particolare, la probabilità di ciascuna linea di ingresso del circuito di essere al valore logico alto è pari a 0.5. Poichè queste sequenze possono essere generate mediante circuiti molto compatti ottenuti da registri a scorrimento (*Linear Feedback Shif Register -LFSR*), esse sono utilizzate nei circuiti in grado di autocollaudarsi (*Built-In Self-Test - BIST*).

8.3 Scelta del livello di descrizione del circuito e dei relativi modelli di guasto

Come anticipato, qualunque strategia di FS implica la scelta del livello di descrizione del circuito e dei modelli di guasto da utilizzare.

A proposito del primo punto, va detto che la FS viene prevalentemente eseguita a livello logico (a sua volta, divisbile nei livelli *gate*, *switch* e *behavioral*), anche se la simulazione a livello elettrico viene talvolta utilizzata in fase di sviluppo dei modelli di guasto al fine di verificarne la validità e l'accuratezza.

Nell'ambito della simulazione a livello logico, nel seguito faremo prima implicitamente riferimento a quella a livello *gate*, che rappresenta la più largamente utilizzata in pratica, in cui il circuito viene descritto mediante porte logiche e linee di interconnessione con segnali che possono assumere uno dei due valori logici alto e basso.

Per ciò che riguarda i modelli di guasto, va osservato che il numero dei possibili guasti è tale da rendere necessarie drastiche semplificazioni al fine di rendere il problema affrontabile praticamente. La più importante di queste semplificazioni è quella di considerare unicamente guasti singoli (cioè in grado di verificarsi solamente uno per volta) e di carattere permanente.

In questo ambito, il modello di guasto più utilizzato è quello di nodo bloccato (*stuck-at-0/1*), ovvero di una linea di segnale che mantiene lo stesso valore logico indipendentemente dal funzionamento del circuito. Questo modello presenta numerosi vantaggi, oltre a quello della semplicità, in quanto è in grado di rappresentare una vasta gamma di malfunzionamenti, anche molto diversi da un punto di vista fisico, accomunati dalla proprietà di alterare la risposta funzionale delle porte logiche (cioè il valore logico dei segnali di uscita nel loro stato stabile). Per questo motivo esso ha avuto un enorme successo ed è, in pratica, il solo utilizzato attualmente nel collaudo di circuiti digitali.

Tuttavia, esso non rappresenta affatto tutte le possibilità di guasto nei circuiti digitali [4] che, dipendentemente dalla loro tecnologia ed architettura, possono manifestare malfunzionamenti non rappresentabili in termini di nodi bloccati. Tali malfunzionamenti, pertanto, richiedono modelli specifici e ciò può condurre alla necessità di simulazioni adeguate: ad esempio, i guasti di tipo ritardo (*delay faults*) [5,6] richiedono simulazioni in grado di rappresentare il comportamento dinamico (ovvero i ritardi di propagazione) del circuito.

Ciò è vero, in particolare, per i circuiti CMOS, in cui alcune importanti cause di malfunzionamento possono essere descritte con modelli più sofisticati come quelli detti di "transistore bloccato acceso" (*transistor stuck-on*) o "transistore bloccato spento" (*transistor stuck-open*) [7], oppure come cortocircuito fra nodi (*bridging*) [8,9,10].

In alcuni casi, questi guasti possono dare luogo a cambiamenti nel valore logico delle uscite primarie del circuito risultando, quindi, rivelabili nello stesso

modo in cui lo sono quelli del tipo *stuck-at*.

Altre volte, invece, essi non sono in grado di cambiare le uscite osservabili, ma provocano significative deviazioni del comportamento del circuito rispetto al caso corretto per quanto riguarda altri importanti aspetti (assorbimento di corrente, ritardi di propagazione, ...). In questo caso, i guasti presentano particolari esigenze dal punto di vista della FS non garantite dalla simulazione a livello *gate*.

In particolare, guasti di tipo *transistor stuck-on* o *bridging* possono dare luogo sia ad un incremento della corrente statica assorbita dal circuito, sia a valori di tensione sulle linee di segnale intermedi tra i livelli standard. Queste tensioni intermedie vengono poi rapidamente normalizzate nella propagazione attraverso alcuni *gate* e possono convergere al valore logico corretto o a quello opposto, a seconda che il loro valore iniziale fosse dalla stessa parte o da quella opposta della soglia logica rispetto al caso privo di guasti.

Anche nel primo caso, tuttavia, essi debbono essere rivelati perchè alterano il funzionamento dinamico del circuito.

Dal punto di vista della FS, quindi, è evidente che occorre essere in grado di confrontare un livello di tensione di tipo essenzialmente analogico con la soglia logica dei *gate*. A questo fine, per altro, è necessario confrontare la conduttanza delle reti che connettono il nodo a cui si manifesta il guasto con l'alimentazione e la massa, il chè implica simulatori in grado di effettuare almeno stime elementari di tipo elettrico.

8.4 Algoritmi per la simulazione di guasto

L'approccio più semplice alla FS consiste nell'eseguire una simulazione (tipicamente a livello logico[4]) di diversi circuiti, ciascuno ottenuto da quello corretto mediante l'inserimento di uno dei possibili guasti dell'insieme considerato. Chiaramente, in circuiti di grandi dimensioni questo approccio dà luogo ad eccessivi tempi di simulazione[5]. Per questo motivo, si rende necessario l'uso di tecniche in grado di migliorare l'efficienza complessiva della simulazione, tipicamente simulando più di un guasto (o più di un vettore di *test*) ad ogni singolo passo.

Fra queste tecniche alcune, dette di tipo "parallelo", cercano di sfruttare le caratteristiche di parallelismo dei calcolatori. Nel particolare caso della simu-

[4] Come notato nel paragrafo precedente, il livello di descrizione circuitale a cui operano i programmi di simulazione di guasto è quello logico. In questo ambito, tra i possibili diversi sottolivelli (*switch*, *gate* e *behavioral*), si farà implicitamente riferimento a quello di *gate* con guasti del tipo *stuck-at 0/1*. Ciò sia perchè questo approccio è il più diffuso, sia perchè esso consente una maggiore semplicità e chiarezza di trattazione. I concetti che verranno esposti sono comunque generali e possono essere estesi a casi più generali.

[5] Si pensi che il numero di guasti da simulare è tipicamente dell'ordine di grandezza del numero G di *gate* presenti nel circuito e che il costo di ciascuna simulazione è proporzionale, oltre che alla lunghezza della sequenza di *test*, a G. Conseguentemente il costo della FS è tipicamente proporzionale a G^2.

lazione a livello logico, ciò può essere ottenuto anche in architetture che utilizzano normali processori scalari, sfruttando il parallelismo fra i *bit* che è intrinseco nelle operazioni logiche della ALU (*Arithmetic Logic Unit*).

Altre tecniche, invece, cercano di sfruttare il fatto che ciascun circuito guasto è solo in minima parte diverso da quello corretto per simulare solo le differenze dovute ai guasti. In questo caso gli algoritmi utilizzati sono detti di tipo "deduttivo" o "concorrente".

Tutti gli algoritmi menzionati in precedenza richiedono l'inserimento del singolo guasto (ovvero di bloccare la linea guasta all'opportuno valore logico) e la propagazione degli effetti di quest'ultimo per determinare se esso dà luogo ad un errore logico su almeno una delle uscite osservabili.

Esistono, però anche tecniche diverse che, utilizzando solamente una simulazione del circuito corretto, cercano di individuare i guasti rivelabili senza simularne direttamente gli effetti. Tra queste tecniche, la più importante è quella denominata propagazione all'indietro (*backtracing*) dei "cammini critici", anche se sono stati sviluppati algoritmi basati sulla compattazione di grafi che a parte gli aspetti teorici (abbastanza diversi da quello dei cammini critici) compiono una serie di operazioni molto simili a quelle svolte dal metodo dei cammini critici.

Gli algoritmi di tipo parallelo, deduttivo, concorrente e il metodo dei cammini critici costituiscono gli approcci più diffusi e di più lunga tradizione alla FS.

Tuttavia, ne esistono anche altri, tra cui alcuni sono modifiche più o meno complesse dei precedenti (tipicamente mirati a migliorarne le prestazioni), mentre altri adottano approcci originali. Fra questi ultimi meritano un accenno quelli basati su metodologie di tipo statistico, come ad esempio le tecniche basate sul campionamento di guasti o sulle misure di collaudabilità.

Degni di nota sono, infine, alcuni approcci algoritmici basati sulla propagazione di guasti singoli, come ad esempio la simulazione di guasti di tipo differenziale, che, in presenza di circuiti di dimensioni veramente grandi, trae vantaggio da una minore occupazione di memoria nell'elaboratore rispetto ad altre tecniche almeno in linea teorica più efficienti o gli algoritmi in cui vengono processati più vettori di collaudo in un singolo passo sfruttando il parallelismo di particolari architetture.

In questo contesto, il resto del presente capitolo è dedicato ad una descrizione più dettagliata degli approcci più diffusi tra quelli accennati in precedenza.

8.4.1 Algoritmo parallelo

In questo algoritmo [11], a ciascuna linea del circuito viene assegnata una parola-macchina (di lunghezza n) in cui il primo *bit* rappresenta il valore logico di tale linea (con il vettore di collaudo corrente) nel circuito privo di guasti, mentre ciascuno degli ulteriori $n-1$ *bit* indica il valore della stessa linea nel caso sia presente

uno dei guasti considerati[6]. In questo modo, per ciascuna coppia omologa di *bit* di due parole (che rappresentano i valori di due linee di ingresso di un *gate*), le operazioni logiche (AND, OR, ...) della ALU dell'elaboratore sono in grado di simulare le analoghe operazioni svolte dalle porte logiche del circuito.

L'aspetto essenziale dell'algoritmo è che un numero di guasti pari a $n-1$ viene simulato in parallelo.

Chiaramente i guasti rivelati sono quelli che, in almeno una delle parole associate alle uscite primarie del circuito, presentano un *bit* diverso da quello del caso privo di guasti (ovvero il primo *bit*).

I limiti di questo algoritmo sono dati principalmente dalla sua scarsa versatilità che lo rende in pratica applicabile solamente in simulatori logici a livello *gate*, in quanto le operazioni logiche della ALU non sono in grado di simulare quelle di blocchi funzionali più complessi. Inoltre, esso non consente di tenere adeguatamente in conto le temporizzazioni del circuito. Infatti, in questo caso non è detto che tutti guasti diano luogo a transizioni dell'uscita di un *gate* fra loro contemporanee e quindi non possono essere simulati in un solo passo.

La Fig. 8.1 mostra un semplice esempio di simulazione di guasti parallela nella quale si ipotizza di utilizzare un calcolatore operante con parole di 8 *bit* (cioè con $n=8$).

8.4.2 Algoritmi deduttivo e concorrente

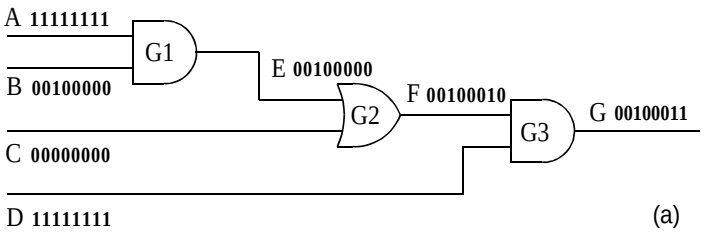
In questo paragrafo vengono descritti sia il metodo deduttivo [12] che quello concorrente [13], che presentano alcuni punti in comune.

Entrambi questi algoritmi cercano di sfruttare il fatto che ciascun circuito guasto differisce "di poco" da quello corretto, calcolando l'uscita dei soli *gate* in cui questa può risentire degli effetti dei guasti. Nel seguito gli algoritmi verranno descritti sia da un punto di vista "statico", ovvero delle strutture di dati che rappresentano lo stato corrente del circuito corretto e di quelli guasti, sia da quello "dinamico", ovvero di come queste strutture vengano aggiornate quando viene simulata una certa sequenza di collaudo.

Nel metodo deduttivo, ad ogni linea di segnale del circuito viene associata una lista che contiene tutti i guasti che ne cambiano il valore rispetto al caso corretto. In questo modo, il valore logico dell'uscita di un *gate* viene calcolato solamente nel caso corretto, mentre la lista di uscita viene calcolata mediante semplici operazioni (unione, intersezione e complementazione) sulle liste di guasti in ingresso[7]. Tali operazioni dipendono evidentemente sia dal tipo di *gate* che dal suo stato logico (valori degli ingressi e delle uscite); naturalmente, la lista

[6] Naturalmente, se il numero dei guasti eccede la lunghezza della parola-macchina, sono necessarie più simulazioni per lo stesso vettore di *test*.

[7] Si consideri, ad esempio, il caso di un OR a due ingressi: nel caso siano entrambi al valore logico basso la lista di uscita è semplicemente data dall'unione delle liste di ingresso.



bit	guasto	
0	fault-free	
1	A s-at-1	
2	B s-at-1	•
3	C s-at-0	
4	D s-at-1	
5	E s-at-0	
6	F s-at-1	•
7	G s-at-1	•

(b)

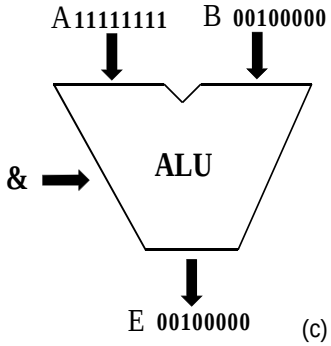


Figura 8.1: **Esempio di applicazione del metodo di FS parallelo ad un semplice circuito combinatorio (a). A ciascuna linea del circuito è associata una parola (in questo caso di 8 *bit*) in cui, per il vettore di *test* corrente, sono contenuti i valori di tale linea nel circuito privo di guasti (primo *bit*) e in presenza dei guasti (elencati nella tabella (b) in cui • indica che il guasto viene rivelato). In (c) è illustrato come la ALU possa calcolare la parola di uscita del *gate* G1.**

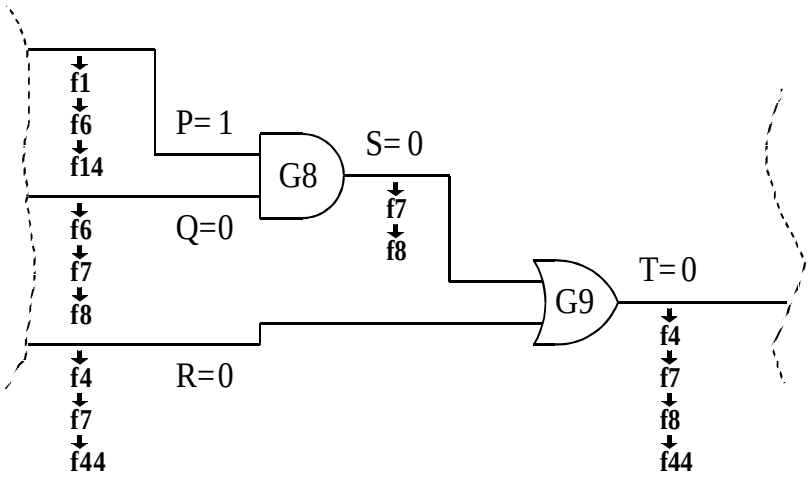


Figura 8.2: **Esempio di applicazione dell'algoritmo deduttivo. A ciascuna linea del circuito è associata una lista di guasti per i quali essa assume un valore diverso da quello corretto. Come esempio del modo in cui la lista di uscita di un *gate* venga ottenuta da quelle di ingresso, si consideri il *gate* G8. Come può essere verificato, i soli guasti che compaiono nella lista di S (in quanto ne cambiano il valore) sono f_7 ed f_8 (tali da rendere Q=1). È poi interessante notare che f_6, che pure cambia il valore di Q, non è propagato poichè esso cambia anche il valore di P e di conseguenza S rimane inalterato.**

associata all'uscita del *gate* contiene tutti e soli i guasti che danno luogo ad un valore dell'uscita diverso da quello che si avrebbe nel circuito corretto.

In pratica, il valore dell'uscita di un *gate* per i guasti che si trovano nelle liste associate alle sue linee di ingresso non viene esplicitamente calcolato, ma "dedotto" a partire dallo stato del *gate* in assenza di guasti.

Con questo metodo, i guasti rivelati da ciascun vettore di *test* sono quelli presenti nelle liste associate alle uscite osservabili del circuito.

In Fig. 8.2 sono illustrate le liste di guasti ottenute mediante l'algoritmo deduttivo nel caso di un semplice sottocircuito (parte di uno più grande) con i valori logici riportati in figura.

Il metodo concorrente è basato su princìpi in parte analoghi a quelli dell'algoritmo deduttivo, ma associa una lista a ciascun *gate* anzichè alle linee del circuito. In tale lista, ciascun elemento contiene lo stato (ovvero i valori degli ingressi e dell'uscita) del gate g considerato nel caso privo di guasti (questo valore è denotato come $s(g,0)$ in quanto esso rappresenta il primo elemento della lista) e in ciascuno dei circuiti guasti per cui tale stato differisce da quello privo di guasti ($s(g,f) \neq s(g,0)$, ove f rappresenta un generico guasto).

Come verrà in seguito illustrato, la lista associata ad un *gate* non viene ot-

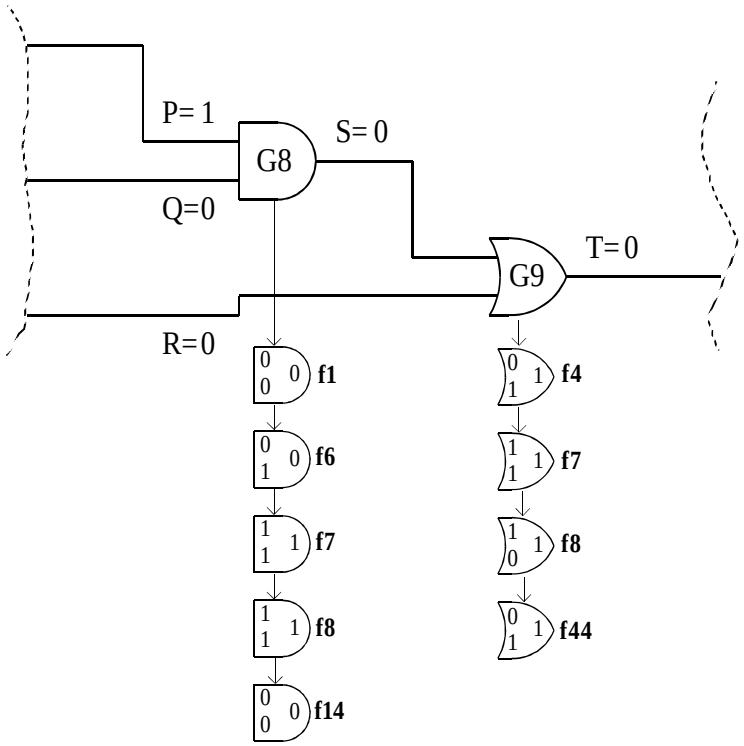


Figura 8.3: **Esempio di applicazione dell'algoritmo concorrente. A ciascun** *gate* **del circuito è associata una lista che contiene, per ciascun guasto, lo stato di tale** *gate* **nel caso in cui questo differisca dal caso corretto. È interessante osservare come non sia necessario che gli elementi di tali liste abbiano il valore di uscita diverso da quello del caso privo di guasti (al riguardo si comparino le liste con quelle di Fig. 8.2).**

tenuta mediante operazioni sulle liste dei *gate* compresi nel suo *fan-in*, ma in maniera del tutto analoga a come viene calcolata l'uscita del gate privo di guasti.

In Fig. 8.3 sono illustrate le liste di guasti ottenute mediante l'applicazione del metodo concorrente allo stesso sottocircuito di Fig. 8.2 e con lo stesso vettore di collaudo.

Da quanto precede, e dagli esempi riportati, è facile rendersi conto che dal punto di vista "statico" (ovvero per il calcolo dello stato stabile del circuito corretto sia di quello dei circuiti "guasti"), il metodo deduttivo e quello concorrente sono simili. Le maggiori differenze si hanno, invece, nel modo in cui queste liste vengono aggiornate durante la simulazione di un'intera sequenza di vettori di collaudo.

Per illustrare questa differenza, conviene qui introdurre brevemente alcuni fon-

damentali concetti utilizzati nella simulazione logica in assenza di guasti, dove, al fine di ridurre i tempi di calcolo, si cerca di sfruttare il fatto che ad ogni cambiamento di stato di un circuito digitale solo una piccola percentuale (10-15%) di *gate* cambia di stato. Pertanto, tutte le volte in cui è necessario aggiornare lo stato del circuito[8], solamente una frazione di tutti i possibili *gate* (quelli che possono cambiare di stato, perciò detti "attivi") deve essere valutata. Questo è ottenibile mediante algoritmi di simulazione del tipo *event-driven* o *activity-oriented*, in cui per ogni cambiamento dell'uscita di un *gate* (evento) viene programmata la valutazione di tutti i *gate* appartenenti al suo *fan-out*. In questo modo un *gate* viene ricalcolato solo in presenza del cambiamento di almeno uno dei suoi ingressi.

Il metodo concorrente, diversamente da quello deduttivo, cerca di trarre vantaggio da questa tecnica in particolare, mentre in entrambi i metodi si cerca di calcolare l'uscita di un *gate* appartenente a un circuito "guasto" solamente nel caso in cui lo stato di questo differisca da quello del circuito privo di guasti, nel caso concorrente lo stato di ciascun *gate* (appartente sia al circuito corretto che a uno di quelli guasti) viene valutato solamente se lo stato dei suoi ingressi è cambiato rispetto al vettore di *test* precedente.

Per meglio comprendere la differenza fra il modo di aggiornare le liste di guasti nei due algoritmi, consideriamo in maggiore dettaglio le operazioni svolte.

Nel metodo deduttivo, con l'applicazione di un nuovo vettore di *test*, la lista di uscita di un *gate* viene ricalcolata quando: *gate*:

1. esso cambia di stato (ingressi e/o uscita);

2. cambia la lista di guasti ad almeno un suo ingresso (perchè sono stati aggiunti o tolti degli elementi).

La condizione 1 è evidentemente necessaria in ogni caso, mentre la condizione 2 porta ad una serie di operazioni non necessarie. Supponiamo, infatti, che sia cambiata una delle liste di guasti in ingresso al *gate* considerato (il cui stato logico, in assenza di guasti, rimane lo stesso che si aveva con il vettore di *test* precedente), ad esempio perchè è stato aggiunto un elemento: in questo caso, evidentemente, sarebbe sufficiente rivalutare l'uscita del *gate* solo per il guasto corrispondente a quell'elemento.

Questo tipo di problema è dovuto alla propagazione degli eventi di lista (*list event*), così chiamati in quanto il cambiamento di una lista di guasti associata ad una linea fa sì che vengano automaticamente ricalcolate le liste di uscita dei *gate* che hanno tale linea come ingresso. Questo e viene evitato nel metodo concorrente. Infatti, sia $L_g(t)$ la lista associata a un *gate* g con il vettore di *test*

[8] Ovvero, per ogni vettore di ingresso applicato al circuito nel caso si consideri un circuito combinatorio (o un circuito sincrono) con i *gate* a ritardo nullo e tipicamente organizzati a livelli (con eventuali elementi di memoria a ritardo unitario). Oppure, per ogni istante della base dei tempi (tipicamente discreta) utilizzata, nel caso si utilizzino *gate* a ritardi di propagazione finiti. È evidente che in quest'ultimo caso si ottengono i maggiori benefici.

t e supponiamo che con il vettore di *test* $t+1$ tale lista sia stata aggiornata a $L_g(t+1)$ [9]. Finita questa operazione, vengono programmati degli eventi sui *gate* di *fan-out* di quello considerato a causa dei seguenti eventi:

1. è cambiato il valore di uscita del *gate* considerato (nel caso corretto);
2. è cambiato il valore di uscita di uno degli elementi di $L_g(t)$ (si noti che anche se questo elemento non è in $L_g(t+1)$ è necessario comunque propagare un evento);
3. sono stati aggiunti degli elementi a $L_g(t+1)$ (rispetto a $L_g(t)$).

Differentemente dal caso deduttivo, gli eventi 2 e 3 (corrispondenti al caso 2 nell'algoritmo deduttivo), implicano la valutazione dei soli elementi nelle liste associate ai *gate* presenti nel *fan-out* di quello considerato. In altri termini, se con il *test vector* $t+1$ un *gate* (sia quello privo di guasti che quelli corrispondenti ai diversi guasti) ha il medesimo stato $s(g,f)$ che aveva con t, allora non verrà valutato.

Come esempio di questa differenza fra i due algoritmi si considerino le Fig. 8.4 e 8.5 in cui è mostrato l'aggiornamento delle liste di guasti quando dopo un primo vettore di *test* (t_1) ne viene applicato un altro (t_2). Al riguardo si consideri il *gate* G_2, il cui stato logico cambia soltanto nel circuito con il guasto f_1. Come si può vedere nelle due figure (in cui le frecce mostrano le implicazioni dei diversi eventi), il metodo deduttivo a causa di questo evento ricalcola completamente la lista di uscita di G_2, mentre quello concorrente si limita ad aggiornare lo stato del solo elemento che descrive lo stato di G_2 con il guasto f_1.

Dal punto di vista dell'utilizzo, il confronto fra i due algoritmi presenta le sequenti caratteristiche:

- occupazione di memoria: è maggiore nel caso concorrente, in quanto devono essere memorizzate più informazioni;
- velocità: il metodo concorrente è, in generale, più veloce, in quanto calcola l'uscita di un *gate* solo se lo stato di tale *gate* è diverso dal circuito privo di guasti, e se tale *gate* è "attivo";
- versatilità: costituisce il maggiore vantaggio del metodo concorrente, in quanto:
  - il metodo deduttivo è limitato al livello di simulazione di *gate*, perchè le operazioni sulle liste possono essere definite in maniera semplice solo

[9] Questo viene fatto aggiornando lo stato dei *gate* guasti nella lista, ed eventualmente eliminando dei guasti oppure aggiungendone di nuovi, al fine di soddisfare le proprietà che tali liste devono avere.

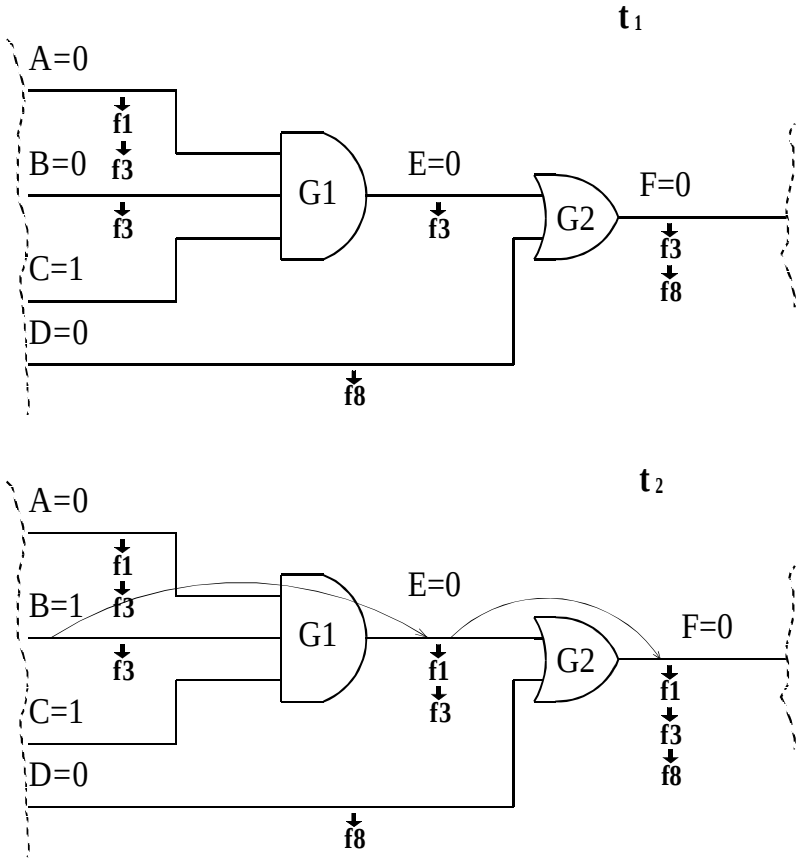


Figura 8.4: **Esempio di aggiornamento delle liste dei guasti nell'algoritmo deduttivo in un semplice circuito (inteso come parte di uno più grande). Nella figura sono illustrate le condizioni (stato logico e liste di guasti) che si hanno con i vettori di** *test* t_1 **e** t_2**, rispettivamente. Come si può osservare, il cambiamento del segnale di ingresso** B $(0 \rightarrow 1)$ **implica di ricalcolare la lista associata alla linea** E**, cui viene aggiunto il guasto** f_1**. Questo da luogo ad un** *list event* **che implica, a sua volta, di dover ricalcolare l'intera lista associata a** G_2**.**

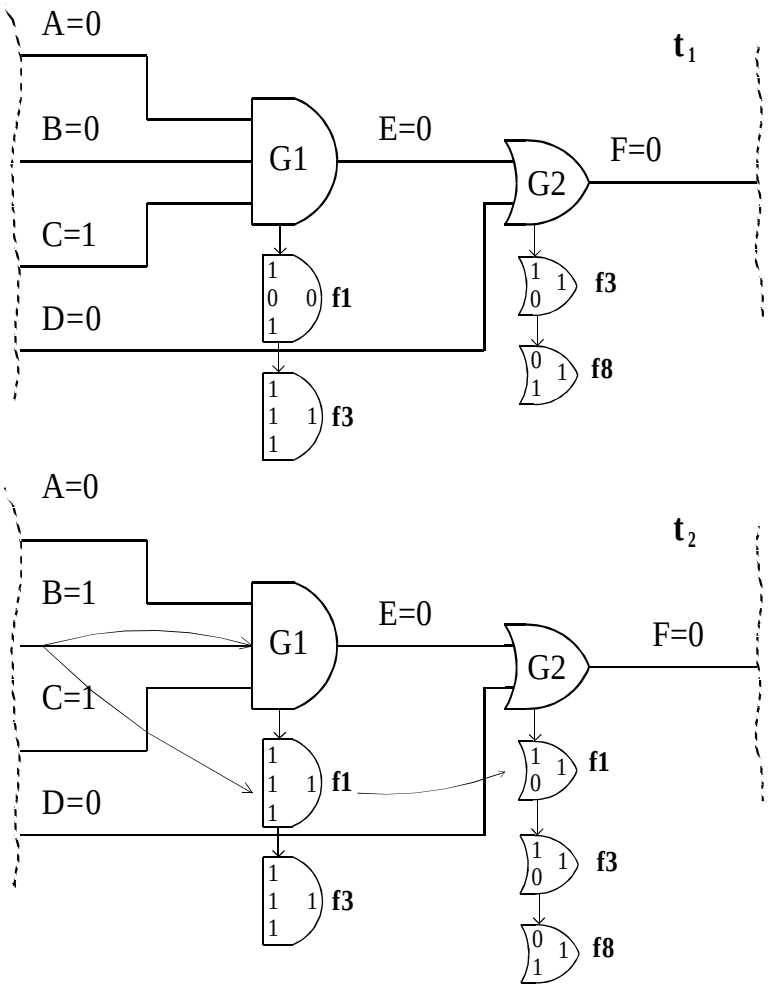


Figura 8.5: **Esempio di aggiornamento delle liste nell'algoritmo concorrente nello stesso caso di Fig. 8.4. In particolare, il cambiamento del segnale di ingresso B ($0 \rightarrow 1$) implica di ricalcolare la lista associata al *gate* G_1, in cui vengono cambiati sia lo stato del circuito privo di guasti che quello del circuito guasto f_1. Questo da luogo ad un solo evento dato dal cambiamento dell'uscita di G_1 per il guasto f_1, che implica semplicemente di aggiornare lo stato di f_1 per il *gate* G_2 (in questo caso è necessario aggiungere un elemento alla lista), senza dover ricalcolare l'intera lista associata a G_2.**

per porte le logiche elementari. Nell'approccio concorrente, invece, lo stato di un generico blocco in presenza di guasti può essere calcolato con le stesse procedure utilizzate per la simulazione in assenza di guasti che quindi possono operare anche a livello *switch* o *behavioral*;

- ritardi di propagazione: il metodo deduttivo presenta seri problemi quando si debbano coniderare ritardi di propagazione finiti dei singoli *gate*, perchè i cambiamenti di stato dello stesso *gate* nei diversi circuiti guasti avvengono ad istanti diversi, in ciascuno dei quali diventa necessario rivalutare la lista di uscita (per gran parte uguale alla precedente). Il metodo concorrente, invece, è in grado di programmare gli eventi dei diversi circuiti guasti a tempi diversi (poichè segue l'approccio di tipo *event-driven*).

Chiaramente, entrambi i metodi traggono vantaggio dall'eliminare da successive simulazioni i guasti già rivelati (*fault-dropping*).

8.4.3 Metodo dei cammini critici

Il metodo di propagazione all'indietro dei cammini critici [14], differentemente da quelli illustrati in precedenza, tratta i guasti solo implicitamente, senza simulare l'attività del circuito in loro presenza. In pratica, in un circuito combinatorio[10], per ciascun vettore di *test* viene eseguita la simulazione in assenza di guasti, determinando lo stato di ciascuna linea; in seguito, a partire dalle uscite primarie (cioè osservabili) del circuito e procedendo verso gli ingressi, vengono identificati i nodi critici, ovvero quelli una cui variazione di stato cambia il valore di almeno un'uscita primaria. Questa definizione implica che a ciascun nodo critico corrisponda un guasto del tipo *stuck-at* rivelabile (quello al un valore opposto del caso corretto).

L'identificazione dei nodi critici può essere descritta nel modo seguente (Fig. 8.6): a) le uscite primarie del circuito sono per definizione critiche; b) una linea di ingresso di un gate è critica se lo è l'uscita e se un suo cambiamento di valore si riflette sull'uscita del gate stesso.

L'analisi della criticità delle linee, nel caso di reti ad albero (prive di riconvergenze di fan-out, ovvero tali che non esista alcun gate raggiungibile dall'uscita di un altro percorrendo due cammini diversi), risulta particolarmente semplice. Nel caso più generale, tuttavia, il metodo dei cammini critici (nella sua formulazione originale); non è esatto, in quanto al fan-out riconvergente sono associati fenomeni, detti di "mascheramento" o di "evidenziamento", che rendono non immediata la determinazione della criticità di un nodo con *fan-out* maggiore di uno. Infatti, un nodo con queste caratteristiche può essere non critico anche se uno dei

[10] Questo metodo può essere comunque esteso a circuiti sequenziali sincroni [15].

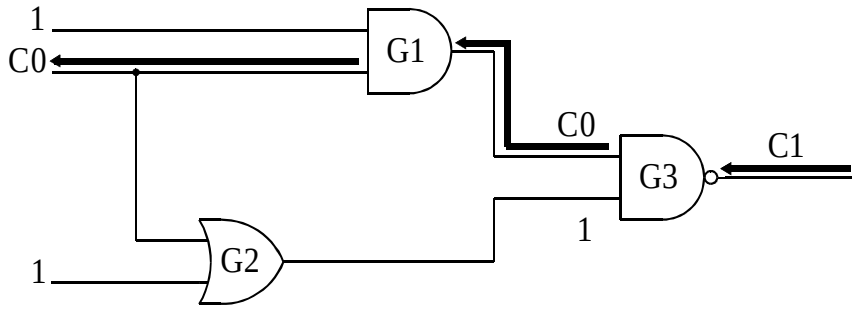


Figura 8.6: **Esempio di applicazione del metodo dei cammini critici. Le frecce in grassetto indicano il cammino critico (unico in questo semplice esempio) nel circuito, tracciato dall'uscita verso gli ingressi. I valori logici delle linee critiche sono preceduti dal simbolo C. Da un analisi di questo esempio è facile verificare che, in particolare, una linea critica al valore logico V corrisponde ad un guasto del tipo *stuck-at-*$\overline{V}$ rivelabile.**

suoi rami di *fan-out* risulti critico ("mascheramento"), oppure può essere critica nonostante nessuno di tali rami lo sia ("evidenziamento").

Per illustrare questo fenomeno con un esempio particolarmente semplice, consideriamo la rete a due livelli che realizza la funzione $O = AB + \overline{B}C$ (Fig. 8.7-a): se $A = 1$, $B = 1$ e $C = 1$, allora O vale 1 indipendentemente da B, e il guasto *stuck-at-1* di B, non è rivelabile nonostante sia possibile tracciare un cammino critico da O fino a B. Viceversa, nella rete che implementa $O = AB + BC$ (Fig. 8.7-b), per $A = 1$, $B = 1$ e $C = 1$, B *stuck-at-0* può essere rivelato nonostante non sia possibile tracciare dei cammini critici da O a B.

I problemi di mascheramento ed evidenziamento, solo parzialmente risolti nella versione originale del metodo dei cammini critici, sono stati in seguito risolti in maniera esatta al prezzo di un incremento nella complessità dell'algoritmo [15].

Poichè richiede solamente la simulazione del circuito in assenza di guasti e l'individuazione dei nodi critici (essenzialmente effettuata attraverso la propagazione a ritroso dei cosiddetti cammini critici), il metodo è potenzialmente più efficiente di quelli descritti in precedenza. Tuttavia, questo vantaggio viene in parte meno nel caso l'algoritmo se si vuole rendere esatto l'algoritmo.

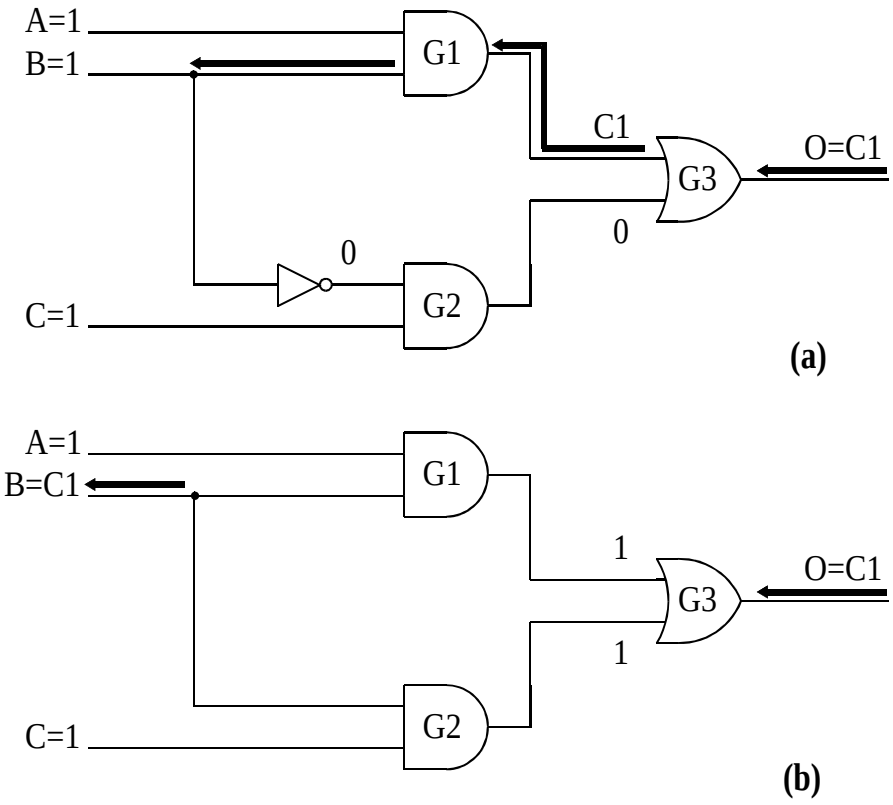


Figura 8.7: **Esempi di "mascheramento" e di "evidenziamento" nel metodo dei cammini critici. Le frecce in grassetto indicano il cammini critici tracciati dall'uscita verso gli ingressi e i valori logici delle linee critiche sono preceduti dal simbolo C.**

8.4.4 Tecniche alternative di simulazione di guasti

Tecniche di tipo statistico

Fra le tecniche di FS di tipo statistico, la simulazione[11] di una sola porzione ("campione") dei possibili guasti (*fault sampling*) [16] consente di trattare circuiti di dimensioni e con un numero di guasti tali da rendere troppo costosa una simulazione completa [17] Naturalmente, la copertura ottenuta su tale campione costituisce solamente una stima di quella effettiva, anche se la confidenza di tale stima può essere calcolata a partire dalla frazione di guasti simulati [Agrawal 81].

Un ulteriore metodo si basa sul fatto, che per particolari tipi di sequenze di vettori di *test* (in particolare quelle generate in maniera pseudo-casuale[12]), è possibile determinare una stima della copertura di guasti in funzione del numero di vettori di *test* [18] senza eseguire una vera e propria FS. Questo è possibile se è nota la probabilità (*detection probability*) di rivelare ciascun guasto mediante l'applicazione di uno stimolo di tipo casuale. Quest'ultima quantità, a sua volta, può essere determinata mediante tecniche (generalmente denominate "misure di collaudabilità") che analizzano il circuito in maniera probabilistica, senza bisogno di ricorrere alla simulazione di guasti.

Il principale svantaggio degli approcci di tipo statistico rimane la scarsa accuratezza che li rende inadatti in applicazioni ove sia richiesta un'elevata affidabilità del processo di collaudo.

Simulazione di guasti di tipo differenziale

Al fine di simulare circuiti sequenziali sincroni di grandi dimensioni è stata sviluppata una tecnica di FS di tipo "differenziale" [19], che mira a ridurre l'occupazione di memoria (particolarmente critica in questo tipo di circuiti) che sarebbe richiesta dal processo di simulazione qualora vengano utilizzate tecniche di tipo deduttivo o concorrente. Anche questa tecnica cerca di sfruttare il fatto che, in generale, due circuiti guasti differiscono fra loro di poco.

La FS di tipo "differenziale" può essere spiegata mediante un esempio: si supponga di voler simulare un circuito sequenziale sincrono per una sequenza di vettori di ingresso $t_0, t_1 \ldots t_n$. Con il primo vettore di ingresso (t_0), viene calcolato lo stato del circuito in assenza di guasti, successivamente viene simulato un guasto per volta.

[11] Tipicamente mediante le tecniche descritte nei paragrafi precedenti.

[12] Queste sequenze, in cui ciscuna linea di ingresso del circuito ha la stessa probabilità di essere al valore logico alto o basso, sono tipicamente utilizzate nei circuiti in grado di autocollaudarsi (*Built-In Self-Test*) in quanto possono essere realizzate semplicemente retroazionando linearmente dei registri a scorrimento (*Linear Feedback Shift Register*, LFSR).

In particolare, dopo aver valutato lo stato del circuito in presenza di un guasto (f_k) (mediante il consueto meccanismo di iniezione e propagazione) Al passo successivo, invece di ripristinare lo stato corretto di tutto il circuito (che richiederebbe, in linea di principio, di aggiornare lo stato di ciascun *gate*), viene iniettato il guasto opposto[13] a f_k (in modo da ripristinare lo stato del circuito cambiando unicamente lo stato delle linee che risentono di f_k) e contemporaneamente viene iniettato il nuovo guasto f_{k+1}. In questo modo, propagando gli effetti dei due guasti iniettati si calcola lo stato del circuito in presenza di f_{k+1}.

Chiaramente, per ciascun guasto si controlla se questo è stato rivelato alle uscite del circuito da vettore di collaudo corrente e si memorizzano i valori assunti dagli elementi di memoria (*Flip-Flop*) che vengono poi riutilizzati ("iniettandoli") nella simulazione dello stesso guasto con i vettori successivi.

Questa tecnica può risultare molto veloce, anche se la sua efficienza dipende dall'ordine in cui vengono simulati i guasti (è evidente che per due guasti consecutivi conviene che l'attività indotta nel circuito sia simile, in modo da valutare il minor numero possibile di *gates*). La sua maggiore limitazione consiste nell'impossibilità di trattare modelli di ritardo più sofisticati di quelli a ritardo nullo e a ritardo unitario.

Simulazione parallela di vettori di collaudo

Tutti gli algoritmi fino ad ora illustrati sono utilizzabili in calcolatori i cui microprocessori hanno un architettura di tipo scalare; seppure meno diffusi esistono anche elaboratori di tipo parallelo, in cui la stessa istruzione viene applicata "in parallelo" a un numero anche molto grande (alcune migliaia) di dati.

Al fine di sfruttare al meglio questo tipo di architetture è stato sviluppato un algoritmo detto $PPSFP$ (*Parallel Pattern Single Fault Propagation*) in cui viene simulato un guasto per volta, ma per tutti i vettori di collaudo[14].

In pratica, il modo in cui viene calcolata la risposta di un circuito è simile a quello utilizzato nella simulazione logica con la differenza che: a) a ogni segnale viene associato un vettore di valori, ciascuno dei quali ne rappresenta lo stato con un dato vettore di ingresso; b) con una sola operazione viene calcolata la risposta di un *gate* per tutti i vettori di ingressso.

Questo metodo è molto veloce, ma è comunque limitato al caso di circuiti combinatori.

[13] Se f_k è uno *stuck-at-1/0* viene iniettato un guasto *stuck-at-0/1* nella stessa locazione di f_k.

[14] Questo è possibile in un singolo passo solo se il numero di vettori da simulare è minore del parallelismo (ovvero il numero di dati elaborabili in un singolo passo) della macchina considerata.

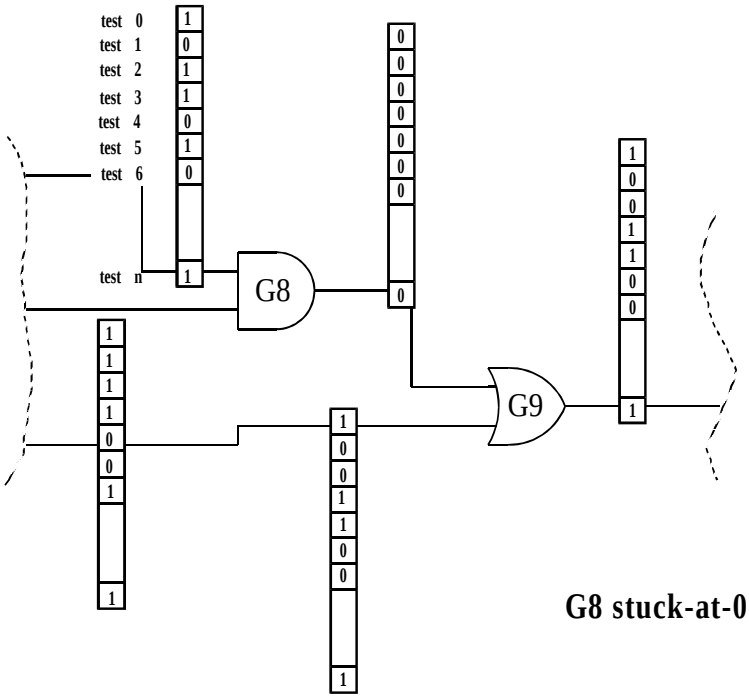


Figura 8.8: **Esempio di applicazione del metodo di FS di tipo PPSFP. Il guasto correntemente simulato è uno** *stuck-at-0* **sull'uscita di** *G8*. **A ciascun segnale viene associato un vettore di dati con un** *entry* **corrispondente a ciascuno degli** n **vettori di ingresso simulati in parallelo (e contenente il valore di tale linea con tale vettore di ingresso). L'uscita di un** *gate* **viene calcolata con unica operazione di tipo parallelo per tutti i vettori di ingresso.**

8.5 Simulazione di guasti non del tipo stuck-at

Come si è visto nel Cap. XXXX il modello di guasto del tipo *stuck-at* rappresenta solo una parte dei possibili malfunzionamenti presenti negli odierni circuiti integrati e in particolare in quelli CMOS. Ciò ha richiesto l'introduzione di nuovi modelli di guasto fra i quali i più importanti sono quelli di tipo cortocircuito (*bridging*), ritardo (*delay*) e i guasti del tipo transistore bloccato (*transistor stuck-open* e *stuck-on*).

Si è quindi posto il problema di sviluppare simulatori che siano in grado di trattare anche questo tipo di guasti. Solo in questo modo, infatti è possibile ottenere stime della copertura sui guasti che siano in grado di fornire una previsione realistica dell'effettiva resa del processo di collaudo.

Nonostante i concetti fondamentali siano i medesimi della simulazione di guasti del tipo *stuck-at* al livello *gate*, la simulazione con questi nuovi modelli di guasto ha posto una serie di nuove problematiche, richiedendo, in taluni casi, radicali cambiamenti negli approcci tradizionalmente utilizzati quali ad esempio il passaggio dal livello *gate* a livelli più vicini a quello elettrico come, ad esempio, quello *switch*.

Questi cambiamenti, tipicamente dovuti alla ricerca di una maggiore accuratezza, hanno poi dato luogo a notevoli problemi di efficienza e quindi a una notevole attività di ricerca che è tuttora ben lontana dall'esaurirsi.

Nel seguito, senza ritornare sugli aspetti generali della FS già considerati nel caso degli *stuck-at*, verranno brevemente analizzate alcune specifiche problematiche della FS nei circuiti CMOS (transistor stuck-open/on e cortocircuiti) e della simulazione di guasti del tipo ritardo. Queste costituiscono, infatti, gli esempi più interessanti di simulazione di guasti non di tipo *stuck-at*.

8.5.1 Simulazione di guasto per circuiti CMOS

Un guasto può alterare la natura complementare del funzionamento dei circuiti CMOS [**?**] in due modi diversi (Cap. XXXX): a) dando luogo a un comportamento "analogico" caratterizzato da assorbimento di corrente in condizioni statiche e da valori di tensione intermedi sui nodi di segnale (guasti del tipo cortocircuito o transistore bloccato acceso); b) dando luogo a un comportamento "sequenziale" di blocchi (*gate*) altrimenti combinatori (transistore bloccato spento).

La necessità di considerare anche questi modelli di guasto ha posto nuove esigenze di accuratezza per quello riguarda la FS che, nel caso dei circuiti CMOS deve essere in grado di trattare comportamenti non rappresentabili mediante la simulazione logica convenzionale. D'altra parte, le dimensioni dei circuiti e il numero di guasti da considerare non consentono l'utilizzo di simulatori al livello elettrico. È quindi evidente come in parallelo all'attività di modellistica di guasto per i circuiti CMOS (illustrata nel Cap. XXXX) sia presente una notevole attività di ricerca mirata allo sviluppo di strumenti di FS (e programmi di

test-generation).

Da quanto detto appare evidente che, in questo caso, il maggiore problema consiste nella ricerca di un compromesso ottimale fra l'accuratezza richiesta dai nuovi modelli di guasto e l'efficienza necessaria per trattare circuiti VLSI.

Nonostante, il modo in cui vengono valutati gli effetti dei guasti sia notevolmente diverso da quelli utilizzati nel caso degli stuck-at, i principi base della simulazione di guasto rimangono gli stessi, per cui, nel resto di questo paragrafo l'attenzione sarà principalmente rivolta ad evidenziare le particolari esigenze della FS per circuiti CMOS piuttosto che ripetere la struttura dei diversi algoritmi.

Nel seguito verranno illustrati in dettaglio i casi dei guasti di tipo cortocircuito e transistore bloccato spento.

Simulazione di guasti di tipo cortocircuito

Il principale problema di accuratezza nella simulazione di questo tipo di guasti consiste nel valutare il valore assunto dal segnale (o dai segnali) affetti dal guasto e il modo in cui tali valori vengono propagati nel circuito.

I simulatori sviluppati inizialmente per questo tipo di guasti (non nell'ambito della tecnologia CMOS) operavano al livello di *gate*, considerando quindi soltanto cortocircuiti fra le linee di uscita dei *gate*, utilizzando i modelli di guasto del tipo *wired-and* e *wired-or* (vedi Cap. XXXX) per descrivere il comportamento di due uscite di *gate* cortocircuitate [8,20,21].

Questo modello, molto conveniente in quanto di tipo logico (e quindi facilmente trattabile con tutte le tecniche di FS illustrate nei paragrafi precedenti), è valido nel caso della tecnologia TTL (*wired-or*) e in quello della tecnologia nMOS, ma non nel caso dei circuiti CMOS. Infatti, come si è visto nel capitolo XXXX il comportamento delle uscite di due *gate* cortocircuitati dipende dalle conduttanze in gioco, le quali sono tipicamente fra loro comparabili dando quindi luogo a valori di tensione intermedi dei segnali. Questo rende quindi inapplicabile

da invalidare qualsiasi assunzione apriori sul prevalere del valore alto o di quello basso.

La simulazione di guasti del tipo cortocircuito (e in modo del tutto analogo dei transistori bloccati accesi) non può quindi prescindere da una corretta valutazione delle conduttanze messe in conflitto a causa del guasto. Data l'inefficienza della simulazione al livello elettrico, sono quindi stati sviluppati una serie di simulatori che, nell'ambito nel continuo dei livelli di descrizione di un circuito digitale, ottengono diversi tipi di compromesso fra l'accuratezza necessaria per ottenere stime realistiche di copertura di guasti e l'efficienza richiesta dalle dimensioni dei circuiti da considerare.

Prima di investigare in dettaglio le tecniche che sono state sviluppate per risolvere, nell'ambito della FS, questi conflitti di conduttanza, conviene ricordare (vedi Cap. XXXX) che

Ritornando al caso specifico dei circuiti CMOS di tipo statico in presenza di guasti del tipo cortocircuito sono stati sviluppati sia simulatori al livello *gate* che al livello *switch*.

In particolare, al livello *gate*, l'attività di guasti del tipo cortocircuito può essere simulata memorizzando in un'opportuna tabella la conduttanza di uscita in funzione del vettore di ingresso per ciascun tipo di *gate* presente nel circuito [**?**]. In questo modo, per un dato valore della resistenza di corto si possono determinare i valori di tensione dei nodi cortocircuitati e quindi la possibilità o meno di propagare tali segnali come errori logici. Mediante l'utilizzo di tabelle, si possono trattare anche transistori bloccati accesi.

Questo approccio, è efficiente solo se il circuito è stato progettato utilizzando solo pochi tipi di *gate*. Inoltre, consente di trattare solo *gate* o *macro-gate* di tipo FCMOS e non consente di considerare guasti fra due nodi qualsiasi (non necessariamente uscite di *gate*).

La necessità di considerare un insieme più generale di guasti del tipo cortocircuito, unitamente alla presenza di circuiti CMOS (come ad esempio quelli che utilizzano *pass-transistor* [Mead-Conway]) non descrivibili a livello *gate*, ha volto l'attenzione verso la simulazione al livello *switch* [22,23,24,25] in cui l'elemento fondamentale utilizzato per descrivere il circuito non è il *gate*, ma il singolo transistore modellato al livello logico (vedi Sex. XXXX).

In particolare, il transistore può essere modellato mediante un interruttore ideale posto in serie ad un resistore la cui conduttanza può appartenere a un insieme di valori discreto (*strength*) o continuo.

Nel primo caso non è chiaramente possibile avere una valutazione anche approssimata delle tensioni ai nodi se non nel caso in cui le conduttanze in gioco differiscano di ordini di grandezza. Se questo, in generale, non costituisce un problema nel caso della simulazione di circuiti privi di guasti, in presenza di guasti come i transistori bloccati accesi o i cortocircuiti, è invece richiesta una più accurata valutazione delle conduttanze in gioco che sono tipicamente dello stesso ordine di grandezza.

VEDI ARTICOLO SUL COMPEURO

Chiaramente, la maggiore accuratezza di questo tipo di simulatori implica un aumento dei tempi di simulazione rispetto al livello *gate*, tale aumento può essere contenuto mediante approcci di tipo gerarchico in cui il livello *switch* viene utilizzato per simulare solo la parte del circuito direttamente interessata al guasto, mentre il resto del circuito può essere simulato a livello *gate*.

Nel caso di guasti del tipo transistore *stuck-open* i simulatori devono essere in grado di tenere in conto anche l'inizializzazione dei segnali. Anche in questo caso si hanno simulatori livello *gate* che tengono implicitamente in conto della

presenza di transistori [] e simulatori livello *switch*.

(stuck-at guasti interni)

Per quello che riguarda l'efficienza delle simulazioni (non solo nel caso di guasti del tipo cortocircuito) l'aggiunta di nuovi modelli di guasto porta a un incremento dei tempi di simulazione. A questo proposito, un approccio molto seguito consiste nel simulare preventivamente il circuito rispetto ai guasti del tipo *stuck-at*, questo rende disponibile un informazione sull'osservabilità di ciascuna linea che può essere utilizzata proficuamente per studiare anche altri tipi di guasto, senza il bisogno di ricalcolare per ciascuno di essi le uscite del circuito.

Vediamo due esempi di come questi concetti possono essere applicati a guasti non del tipo *stuck-at*, in entrambi i casi si consideri il circuito di Fig. XXXX con applicato il vettore di collaudo XXXX. Il circuito è stato simulato mediante il metodo dei cammini critici (ma le considerazioni che verranno fatte valgono per qualsiasi altro algoritmo) e sono riportate le linee critiche. Si consideri il caso di cortocircuiti fra due linee di uscita di *gate* (che non diano luogo a retroazioni) tali guasti possono essere rivelati se le due linee cortocircuitate: a) hanno valori logici diversi nel circuito privo di guasti; b) quella (se esiste) di esse che cambia di valore logico a causa del corto deve essere osservabile in uscita. Quindi, invece di simulare tutti i guasti per cui la condizione a) è soddisfatta, i cortocircuiti da esaminare sono quelli che oltre a soddisfare la condizione a) presentano almeno una delle due linee critica, a questo punto una volta risolto il conflitto elettrico (determinando così quale linea cambia di valore) la rivelabilità del guasto è nota.

Simulazione di guasti di tipo *transistor stuck-open*

Nello studio della rivelazione di guasti del tipo *transistor stuck-open* è molto importante tenere in conto i due fenomeni che possono dare luogo all'invalidazione di una sequenza di due vettori di collaudo la quale sarebbe, almeno in linea di principio, in grado di rivelare un transistore bloccato spento:

- la rimozione del valore di carica iniziale a causa dell'apertura di un cammino conduttivo nel passaggio fra i due vettori di collaudo;
- fenomeni di ripartizione di carica.

La verifica di queste condizioni mediante simulazione risulta complesso e, in pratica, non particolarmente significativo, infatti, queste dipendono entrambe da delicate considerazioni sulle tempistiche del circuito le quali sono a loro volta note con un certo margine d'incertezza. Quello che si preferisce fare è distinguere a priori fra le coppie di vettori di collaudo che possono dare luogo a problemi di invalidazione (*non-robust*)

Tipo di circuito	*stuck-at*	*transistor stuck-open*	*transistor stuck-on*	Cortocircuito	
				interni	esterni
gate/macro-gate statici	GL, SL	GL^*, SL	GL, SL	SL	GL, SL
generali (pass-transistor)	SL	SL	SL	SL	SL

* senza poter tenere in conto fenomeni di charge sharing

Tabella 8.1:

8.5.2 Guasti del tipo ritardo

Le sempre più stringenti specifiche sulla velocità dei circuiti digitali hanno posto l'attenzione su quei guasti che pur non alterando la risposta del circuitoin in condizioni di stato stabile ne condizionano il funzionamento dinamico. Come si è visto nel capitolo XXXX diversi tipi di malfunzionamento al possono dare luogo a un degrado delle prestazioni dinamiche di una parte di circuito (ad esempio i cortocircuiti). Al livello logico (in particolare a quello di *gate*), questo tipo di comportamento viene descritto mediante il modello di guasto del tipo ritardo (*delay fault*) (che comprende anche, come caso più semplice i guasti del tipo *transition fault*, ovvero quei guasti per cui non si verifica la transizione dell'uscita di un gate attesa nel circuito privo di guasti).

La simulazione di guasti del tipo *transition fault* si riconduce, in pratica, a quella di guasti del tipo *stuck-at*. Ad esempio, un guasto che impedisce la transizione $0 \rightarrow 1$ dell'uscita G di un *gate*, risulta rivelabile se il vettore di collaudo $k-1$-mo inizializza G a 0 e il k-mo è in grado di rivelare G *stuck-at-0* rendendone osservabile il valore a un uscita[15].

Quindi, la simulazione di guasti del tipo *transition fault* si riconduce al caso dei guasti del tipo *stuck-at*, con in più la necessità di tenere in conto il valore di ciascun segnale con il vettore di collaudo precedente a quello corrente (per questo motivo, tali guasti sono detti *stuck-at* di tipo condizionale).

In Fig. **??**a and b è illustrata la rivelazione di un guasto che impedisce la transizione $0 \rightarrow 1$ per due diverse sequenze di vettori di ingresso; come si vede nel primo caso si ha la propagazione (nel circuito fault-free) di una transizione dalla sede del guasto all'uscita, nel secondo caso, invece, il valore finale e quello iniziale dell'uscita coincidono (ma il guasto è comunque rivelato).

Passando al caso, più complesso, dei veri e propri guasti di tipo ritardo si è visto come questi siano caratterizzati da un parametro (*size*) che rappresenta

[15] Vale la pena ricordare che mentre il collaudo di guasti non parametrici è un processo tipicamente binario nel caso di guasti parametrici entrano in gioco considerazioni di tipo quantitativo.

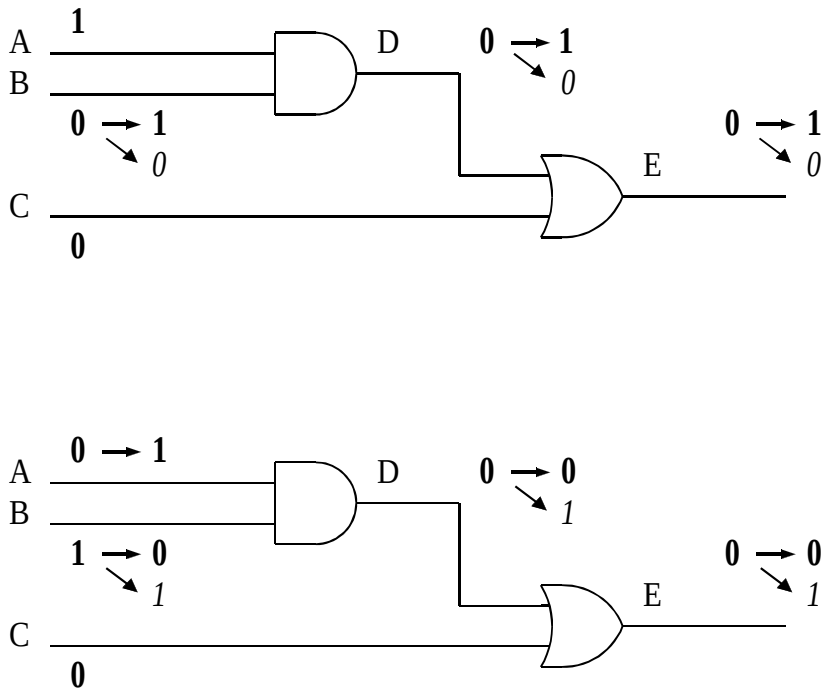


Figura 8.9: **Esempi di rivelazione di guasti di tipo *transition* in un semplice circuito combinatorio. I valori in grassetto indicano il valore delle linee (con i due vettori di collaudo applicati) nel circuito privo di guasti, mentre quelli in corsivo rappresentano (quando diversi dai precedenti) quelli che si avrebbero in presenza di un *transition fault* sulla linea B. Nel caso a) si tenta di propagare una transizione attraverso il gate guasto fino all'uscita; nel caso b), invece, il guasto è rivelato nonostante il valore iniziale e finale dell'uscita in assenza di guasti siano coincidenti.**

l'incremento, rispetto alle condizioni, nominali o di caso peggiore, dei ritardi di propagazione della porzione del circuito affetta dal guasto (singolo *gate* o cammino di propagazione, a seconda che si sia adottato il modello di tipo *gate-delay* o *path-delay*).

La rivelazione di questo tipo di guasti non dipende quindi soltanto dalla topologia del circuito e dall'insieme di vettori di collaudo applicati (come nel caso dei *transition-fault*), ma anche dal valore del parametro che li caratterizza e dalla "sensibilità" delle condizioni di collaudo al contorno (tempo di campionamento delle uscite etc.).

Chiaramente, l'effettivo *size* di un guasto di tipo ritardo dipende dalle cause che hanno dato luogo al guasto stesso ed è quindi tipicamente non noto, scopo della simulazione di guasto è pertanto determinare una soglia per il *size* del guasto oltre la quale il guasto è rivelabile. L'esempio di Fig. XXXX mostra chiaramente come questa soglia dipenda non solo dalle condizioni logiche create dai vettori di collaudo applicati al circuito (come nel caso dei *transition faults*), ma anche dalla posizione del guasto nel circuito, dalla distribuzione dei ritardi ed infine dall'intervallo di tempo che intercorre fra l'applicazione del vettore di collaudo e il campionamento delle uscite.

Appare quindi chiaro come la simulazione di guasto debba tenere in conto, innanzitutto le temporizzazioni del circuito, e pertanto non sono utilizzabili algoritmi di simulazione a ritardo unitario (vedi Sez. XXXX). Tipicamente vengono utilizzati algoritmi di simulazione di tipo event-driven.

Nel presentare brevemente le tecniche di simulazione per questo tipo di guasti, restringeremo l'attenzione al caso di circuiti combinatori, questo perchè in generale evventuali ritardi aggiuntivi nei *flip-flop* possono sempre essere inclusi nella parte combinatoria e in molti casi opportune tecniche di DFT riconducono il caso sequenziale a quello combinatorio. Inoltre, verrà considerato il modello di guasto di tipo *gate-delay*, anche se tutte le considerazioni che verranno svolte possono essere agevolmente estese al caso del modello di tipo *path-delay*.

Un ulteriore restrizione consiste nel non considerare quei guasti di tipo ritardo per cui non esiste solo una soglia inferiore per la valutazione del *size* ma anche una soglia superiore (questo è possibile se si cerca di rivelare uno di questi guasti sfruttando la propagazione di glitch nel circuito).

In questo contesto, per determinare se un guasto di tipo ritardo (ad esempio all'uscita di un gate) è rivelato o meno è necessario considerare due vettori di collaudo $< V_1, V_2 >$ il primo dei quali deve inizializzare tale segnale e il secondo deve provocare un opportuna transizione dell'uscita di tale gate rendendo osservabile il valore finale di tale segnale alle uscite, in maniera del tutto analoga a quanto visto nel caso di guasti di tipo *transition*.

VERIFICARE BENE

8.5.3 Simulazione di guasti al livello comportamentale

In presenza di circuiti di grandi dimensioni (a partire da quelli contenenti decine di migliaia di porte logiche), la descrizione in termini di *gate* elementari (e maggior ragione di transistori) dà luogo a problemi di efficienza della simulazione. In questo caso, è conveniente descrivere il comportamento di interi blocchi del circuito in maniera funzionale[16] piuttosto che fornirne la struttura circuitale.

I vantaggi di un approccio di questo tipo sono particolarmente evidenti nel caso, computazionalmente più complesso, della simulazione di guasto, anche se l'approccio *behavioral* pone inevitabilmente dei seri problemi per quanto riguarda l'accuratezza nei modelli di guasto utilizzabili. Infatti, tradizionalmente, quando si opera al livello comportamentale (*behavioral*), il modello più largamente utilizzato è quello di guasti di tipo *stuck-at* singolo sulle linee di interconnessione tra blocchi, che non è però rappresentativo dei guasti interni ai blocchi [26] (a questo proposito vale la pena di osservare come, se anche tutti i guasti sugli ingressi e sulle uscite di un blocco combinatorio vengono rivelati, è possibile che non siano stati rivelati tutti i guasti interni).

Il problema principale, della simulazione di guasti al livello comportamentale tuttora oggetto di ricerche, è quindi quello di trovare modelli di guasto che siano sufficientemente rappresentativi di quanto avviene a un minore livello di dettaglio. Al riguardo sembrano essere promettenti quegli approcci che, a partire da una descrizione "strutturale" di un blocco, estraggono non solo il comportamento del circuito in assenza di guasti ma anche quello in presenza di guasti [27].

8.6 Conclusioni

Come descritto nel lavoro, la simulazione di guasto rappresenta una delle operazioni fondamentali per la messa a punto di efficaci programmi di collaudo dei circuiti integrati digitali.

Questo tipo di simulazione, d'altra parte, risulta estremamente impegnativo sotto il profilo dei tempi di calcolo, soprattutto a causa del grande numero di casi da trattare. Ciò, dunque, richiede algoritmi in grado di ottimizzare l'utilizzo dei calcolatori, nonchè rappresentazioni piuttosto semplificate dei circuiti e dei loro guasti, anche se ciò pone necessariamente il problema dell'accuratezza dei risultati (infatti, alcuni tipi di guasto possono essere descritti accuratamente solo a livelli più vicini a quello elettrico).

Da questo punto di vista, è facile prevedere che la soluzione al problema della simulazione di guasti nei circuiti integrati digitali potrà difficilmente essere trovata ad un unico livello, ma richiederà programmi di simulazione di tipo ger-

[16] Ovvero descrivendone il comportamento tramite procedure la cui sintassi ricorda quella tipica dei linguaggi ad alto livello. Ad esempio, *if (s=0) then out:=a else out:=b;* descrive il funzionamento di un *Multiplexer* a due vie.

archico, in cui sia possibile simulare differenti parti del circuito a diversi livelli di astrazione.

Infine, rimane da notare come la simulazione di guasto dia attualmente luogo a una notevole attività di ricerca. In particolare, essendo ormai acquisite tecniche sufficientemente efficienti a livello *gate*, i maggiori sforzi si indirizzano sia verso un incremento di accuratezza dei simulatori (per poter trattare modelli di guasto sufficientemente realistici), sia verso la ricerca di strategie di tipo gerarchico in grado di sfruttare i vantaggi forniti dai diversi livelli di simulazione.

Bibliografia

[1] M. Abramovici, M. Breuer, and A. Friedman, *Digital Systems Testing and Testable Design*. : Comp. Science Press, 1990.

[2] T. Williams and N. Brown, "Defect Level as a Function of Fault Coverage," *IEEE Transaction on Computers*, vol. 30, pp. 987 – 988, 1981.

[3] P. H. Bardell, W. H. McAnney, and J. Savir, *Built-In Test for VLSI - Pseudorandom Techniques*. New York: John Wiley & sons, 1987.

[4] J. A. Abraham and W. K. Fuchs, "Fault and Error Models for VLSI," *Proc. of the IEEE*, vol. 74, pp. 639 – 654, 1986.

[5] C. Lin and S. M. Reddy, "On Delay Fault Testing in Logic Circuits," in *Proc. of IEEE Int. Conf. On Computer Aided Design*, pp. 148 – 151, 1986.

[6] S. Koeppe, "Modeling and Simulation of Delay Faults in CMOS Logic Circuits," in *Proc. of IEEE Int. Test Conf.*, pp. 530 – 535, 1986.

[7] R. L. Wadsack, "Fault Modeling and Logic Simulation of CMOS and NMOS Integrated Circuits," *Bell Syst. Tech. J*, vol. 57, pp. 1449 – 1474, 1978.

[8] K. Mei, "Bridging and Stuck-at Faults," *IEEE Transaction on Computers*, vol. C-23, pp. 720 – 727, 1974.

[9] J. Acken, "Testing for Bridging Faults (shorts) in CMOS Circuits," in *Proc. of Design Automation Conf.*, pp. 717 – 718, 1983.

[10] W. Maly, "Realistic Fault Modeling for VLSI Testing," in *Proc. of Design Automation Conf.*, pp. 173 – 180, 1987.

[11] E. Thompson and S. Szygenda, "Digital Logic Simulation in a Time-Based, Table-Driven Environment; Part 2. Parallel Fault Simulation," *Computer*, vol. , pp. 38 – 49, Mar 1975.

[12] D. Armstrong, "A Deductive Method for Simulating Faults in Logic Circuits," *IEEE Transaction on Computers*, vol. C-21, pp. 464 – 471, May 1972.

[13] E. Ulrich and T. Baker, "The Concurrent Simulation of Nearly Identical Digital Networks," in *Proc. of Design Automation Conf.*, pp. 145 – 150, 1973.

[14] M. Abramovici, P. R. Menon, and D. Miller, "Critical Path Tracing: an Alternative to Fault Simulation," *IEEE Design & Test*, vol. 1, pp. 83 – 92, Feb 1984.

[15] P. Menon, Y. Levendel, and M. Abramovici, "Critical Path Tracing in Sequential Circuits," in *Proc. of IEEE Int. Conf. On Computer Aided Design*, pp. 162 – 165, 1988.

[16] J. K. T.T. Butler, T.G. Hallin and K. Johnson, "LAMP: Application to Switching Systems Development," *Bell Syst. Tech. J*, vol. 53, pp. 1535 – 1555, 1974.

[17] R. L. Wadsack, "Design Verification and Testing of the WE32100," *IEEE Design & Test*, vol. 1, pp. 66 – 75, 1984.

[18] C. Chin and E. McCluskey, "Test Length for Pseudorandom Testing," *IEEE Transaction on Computers*, vol. 36, pp. 252 – 256, 1987.

[19] W. Cheng and M. Yu, "Differential Fault Simulation for Sequential Circuits," *J. of Electronic Testing Theory and Applic.*, vol. 1, pp. 7 – 14, 1990.

[20] M. Abramovici and P. R. Menon, "A Practical Approach to Fault Simulation and Test Generation for Bridging Faults," *IEEE Transaction on Computers*, vol. C-34, pp. 658 – 663, Aug 1985.

[21] S. Millman and E. McCluskey, "Detecting Bridging Faults with Stuck-at Test Sets," in *Proc. of IEEE Int. Test Conf.*, pp. 773 – 778, 1988.

[22] R. Bryant, "A Switch-Level Model and Simulator for MOS Digital Systems," *IEEE Transaction on Computers*, vol. 33, pp. 160 – 177, 1984.

[23] J. P. Hayes, "Pseudo Boolean Logic Circuits," *IEEE Transaction on Computers*, vol. C-35, pp. 602 – 612, Jul 1986.

[24] R. Bryant, "A Survey of Switch-Level Algorithms," *IEEE Design & Test*, vol. , pp. 26 – 40, 1987.

[25] J. Hayes, "An Introduction to Switch-Level Modeling," *IEEE Design & Test*, vol. , pp. 18 – 25, 1987.

[26] S. Hong, "Fault Simulation Strategy for Combinational Logic Networks," in *Proc. of Int. Fault Tolerant Comp. Symp.*, pp. 96 – 99, 1978.

[27] D. Blaauw and et al., "Automatic generation of behavioral models from switch-level descriptions," in *Proc. of Design Automation Conf.*, pp. 179 – 184, 1987.