특 집

고장시뮬레이션의 가속화를 위한 기술동향

宋 五 永

中央大學校 制御計測工學科

I. 서 론

VLSI기술이 점점 더 발달되어 감에 따라서 게이트 하나당 설계 비용은 감소되고 있으며, 게이트 하나당 테스트 비용은 증가되고 있다. 테스트 비용을 낮추기 위해서 LSSD[6]와 같은 테스트를 위한 설계(design for testability) 기법들이 폭 넓게 사용되고 있다. 이러한 기법들에 의해서 신뢰성 높은 VLSI제품들이 출하될 수 있으며 테스트 및 디버깅 비용을 감소시킬 수 있다. 또한 테스트패턴 자동발생(automatic test pattern generation)과 고장시뮬레이션(fault simulation)을 이용함으로써 디지탈회로들의 테스트를 보다 더 쉽게 할 수 있다. 디지탈회로에서의 고장검출(fault detection)은 주로 테스트패턴을 발생시켜서 이를 사용하는 기술에 의존한다. 테스트패턴은 고장검출율(fault coverage)에 관한 요구조건이 만족될 때까지 계속적으로 발생되어져야 하고 평가되어져야 한다. 여기서 고장검출율이란 시뮬레이션을 위해 고려하고 있는 모든 고장들의 숫자에 대한 검출된 고장들의 숫자의 비율로써 정의된다. 주어진 테스트패턴들의 평가는 고장시뮬레이션에 의해 수행되어진다. 또한 고장시뮬레이션은 고장목록사전(fault dictionary)을 구축하여 고장위치검색(fault location)을 위해서 사용될 수 있다. 여러 가지 고장들에 대한 시뮬레이션 결과로써 얻어진 반응패턴(response pattern)들이 데이터베이스에 저장된다. 이렇게 저장된 반응패턴과 실제 회로로부터 관찰된 실제패턴(actual pattern)을 비교함으로써 가장 발생 가능한 고장을 추론해 낼 수 있다.

고장시뮬레이션은 테스트패턴 발생과정에서 필수 불가결하다. 디지탈회로 혹은 IC를 생산하는 비용 가운데 상당한 비용이 테스트에 기인한다는 사실을 고려할 때 설계－테스트－재설계의 순환과정의 비용을 줄이기 위해서는 가능한 한 짧은 시간내에 고장시뮬레이션을 수행하는 것이 중요하다. 고장시뮬레이션은 매우 큰 VLSI chip들에 대해서 많은 양의 컴퓨터 자원(CPU 수행시간, 기억장치)을 필요로 한다는 점에서 VLSI개발 과정에서 가

장 어려운 작업중의 하나이다.

고장시뮬레이션을 가속화하는 방법들은 일반적으로 다음 세 가지 종류로 분류될 수가 있다: (가) 게이트 레벨 조합형 혹은 동기식 순차회로와 같은 특별한 종류의 회로에 대한 효과적인 알고리듬들, (나) 범용 멀티프로세서를 위한 알고리듬들, (다) 고장시뮬레이션 가속을 위한 하드웨어 가속기.

테스트 목적 때문에 순차회로를 조합회로로 바꾸어 주는 LSSD[6]기법과 같은 테스트를 위한 설계 기법의 폭 넓은 사용으로 말미암아 조합회로를 시뮬레이션하는 특별한 기법이 매우 중요해졌다. 조합회로에 대해서 고장시뮬레이션의 시간복잡도(time complexity)는 O(n2)이다. 여기서 n은 회로에서 게이트들의 숫자이다[7]. 이러한 시간복잡도는 VLSI회로에서는 매우 높은 것으로 조합회로들을 위한 매우 효율적인 고장시뮬레이션 기법들이 필요하다.

완전스캔(full-scan) 설계는 면적과 시간지연과 같은 오버헤드를 초래하므로 모든 회로가 완전스캔을 채용하여 설계되어지지 않는다. 그러므로 순차회로들을 위한 VLSI CAD tools들 또한 필요하다. 고장시뮬레이션에 있어서 컴퓨터자원에 대한 요구도(Computer resource reguirement)에 관한 문제는 순차회로들에 대해서 훨씬 더 심각하므로 순차회로에 대한 매우 효율적인 고장시뮬레이션이 필요하다.

일반적인 회로들에 대해서 비교적 간단한 시뮬레이션 알고리듬보다 특별한 종류들로 분류될 수 있고 널리 사용되는 회로에 대한 알고리듬을 개발함으로써 고장시뮬레이션의 효율을 개선할 수가 있다. 특별한 목적의 하드웨어 가속기는 초기 설계와 유지에 있어서 많은 비용을 필요로 하며 때때로 다른 고장시뮬레이션 알고리듬의 사용을 불가능하게 할 수도 있다, 범용 멀티프로세서가 많이 개발되고 있기 때문에 이것은 고장시뮬레이션을 위한 매력적인 컴퓨터가 될 수도 있다.

본 논문에서는 일반적인 회로들에 대해서 적용가능한 전형적인 고장시뮬레이션 방법들에 대한 소개와, 위에서 분류한 가속을 위한 세가지 방법 중에서 처음 두가지 방법들에 대해서 기술 동향을 설명할 것이다.

II. 전형적인 방법

일반적인 디지탈회로들에 적용 가능한 전형적인 고장시뮬레이션 방법은 다음과 같이 3가지가 있다 : 병렬적 고장시뮬레이션(parallel fault simulation), 추론적 고장시뮬레이션(deductive fault simulation), 동시적 고장시뮬레이션(concurrent fault simulation).

병렬적 고장시뮬레이션[25]은 사용중인 컴퓨터의 단어크기(word size)만큼의 고장회로(faulty circuit)들을 동시에 시뮬레이션한다.

그 방법은 게이트 출력 계산(gate evaluation)을 할 때 비트단위에 기반을 둔 논리 인스트럭션을 사용한다. 각각의 비트는 각각의 다른 회로들과 대응한다. 원래의 회로와 (n-1)개의 고장회로들이 그룹을 이뤄서 병렬적으로 시뮬레이션된다. 여기서 n 이란 사용된 컴퓨터의 단어크기이다. 이러한 과정은 모든 고장회로들에 대해서 시뮬레이션이 완성될 때까지 반복된다. 따라서 이는 여러 번의 패스에 의존하는 방법으로써 다음 두 가지의 단점을 갖고 있다 : 시뮬레이션이 반복될 때마다 원래의 회로에 대한 시뮬레이션은 다시 수행되기 때문에 이는 낭비적이다. 또한 어떤 고장이 검출된 후에는 그 고장에 대해서 더 이상 시뮬레이션을 할 필요가 없는 경우(fault dropping)에는 그 고장의 검출 후부터의 시뮬레이션은 불필요함에도 불구하고 수행되기 때문에 낭비적이다.

추론적 고장시뮬레이션[5]은 원래의 회로를 무고장 상태(fault free state)로 시뮬레이션한다. 그 다음 현재 상태에서 검출될 수 있는 모든 고장들을 현재의 무고장 상태로부터 추론한다. 이 방법은 이런 방식으로 동시에 모든 검출 가능한 고장들을 계산해내기 때문에 각각의 테스트패턴에 대해서 각각의 패스로 시뮬레이션을 수행한다.

회로 상에 있는 각각의 선은 하나의 고장목록

(fault list)을 갖는데, 이 고장목록은 그 선 위에서 에러로 나타날 수 있는 고장들로 구성된다. 이것이 의미하는 바는 다음과 같다. 시뮬레이션을 수행하는 동안 주어진 시간에 어떤 고장이 고장목록에 없다면 그 선 위의 신호값은 그 고장으로 인한 어떠한 영향을 받지 않았다는 것을 뜻한다. 어떤 게이트의 무고장 상태에서의 출력값은 하나 이상의 입력값이 바뀔 때마다 계산된다. 그 출력단의 고장목록은 무고장 상태에서 적어도 하나의 입력단의 신호값이 바뀌거나 적어도 하나의 입력단의 고장목록이 바뀔 때마다 계산된다.

동시적 고장시뮬레이션[26]은 추론적 고장시뮬레이션과 마찬가지로 각각 테스트패턴에 대해서 하나의 패스에 의한 시뮬레이션을 수행한다. 동시적 고장시뮬레이션은 무고장 상태의 회로와 모든 고장회로들을 동시에 시뮬레이션한다. 출력값 계산의 횟수를 줄이기 위해서 무고장 상태에서의 신호값과 고장 상태에서의 신호값이 다를 때만 고장회로의 출력값을 계산한다.

하나의 회로는 요소(element)들과 그 요소들의 연결로서 구성된다. 하나의 고장상태목록(faulty state list)은 회로에서 각각 하나의 요소와 관련된다.

그 고장상태목록은 에러를 야기하는 모든 고장들의 이름과 그 선 상의 고장들의 신호값으로 구성된다. 추론적 고장시뮬레이션은 변동된 고장목록에 있는 모든 고장들에 대해서 집합연산(set operation)을 수행하는 반면에 동시적 고장시뮬레이션은 이미 변동된 고장 상태를 갖는 고장회로들만을 계산할 필요가 있다. 이 두 가지 방법 모두 현재 기술수준으로 설계되고 있는 큰 회로들에 대해서 많은 기억용량을 필요로 하는, 목록을 유지해야 하는 단점이 있다. 다음은 세 가지 방법을 논리값들의 집합, 함수적 모델링(functional modeling), 지연시간 모델링과 기억용량 요구도등의 관점에서 비교할 것이다.

병렬적 고장시뮬레이션과 추론적 고장시뮬레이션은 스위칭 대수(switching algebra)로 아주 잘 정의되어 있기 때문에 이런 것들은 이진논리(two-valued logic)에 대해서 효율적이다. 이런 방법들이 삼진논리(three-valued logic)를 위해 간단히 확장될 수 있을 지라도 이진논리에 비하면 이런 방법들이 필요로 하는 계산요구도(computational requirement)는 매우 크다. 삼진논리보다 더 복잡한 논리에 대해서도 확장은 가능하나 매우 복잡하다[1, 12, 13]. 이런 방법들이 함수적 블럭(funcion block)들을 다룰 수 있을 지라도[16, 13], 이들은 오직 스위칭 함수들의 집합으로 변환가능한 함수적 블럭만을 허용한다.

동시적 고장시뮬레이션은 무고장회로와 고장회로들 사이의 차이만을 표현하고 각각의 회로들, 즉 무고장회로와 고장회로들을 각각으로 시뮬레이션하기 때문에 이 방법은 논리값 집합과 무관하며 함수적 모델링에 어떠한 제한도 받지 않는다. 병렬적 고장시뮬레이션과 추론적 고장시뮬레이션에 있어서 개별적인 고장들과 신호값 상승지연(rise delay)과 신호값 하강지연(fall delay)들을 관련시킴으로써 자세한 시간지연모델의 사용이 가능하다[11, 13]. 그러나 무고장회로 신호값과 고장회로 신호값 사이의 밀접한 연관성 때문에 자세한 시간지연모델은 많은 계산을 필요로 한다. 동시적 고장시뮬레이션에서는, 각각의 고장의 시간지연은 독립적으로 취급될 수 있기 때문에 다른 방법들에 비해서 자세한 시간지연에 대한 추가적인 계산은 그리 크지 않다.

병렬 고장시뮬레이션에 있어서 회로와 하나의 패스동안 시뮬레이션되어질 고장들이 주어지면 필요한 컴퓨터 기억용량은 일정하게 결정되어질 수 있다. 비교적 큰 숫자의 고장들은 기억용량을 증가시키지 않더라도 여러 번의 패스에 의해 시뮬레이션된다. 추론적 고장시뮬레이션과 동시적 고장시뮬레이션에 있어서 기억용량 요구도는 시뮬레이션을 하기 전에 미리 측정될 수가 없다. 왜냐하면 이는 회로의 활성도(activity)와 순차도(sequentiality)에 의존하기 때문이다.

두가지 방법 모두 병렬적 고장시뮬레이션보다 많은 기억용량을 필요로 한다.

이러한 세가지 고장시뮬레이션 방법들은 조합회로, 동기식 순차회로, 비동기식 순차회로등을 포함하는 일반적인 종류의 회로들에 적용 가능하다.

III. 가속화 방법

1. 조합회로를 위한 방법

테스트 목적 때문에 순차회로를 조합회로로 변환시켜주는 테스트를 위한 설계방법인 완전 스캔이 폭 넓게 사용됨에 따라 조합회로들에 대해서 전문화된 고장시뮬레이션에 대한 연구가 진행되어 왔다. 그런 종류의 회로에 있는 stuck-at 고장들을 테스트하기 위해서는 그 회로의 출력은 그 회로가 최종값으로 안정화된 후에만 관찰되어진다. 출력의 최종값들은 테스트입력패턴들이 회로에 인가되는 순서 혹은 내선(internal line)상의 신호값이 바뀌는 순서에 따라서 값이 바뀌지 않는다. 그러므로 단위지연(unit delay) 혹은 무지연(zero delay)과 같은 단순화된 시간지연모델이 사용 가능하다. 또한 사용중인 컴퓨터의 단어크기(computer word size)만큼의 테스트패턴들을 병렬화하여 동시에 게이트들의 출력을 계산할 수 있다. 이런 특성들은 조합회로를 위해 전문화된 고장시뮬레이션에 부분적으로 이용되어졌다.

Hong[9]은 조합회로들에 대해서는, 회로 종류에 제한을 받지 않는 전형적인 방법보다 훨씬 빠른 고장시뮬레이션을 제시하였다. 그 방법은 지선(fanout stem)들 위에 있는 고장들에 대해서는 고장시뮬레이션을 수행하고 회로의 무분기선 영역(fanout-free region) 내에서는 후진추적(backt-racing)을 수행한다. 고장시뮬레이션의 계산복잡도는 $O(n^2)$인 반면에 후진추적의 계산복잡도는 $O(n)$이다. 여기서 n이란 게이트의 숫자이다. 회로의 어떤 무분기선 영역 내에서의 후진추적을 행함으로써 고장의 영향(fault effect)이 무분기선 영역의 지선(the stem of the fanout-free region)까지 전달되는, 무분기선 영역 내의 고장들을 알아낸다. 지선 위의 고장들의 검출여부가 한번 결정되면 무분기선 영역 내의 고장들의 검출여부는 쉽게 결정될 수 있다. 왜냐하면 무분기선 영역에는 어떠한 재집결점(reconvergence)이 없기 때문이다. 주어진 테스트패턴 하에서 무분기선 영역 내의 어떠한 고장도 그것의 지선까지 전달되지 않는다면 무분기선 영역 내의 고장들을 대표하는, 그것의 지선위의 고장들은 시뮬레이션이 될 필요가 없다. 추론적 방법[5]이 지선 위의 고장들을 시뮬레이션하기 위해 사용되었다.

Waicukauski[27]는 조합회로에 대한 매우 효과적인 고장시뮬레이션 방법을 제시하였다. 그 방법은 사용중인 컴퓨터의 단어크기를 기반으로 하는 병렬패턴 계산(parallel pattern evaluation)의 개념과 단일 고장 전송(single fault propagation)[24]에 근거한다. 그 방법은 "병렬패턴 단일 고장 전송"(parallel pattern single fault propagation 혹은 PPSFP)라고 불리며 여러 번의 패스에 의한 시뮬레이션을 수행한다. PPSFP는 n개의 테스트패턴을 그룹핑하여 벡터화한다. 무고장회로에 대해서는 병렬패턴(parallel pattern)이라 불리는 벡터화된 패턴을 이용하여 병렬적으로 시뮬레이션한다. 여기서 n은 사용중인 컴퓨터의 단어크기이다. 시뮬레이션될 고장들의 집합으로부터 고장이 하나씩 선택되어 회로에 주입된다(fault injection). 주입된 고장의 효과는 고장의 위치로부터 병렬패턴에 대해서 주요출력단쪽으로 전송된다. 이러한 단일 고장 전달과정(the process of single fault propagation)은 고장들의 집합 내의 모든 고장들이 고장시뮬레이션될 때까지 반복된다. 이렇게 반복되는 전 과정은 n개의 테스트패턴으로 구성된 각각의 벡터에 대해서 반복된다.

Antreich와 Schulz[4]는 Hong의 방법과 PPSFP[27] 방법의 개념을 이용하였다. 그 방법은 조합회로들에 대해서 가장 효율적인 고장시뮬레이션 방법들 중의 하나이다. PPSFP의 효율성은 여러 개의 입력패턴을 사용중인 컴퓨터의 단어크기 만큼씩 그룹핑하여 병렬적으로 처리함으로써 얻어진다. 무고장회로와 각각의 고장회로들은 별도로 고장시뮬레이션된다. Antreich와 Schulz에 의해 제시된 알고리듬에서는 PPSFP와 마찬가지로 사용중인 컴퓨터의 단어크기와 같은 숫자만큼의 테스트패턴들을 동시에 처리한다. 이 방법은 병렬패턴 지선추적(parallel pattern stem-based tracing 혹은 PPST) 알고리듬이라 불린다. 지선(fanout

stem)으로부터의 후진추적과 지선고장시뮬레이션도 사용중인 컴퓨터의 단어크기 만큼의 테스트패턴에 대해서 병렬적으로 수행된다. 또한 고장 검출 여부를 결정하기 위해서 분석되어질 지선들의 숫자는 다음의 이유로 감소될 수 있다 : 어떤 무분기선 영역 내의 어떤 고장도 병렬패턴 내의 한 테스트패턴에 대해 무분기선 영역의 지선까지 전송될 수 없다면 그 지선 위에 있는 고장은 주어진 테스트패턴에 대해서 시뮬레이션될 필요가 없다.

Maamari와 Rajski[14, 15]는 지선 영역(stem region)과 탈출선(exit line)의 개념을 사용하였다. 지선과 그 지선의 마감 재집결 게이트(closing reconvergent gate) 사이의 모든 게이트들로 지선 영역을 형성한다. 지선 영역을 벗어나면서 그 지선 영역으로 부터의 어떤 분기선(fanout)과도 재집결하지 않는 선들을 탈출선이라 부른다. 어떤 지선 고장의 검출 여부는 그 고장의 탈출선에 대한 고장효과 전송 여부와 탈출선의 주요출력단에 대한 고장효과 전송여부에 의해서 결정되어질 수 있다. 그러나 그들은 기존의 방법들보다 더 효율적이라는 것을 실험을 통해 검증을 하지는 않았다.

2. 동기식 순차회로를 위한 방법

동기식 순차회로에 대한 후진추적 방법들(trace-based methods)[8, 17, 28, 18]은 동기식 순차회로를 반복 배열 모델(iterative array model)로 바꾼다. 반복 배열 모델에서는 "셀"이라고 불리는 조합회로 부분들이 반복적으로 상태변수(state variables)들을 통해서 연결된다. 이런 후진추적 방법들은 각각의 셀에서는 후진추적을 수행하며, 셀 간에는 한 셀에서 다음 셀로 고장효과들을 전송하는 방법에 근거하고 있다. 다음으로 진행하기 전에 간단한 용어의 정의를 한다. "시간후레임 t"란 시간적으로 t번째에 대응하는 셀을 뜻한다.

SCRIPT[17, 18]는 시간후레임 t에 있는 상태변수들에 도달된 모든 고장들에 대해서 한번에 하나씩 고장들의 효과를 시간후레임 t+1로 전송한다. 시간후레임 t−1로부터 전송되지 못한 고장들의 관찰도(observability)를 결정하기 위해서 임계경로 추적을 수행한다. 보다 더 효율적인 방법들이 다음과 같이 제시되었다 : (가) 상태변수들의 어떤 부분집합에 도달한 고장들을 동시에 처리한다. (나) 어떤 상태변수들에 도달하였으나 고려중인 시간후레임에서 동일한 고장들과 상호작용을 하지 않는 고장을 분류하여 그러한 고장들의 관찰도를 임계경로추적방법에 의해 결정한다. SCRIPT는 수 개의 순차회로들에 대한 실험을 통해서 동시적 고장시뮬레이션보다 빠름을 보여주었다.

Hill[8, 28]은 모든 신호값이 이진값을 갖는다는 전제하에서 순차회로에 대해서 후진추적 방식에 근거한 고장시뮬레이션방법을 제시하였다. 어떤 상태변수들에 도달한 고장들에 대해서 추론적 고장시뮬레이션을 사용하였다.다른 종류의 고장들은 다음 방법으로 시뮬레이션된다 : 모든 지선고장들의 관찰도(observability)를 결정하기 위해서 병렬적 고장시뮬레이션을 사용하였고, 다른 종류의 고장들의 관찰도를 결정하기 위해서는 후진추적 방법을 사용하였다. Hill은 그 방법이 병렬적 고장시뮬레이션보다 빠름을 가산회로들에 대한 실험을 통해서 보여주었다.

PROOFS[21]라는 방법에서는 동기식 순차회로에 대한 반복 배열 모델이 사용된다. 사용 중인 컴퓨터의 단어크기 만큼의 병렬성을 이용하기 위하여 동적 고장 그룹핑(dynamic fault grouping)을 하여 그 단어크기의 숫자 만큼의 고장들을 병렬적으로 시뮬레이션한다. 현재의 시간후레임에서 활동적(active)이긴 하나, 바로 전의 시간후레임에서는 검출되지 않은 고장들만 그룹핑함으로써 컴퓨터의 단어에 있어서 비트들이 불필요하게 사용됨을 방지하였다. 상태변수들에서는 오직 고장회로들에 대한 신호값만을 저장한다. 고장주입 시에는 활동적인 고장들을 주입하기 위해, 비트마스크(bit masks)들을 사용하지 않고 회로를 약간 수정하였다. PROOFS는 다진논리값(multivalued logic)을 취급한다 : 0,1,X(unknown), 그리고 Z(high impedance). 그 방법은 동시적 고장시뮬레이션보다 시뮬레이션 시간과 주기억 용량의 관점에서 더 효율적인 것을 실험을 통해 보여주었다.

3. 범용 멀티프로세서를 사용하는 방법

최근에 범용 멀티프로세서를 이용하여 고장시뮬레이션을 수행하는 방법에 관한 연구들이 있었다. Duba[21]는 계층적 동시적 고장시뮬레이션을 프로세서들이 네트워크를 통해서 연결된 분산환경에서 실현하였다. 그 방법에서 고장들을 나누고 나누어진 고장들로 이루어진 각각의 부분집합을 각각 하나의 프로세서에 할당함으로써 고장시뮬레이션의 문제를 여러 개의 작은 태스크로 분리하였다. 고장들을 나누는 방법과 사용된 프로세서들의 숫가가 적절히 선택될 경우 스피드개선은 거의 최적이었다는 것을 세개의 곱셈회로들에 대한 실험을 통해서 보여주었다.

동시적 고장시뮬레이션은 공유기억장치를 갖는 멀티프로세서에서 구현된 적도 있다[43]. 그 방법에서 회로는 작은 여러개의 부회로를 나뉘어지고 각각의 부회로는 각각의 프로세서에 할당된다. 각각의 프로세서는 각각의 부회로에 대해 동시적 고장시뮬레이션을 수행한다. 부회로는 서로 중첩되는 부분이 존재하므로 중첩된 부분들에 대한 시뮬레이션은 낭비적이다. 벡터화된 병렬적 고장시뮬레이션 알고리듬이 Cray X-MP 슈퍼컴퓨터에서 구현되었다[47]. 병렬적 고장시뮬레이션 알고리듬이 Connection Machine에서 구현된 적도 있었다[4, 44]. Ishiura[29]는 벡터슈퍼컴퓨터에서 병렬적 고장시뮬레이션을 구현하였다. 이 방법에서 병렬성을 얻기 위해 각각의 벡터를 형성하기 위해 고장그룹핑및 패턴그룹핑을 사용하였다.

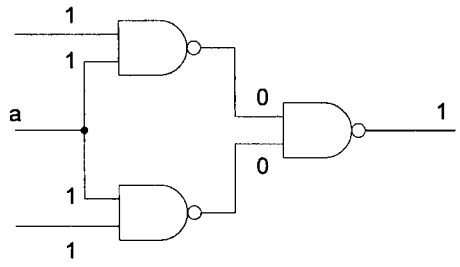

Figure 1 : 임계경로추적

IV. 맺음말

고장시뮬레이션은 테스트패턴의 평가 및 테스트패턴 생성에 주로 이용되는 것으로서 VLSI 테스트 분야에서 핵심적인 기술 중의 하나이다. 일반적인 모든 종류의 회로들에 적용될 수 있는 고장시뮬레이션 알고리듬인 동시적 고장시뮬레이션은 고장의 종류, 지연시간 모델, 함수적 블록 등에 구애받지 않기 때문에 현재에도 많이 쓰이고 있는 것 중의 하나이다. 조합회로 혹은 동기식 순차 회로와 같은 특별한 종류의 회로에만 잘 적용되는 고장시뮬레이션 알고리듬의 개발로 인하여 시간적으로 매우 효율적인 고장시뮬레이션을 수행할 수 있게 되었다. 그러나 주문형 반도체의 집적도가 급증해감에 따라 고장시뮬레이션을 수행해야 할 회로의 크기도 급격히 커지고 있어서 계속적인 연구가 필요하다. 또한 일반적인 종류의 회로들에 적용될 수 있는 고장시뮬레이션 방법에 있어서 동시적 고장시뮬레이션의 개발 이후에 뚜렷한 성과를 보여줄 만한 방법의 개발이 현재까지는 없다. 이 분야에 있어서도 보다 더 많은 좋은 연구결과를 기대한다.

참 고 문 헌

[1] Abramovici, M., Breuer, M A., and Kumar, K., "Concurrent Fault Simulation and Functional Modeling," *Proc. 14th Des. Autom. Conf.*, pp. 128-137, Jane 1977.

[2] Abramovici, M., Menon, P.R., and Miller, D.T., "Critical Path Tracing : An Alternative to Fault Simulation," *IEEE Design & Test of Computers*, vol. 1, no. 1, pp. 89-93, February 1984.

[3] Agrawal, A. and Bhattacharya, D., "CMP3F : A High Speed Fault Simulation for the Connection Machine," *Proc. 1990*

Intl. Test Conf., pp. 410-416, September 1990.

[4] Antreich, K.J. and Schulz, M.H., "Accelerated Fault Simulation and Fault Grading in Combinational Circuits," *IEEE Trans. CAD*, vol. CAD-6, pp. 704-712, September 1987.

[5] Armstrong, D.B., " A Deductive Method of Simulation Faults in Logic Circuits," *IEEE Trans. Comput.*, vol C-21, pp. 464-471, May 1972.

[6] Eichelberger, E.B. and Williams, T.W., "A Logic Design Structure for LSI Testability," *Proc. 14th Des. Autom. Conf.*, pp. 462-468, June 1977.

[7] Goel, P., "Test Generation Costs Analysis and Projections," *Proc. 17th Des. Autom.* Conf., pp. 77-84, 1980.

[8] Hill, F.J. Abuelyamen, E., Huang, W.K., and Shen, G.-Q., "A New Two Tast Algorithm for Clock Mode Fault Simulation in Sequential Circuits," *Proc. 25th Des. Autom. Conf.*, pp. 583-586, June 1988.

[9] Hong, S.J., "Fault Simulation Strategy for Combinational Logic Networks," *Proc. 8th Intl. Symp. Fault Tolerant Computing*, pp. 96-99, June 1979.

[10] Ishiura, N. Ito, M., and Yajima, S., "Dynamic Two-Dimensional Parallel Simulation Technique for High-Speed Fault Simulation on a Vector Processor," *IEEE Trans. CAD*, vol. 9, pp. 868-875, August 1990.

[11] Kjelkerud, E. and Thessen, O., "Techniques for Generalized Deductive Fault Simulation," *J. Des. Autom. Fault Tolerant Comput.*, vol. 1, pp. 377-390, October 1977.

[12] Levendel, Y.H. and Schwartz, W.C., "Impact of LSI on Logic Simulation," *Proc. Spring COMPCON78*, San Francisco, February 1978.

[13] Levendel, Y.H. and Menon, P.R., "Fault Simulation Methods-Extensions and Comparison," *Bell Syst. Tech. J.*, vol. 60, no. 9, pp. 2235-2258, November 1981.

[14] Maamari, F. and Rajski, J., "A Method of Fault Simulation Based on Stem Regions," *IEEE Trans. CAD*, vol. 9, pp. 212-220, February 1990.

[15] Maamari, F. and Rajski, J., "The Dyanmic Reduction of Fault Simulation," *Proc. 1990 Intl. Test Conf.*, pp. 801-808, September 1990.

[16] Menon, P.R. and Chappell, S.G., "Deductive Fault Simulation with Functional Blocks," *IEEE Trans. Comput.*, vol. C-27, pp. 687-695, August 1978.

[17] Menon, P.R., Levendel, Y., and Abramovici, M., "Critical Path Tracing in Sequential Circuits," *Proc. Intl. Conf. CAD*, pp. 162-165, November 1988.

[18] Menon, P.R., Levendel, Y., and Abramovici, M., "SCRIPT : A Critical Path Tracing Algorithm for Synchronous Sequential Circuits," *IEEE Trans. CAD*, vol. 10, pp. 738-747, June 1991.

[19] Mueller-Thuns, R.B., Saab, D.G., Damiano, R.F., and Abraham, J.A., "Portable Parallel Logic and Fault Simulation," *Proc. Intl. Conf. CAD*, pp. 506-509, November 1989.

[20] Narayanan, V. and Pitchumani, V., "A Parallel Algorithm for Fault Simulation on the Connection Machine," *Proc. 1988 Intl. Test Conf.*, pp. 89-93, September 1988.

[21] Niermann, T.M., Cheng, W.-T., and Patel, J.H., "PROOFS : A Fast, Memory Efficient Sequential Circuit Fault Simulation,"

Proc. 27th Des. Autom. Conf., pp. 535-540, June 1990.

[22] Nishida, T., Miyamoto, S., Kozawa, T., and Sato, K., "RFSIM : Reduced Fault Simulator," *Proc. IEEE Intl. Conf. CAD*, pp. 198-201, November 1985.

[23] Ozguner, F. and Daoud, R., "Vectorized Fault Simulation on the Gray X-MP Supercompter," *Proc. Intl. Conf. CAD*, pp. 198-201, November 1988.

[24] Roth, J.P., Bouricius, W.G., and Schneider, P.R., "Programmed Algorithms to Compute Tests to Detect and Distinguish Between Failures in Logic Circuits," *IEEE Trans. Electron. Comput.*, vol. EC-16, pp. 567-580, 1967.

[25] Seshu, S., "On An Improved Diagnosis Program," *IEEE Trans. Electron. Comput.*, vol. EC-14, pp. 76-79, 1965.

[26] Ulrich, E.G. and Baker, T., "Concurrent Simulation of Nearly Identical Digital Networks," *Computer*, vol. 7, pp. 39-44, April 1974.

[27] Waicukauski, J.A., Eichelberger, E.B., Forlenza, D.O., Lindbloom, E., and McCarthy, T., "A Statistical Calculation of Fault Detection Probabilities by Fast Fault Simulation," *Proc. 1985 Intl. Test Conf.*, pp. 779-784, November 1985.

[28] Wang, X., Hill, F.J., and Mi, Z., "A Sequential Circuit Fault Simulation by Surrogate Fault Simulation," *Proc. 1989 Intl. Test Conf.*, pp. 9-18, 1989.

저 자 소 개

宋 五 永

1980年 2月 서울대학교 전기공학과 공학사
1982年 2月 한국과학기술원 전기 및 전자공학과 공학석사
1992年 2月 University of Massachusetts at Amherst 전기 및 컴퓨터공학과 공학박사

1982年 3月～1985年 5月 국방부 기술연구 사무관
1991年 9月～1992年 10月 Intergraph Corp. Electronics 수석연구원
1992年 11月～1993年 11月 IBM Corp. Microelectronics 수석연구원
1994年 1月～1994年 8月 삼성전자 LSI사업부 수석연구원
1994年 9月～현재 중앙대학교 제어계측공학과 조교수

주관심분야 : VLSI 시스템 설계, CAD, 테스트