KIPS Tr. Comp. and Comm. Sys.
Vol.3, No.3 pp.73~80 pISSN: 2287-5891

http://dx.doi.org/10.3745/KTCCS.2014.3.3.73

A Practical Approach to Incremental Event-driven HDL Simulation

Seiyang Yang† · Kyuho Shim††

ABSTRACT

In this paper, we propose an incremental simulation method in event-driven HDL simulation to reduce the simulation execution time. In general, the simulation is repeated with a series of design changes. Incremental simulation is an efficient simulation method that shortens the simulation execution time for the following simulation by using the result of previous simulation. We have observed the effectiveness of the proposed approach through the experimentation with multiple real designs.

Keywords : Verification, Simulation, Incremental Simulation

인크리멘탈 이벤트-구동 HDL 시뮬레이션에의 실제적 접근법

양 세 양† · 심 규 호††

요 약

본 논문에서는 이벤트구동 HDL 시뮬레이션에서 시뮬레이션 실행 시간 단축을 위한 인크리멘탈 시뮬레이션 방법을 제시한다. 일반적으로 시뮬레이션 과정은 일련의 반복적인 설계수정들과 동반되어 반복적으로 일어난다. 인크리멘탈 시뮬레이션은 이와같은 반복적인 시뮬레이션에서 설계수정 전의 시뮬레이션 결과를 이용하여서 설계수정 후에 진행되는 시뮬레이션의 수행 시간을 단축할 수 있는 효과적인 시뮬레이션 방법이다. 본 논문에서 제안된 인크리멘탈 시뮬레이션 방법의 유용함은 다양한 실제 디자인들에 적용한 실험을 통하여 확인할 수 있었다.

키워드 : 검증, 시뮬레이션, 인크리멘탈 시뮬레이션

1. 서 론

디지털 반도체 설계의 규모가 커지고 해당 설계 복잡도가 증가함에 따라서, 최근에는 설계 시간의 최대 70%까지를 검증을 위하여 사용되는 것으로 알려져 있다.[1,2] 현재 검증에는 매우 다양한 기술들이 함께 사용되고 있지만, 제일 많이 사용되는 검증 방법은 이벤트-구동(event-driven) HDL(Hardware Description Language) 시뮬레이터를 활용하는 이벤트-구동 논리 시뮬레이션이다. 이의 이유로는 소프트웨어 방식의 높은 유연성, 높은 가시도(visibility)와 제어도(controllability)를 통한 디버깅의 용이성, 사용의 편리성, 높은 경제성 등과 같은 시뮬레이션 특유의 여러가지 장점들이 있기 때문이다. 그러나, 최근의 SOC(System On Chip) 설계와 같은 설계 복잡도가 매우 높은 설계 대상은 시뮬레이션 실행 대상이 되는 객체의 총 수가 매우 커서(예로 로직게이트 수가 1억 게이트 이상), 이들을 순차적으로 처리하여야만 하는 시뮬레이션 특성으로 인하여 시뮬레이션의 속도 저하가 큰 단점으로 대두되고 있다. 이를 보완하기 위하여 별도의 하드웨어 검증플랫폼을 활용하는 에뮬레이션 또는 FPGA(Field Programmable Gate Array)를 활용하는 프로토타이핑을 통한 하드웨어기반 방식의 검증이 활용되고는 있지만, 합성가능하지 않은 설계객체(Verilog에서는 module, VHDL에서는 entity)들에 대한 처리 불가능, 유연성 결여, 디버깅의 어려움, 높은 가격 등의 문제들로 인하여 시뮬레이션과 같은 정도로 보편적으로 사용되고 있지는 못한 것이 현실이다. 따라서, 이벤트-구동 HDL 시뮬레이터를 활용하는 시뮬레이션의 성능 향상의 요구가 꾸준히 지속적으로 제기되고 있다. 이에 대하여 EDA(Electronic Design Automation) 벤더들은 자사의 이벤트-구동 시뮬레이터에 적용한 알고리즘 개선과 구현 상에서의 최적화, 또한 시뮬레이터가 돌아가는 워크스테이션의 성능 향상을 통하여 지속적으로 시뮬레이션의 성능을 향상 시켜오고 있다.

본 논문에서는 이와 같은 이벤트-구동 시뮬레이션의 성능 향상을 위한 일반적인 시도들과는 전혀 다른 방식으로서, 인크리멘탈 이벤트-구동 시뮬레이션에 의한 시뮬레이션 실

※ 이 논문은 부산대학교 자유과제 학술연구비(2년)에 의하여 연구되었음.
† 정 회 원 : 부산대학교 정보컴퓨터공학부 교수
†† 정 회 원 : 삼성전자 SYS.LSI 기반설계센터 책임연구원
논문접수 : 2013년 11월 19일
수 정 일 : 1차 2014년 1월 6일, 2차 2014년 1월 24일, 3차 2014년 2월 18일
심사완료 : 2014년 2월 19일
* Corresponding Author : Seiyang Yang(syyang@pusan.ac.kr)

행 시간 단축을 통하여 궁극적으로 시뮬레이션의 성능을 향상시키는 것과 동일한 효과를 얻는 새로운 시뮬레이션 방법을 제안하고, 이의 실질적 효과를 실제 설계들에 대하여 적용하여 확인하고자 한다. 본 논문의 앞으로의 구성은 다음과 같다. 2장 본론에서는 일반적인 시뮬레이션 진행과정과 기존 연구에서 제안된 방법들을 설명하고, 본 논문에서 제안되는 이벤트-구동 인크리멘탈 시뮬레이션 방법을 구체적으로 설명하고, 3장에서는 실험 결과에 대하여 논의하고, 마지막으로 결론을 언급한다.

2. 시뮬레이션 기반의 검증 과정 및 기존 연구

2.1 일반적인 시뮬레이션 기반의 검증 과정

시뮬레이션을 수행하기 위해서는 HDL 언어로 구술된 2개의 성격이 다른 설계객체가 필요한데, 하나는 설계검증대상(DUT: Design Under Test)이고 다른 하나는 테스트벤치(TB: Test Bench)이다. DUT는 설계 후에 최종적으로 반도체 칩으로 제조되는 대상이며, TB는 기본적으로 시뮬레이션 실행 과정에서 DUT 입력단에 시뮬레이션을 위한 스티뮬러스입력값들을 인가하고 DUT 출력단에서 발생하는 출력값들이 예상출력값들과 일치하는지를 모니터링하여 시뮬레이션이 의도한대로 동작하고 있는지를 확인하는 역할을 수행하게 된다. 시뮬레이션을 통한 검증 과정에서는 하나의 DUT에 대하여 적게는 수십개에서 많게는 수백개 이상의 TB들을 활용하여서 수많은 횟수의 시뮬레이션 실행을 하게 된다. 개개의 TB는 DUT의 특정한 함수적 기능 혹은 올바른 타이밍을 검증하는 역할을 수행한다. 그림 1은 이와 같은 특정한 하나의 TB를 DUT와 결합하여서 디버깅 과정이 포함된 일반적인 시뮬레이션 기반의 검증 흐름도를 보여주고 있다. [3]

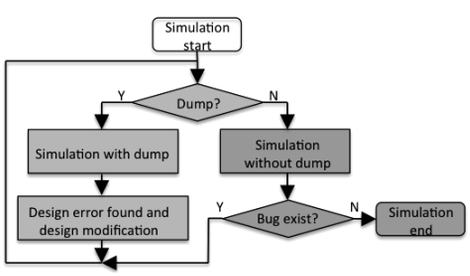


Fig. 1. A typical flow of simulation-based verification

그림 1에서와 같이 특정한 설계오류를 발견하기 위하여서 1회 이상의 시뮬레이션 실행이 진행되고(이를 본 논문에서는 “설계오류 발견을 위한 시뮬레이션”이라 칭함), 설계오류를 발견한 후에는 이를 수정하고 난 후에도 1회 이상의 시뮬레이션 실행(이를 본 논문에서는 “설계오류 수정후 시뮬레이션”이라 칭함)하게 되며, 이와 같은 설계오류 발견을 위한 시뮬레이션과 설계오류 수정후 시뮬레이션이 시뮬레이션 기반의 검증 과정 전체에 걸쳐서 반복적으로 진행하여 진다.

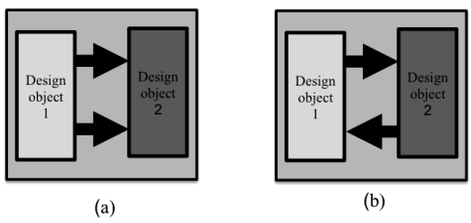


Fig. 2. An example of interconnection among design objects in designs

인크리멘탈 시뮬레이션이란 기본적으로 설계오류 발견을 위한 시뮬레이션 결과를 활용하여서 설계오류 수정후 시뮬레이션을 시뮬레이션 대상이 되는 DUT와 TB 전체를 대상으로하는 시뮬레이션 실행을 시뮬레이션 시간 전체에서 수행하는 대신에, 특정 시뮬레이션 시간대에서는 DUT와 TB의 일부분의 설계객체에 대해서만 시뮬레이션을 실행함으로서 전체 시뮬레이션 실행시간을 단축시키는 시뮬레이션 방법이다. 지극히 단순한 예를 들어보면 그림 2와 같다. 그림 2(a)에서는 설계객체 1과 설계객체 2가 한방향으로만 연결되어 설계객체 2에서의 변화가 설계객체 1에는 영향을 줄 수 없는 경우인 반면에, 그림 2(b)에서는 설계객체 1과 설계객체 2가 양방향으로 연결되어 한 설계객체에서의 변화가 다른 설계객체에 영향을 줄 수 있는 상황이다. 그림 2(a)와 같은 설계에서 설계객체 2에서 설계수정이 이루어진 경우에 설계오류 발견을 위한 시뮬레이션 실행 과정에서 설계객체 2의 입력들을 시뮬레이션 전체 시간에서 덤프하여서 저장하였다고 한다면, 설계오류 수정후 시뮬레이션에서는 이 저장된 설계객체 2의 입력을 입력 스티뮬러스로 활용하여서 설계객체 2에 대해서만 시뮬레이션을 수행하면 되고, 이것이 인크리멘탈 방식의 가장 단순한 예이다.

2.2 기존 연구

로직 시뮬레이션에 대한 인크리멘탈 시뮬레이션은 스탠포드대에서 1980년 중반에 제안되었는데, 여기에는 시간적 방법과 공간적 방법의 두가지 방법이 있다.[4,5] 제안되어진 공간적 인크리멘탈 시뮬레이션 방법[4]은, DUT와 TB로 이루어진 디자인에 여러 설계객체들이 공간 상에서 계층구조로 연결되어져 있다고 생각하고, 설계수정이 있는 국지적 설계객체들 부분에 한해서만 부분적으로 시뮬레이션을 진행하는 것이다. 이를 위하여 공간적 인크리멘탈 시뮬레이션 알고리즘은 궁극적으로 그림 3과 같이 설계수정이 영향을 미치는 부분(빗금친 부분)과 영향을 받지 않는 부분(흰 부분)을 구분하고, 설계수정이 영향을 미치는 부분에 대해서만 재 시뮬레이션을 수행하는 것이다. 이를 위하여 공간적 인크리멘탈 시뮬레이션 방법은 시뮬레이션 시간 전체에 대한 특정 설계객체의 값변화에 대한 기록인 이벤트 로그를 유지하는데, 시뮬레이션 중인 모든 설계객체들의 이벤트 로그 모음을 상태표(state table)라고 하며, 공간적 인크리멘탈 시뮬레이션은 이와 같은 상태표를 생성하고 시뮬레이션 실행 중에 관리하여야만 한다.

이와 같은 공간적 인크리멘탈 시뮬레이션 방법은 조합회로나 피드백이 존재하지 않는 소규모의 순차회로(그림 2(a)와 같은 상황)에는 효과적일 수 있지만, 피드백이 존재하는 순차회로나 버스 등으로 서로에게 영향을 미치게 되는 디자인들(그림 2(b)와 같은 상황)에 대해서는 설계수정이 영향을 미치는 부분(그림 3에서 빗금친 부분)이 디자인의 대부분으로 확대됨으로서 공간적 인크리멘탈 시뮬레이션 방법을 통한 이득을 기대하기 어려운 문제점이 존재한다. 최근의 SOC급 설계들은 서브모듈들간의 연결 구조가 대부분 그림 2(b)와 같이 서브모듈들간에 입력과 출력을 주고 받는 구조이며, 이와 같은 경우에서는 공간적 인크리멘탈 시뮬레이션 방법을 적용하기가 매우 곤란하다. 이와 같은 문제점을 완화시키고자 제안된 시간적 인크리멘탈 시뮬레이션 방법[5]은 설계수정에 의한 영향을 시뮬레이션 시간 상에서 구분하여 변화된 시뮬레이션 시간 구간에 대해서만 재 시뮬레이션하는 것이다. 재 시뮬레이션이 필요한 설계객체는 활성(active) 상태로 구분하고, 그렇지 않은 것을 비활성(inactive) 상태로 구분한다. 설계객체가 비활성 상태에서 활성 상태로 바뀌는 것은 설계객체의 입력 또는 내부 상태가 이전 시뮬레이션에서와 달라지는 경우이다. 반대로, 활성 상태에서 비활성 상태로 바뀌는 것은 모든 입력과 내부 상태가 이전 시뮬레이션에서와 같아지는 경우이다. 시간적 인크리멘탈 시뮬레이션 방법은 내부 상태를 이용하기 때문에 공간적 인크리멘탈 시뮬레이션 방법과는 달리 내부 상태도 상태표에 추가적으로 저장되어야 한다. 그런데, 저장하고 유지 보수해야 할 상태표의 크기는 디자인의 크기와 시뮬레이션 시간에 비례하여 급격히 커지게 된다. 참고문헌 [5]에서는 이의 문제점을 완화시키고자 선택적 저장법을 제안하였다. 그러나 상기 논문이 쓰여진 1980년대 당시에는 디자인의 크기가 현재와는 비교할 수 없을 정도로 작은 수천 개의 노드를 가진 수준이었고 그에 따른 상태표의 크기 또한 메가바이트 단위 수준의 파일로 저장할 수 있을 정도였지만, 현재의 SOC 디자인들은 이미 수억 게이트급의 규모가 일반적이며 이에 따른 상태표의 크기는 기가바이트 단위 수준의 저장 공간을 쉽게 요구하게 된다. 결국 이러한 큰 규모의 상태표를 만들고 시뮬레이션 전체 과정에서 유지하는 것에 걸리는 추가적인 시간이 인크리멘탈 시뮬레이션으로 절약할 수 있는 시뮬레이션 시간을 쉽게 초과할 수 있다. 뿐만아니라, 이와 같은 규모의 자료구조는 주기억장치에 상시적으로 올려 놓을 수도 없고 보조기억장치를 활용한 파일 구조 또는 데이터 베이스를 이용하여야만 한다. 그러나, 이와 같은 보조기억장치를 활용하면서 시뮬레이션 실행 과정 중에서 빈번한 상태표에 대한 접근 및 수정은 더 큰 성능 저하를 유발시키게 된다. 뿐만 아니라, 기존의 인크리멘탈 시뮬레이션 방법들은 현재 보편성있게 사용되고 있는 유일한 HDL 시뮬레이터인 이벤트-구동 시뮬레이터에 적용하기 위해서는 시뮬레이터의 성능을 고려하지 않더라도 상태표를 지원하는 완전히 새로운 구현이 필요함으로서, 현재 사용중인 이벤트-구동 HDL 시뮬레이터를 활용하여 기존의 인크리멘탈 시뮬레이션 방법을 적용하는 것이 불가능하다는 또 다른 문제점이 존재한다.

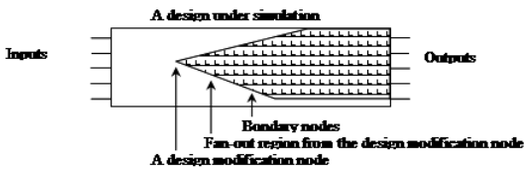


Fig. 3. The concept of spatial incremental simulation

3. 새로운 이벤트-구동 인크리멘탈 시뮬레이션

3.1 사전에 고려하여야할 사항들

설계오류 수정후 시뮬레이션을 이벤트-구동 인크리멘탈 시뮬레이션 방식으로 진행하기 위해서는 다음과 같은 사항들이 설계오류 발견을 위한 시뮬레이션 실행 과정에서 고려되어 준비되어야 한다. 즉, 설계오류 발견을 위한 시뮬레이션 실행 도중에 주기적으로 DUT에 대한 디자인 체크포인트(디자인 체크포인트란 설계객체에 존재하는 플립플롭, 래치, 메모리와 같은 모든 기억소자의 값을 저장하는 것)을 진행하여 상태복원(상태복원이란 추후 시뮬레이션을 임의의 체크포인트에서부터 재출발(restart)할 수 있도록 설계객체에 존재하는 플립플롭, 래지, 메모리와 같은 모든 기억소자에 이전 체크포인트에서 저장된 값을 복원해주는 동작) 후 재출발이 가능하도록 한다. 참고문헌 [3,6,7,8,9]에서 제안된 것과 같이 시뮬레이션 실행 과정 중에 주기적으로 디자인 체크포인트를 진행하여, 추후에 임의의 체크포인트에서 상태복원(state restoration) 후에 재출발이 가능하도록 하는 것이 설계오류의 근원을 찾고 수정하는 디버깅을 효율적으로 수행하기 위하여 설계오류 발견을 위한 시뮬레이션에서는 매우 바람직하다. 뿐만 아니라, 설계오류 수정후 시뮬레이션을 이벤트-구동 인크리멘탈 시뮬레이션 방식으로 진행하는 것이 가능하도록 하기 위해서는 DUT의 입력과 출력, 그리고 설계오류가 존재할 가능성이 있는 DUT 내의 1 이상의 설계객체들의 입력과 출력에 대한 덤프를 시뮬레이션 시간 전체에 걸쳐서 진행하도록 하는 것도 필요하다. 만일, 설계오류 발견을 위한 시뮬레이션 실행에 앞서서 설계오류가 존재할 가능성이 있는 DUT 내의 설계객체를 예상할 수 없는 경우에는 계층구조적으로 DUT 아래에 존재하는 주요 설계객체들(예로 SOC 디자인의 경우에는 DUT에 존재하는 모든 IP들)의 입력과 출력에 대한 덤프를 시뮬레이션 시간 전체에 걸쳐서 진행하도록 한다.

```
INC_sim_main()
{ proc_step1();
proc_step2();
proc_step3();}

proc_step1()
{ Partial_sim(MD);} // Td까지 설계수정된 설계객체 MD만을 대상으로하는 부분 시뮬레이션 //
Partial_sim(MD)
{Run checkpoint simulation for MD with the input stimulus saved before design modification;
```

```
If the actual output of MD differs from the expected output saved before design modification,
record the current simulation time Td and go to proc_step2();}

proc_step2()
{ Testbench_forwarding(TB, Tr); // Tr 까지 테스트벤치 포워딩 //
State_restoration(DUT-MD, MD, Tr);} // Tr 에서 상태복원 //

Testbench_forwarding(TB, Tr)
{ Run simulation for TB with the output stimulus of DUT saved before design modification until Tr;}

State_restoration(DUT- MD, MD, Tr)
{Perform state_restoration for DUT-MD with a checkpoint made prior to design modification, and MD
with a checkpoint made in proc_step1 at Tr;}

proc_step3()
{ Normal_sim(DUT, TB);} // Tr 에서부터 DUT+TB 전체를 대상으로 하는 통상적 시뮬레이션 //
```

Fig. 4. The high-level pseudo code for the proposed incremental simulation

디자인 체크포인트와 설계객체들의 입력과 출력에 대한 덤프는 일순 생각하면 설계오류 발견을 위한 시뮬레이션 단계에서 추가적인 오버헤드로 생각할 수 있는데(평균적으로 DUT에 대한 디자인 체크포인트와 계층구조 상으로 DUT의 바로 밑에 존재하는 모든 설계객체들의 입력과 출력에 대한 덤프는 전체 시뮬레이션 시간을 약 10-20% 정도 증가시킴), 사실은 오류 발견을 위한 시뮬레이션의 목적인 설계오류 발견을 생각한다면 그렇치 않음을 알 수 있다. 즉, 설계오류를 발견하기 위해서는 필수적으로 덤프를 통하여 DUT에 존재하는 시그널들에 대한 가시도를 확보한 후에 파형분석기를 통한 분석이 반드시 필요하다. [3,6,7,8,9] 이와 같은 DUT에 존재하는 시그널들에 대한 가시도 확보를 위해서는 오류 발견을 위한 시뮬레이션 단계에서 요구되는 시뮬레이션 실행 횟수가 실제로는 2회 이상이며, 이 가운데 1회 이상이 DUT에 존재하는 시그널에 대한 덤프를 동반하는 시뮬레이션이 필요하다. 체크포인트가 존재하여 재실행이 가능하다면 덤프를 동반하는 시뮬레이션의 실행을 시뮬레이션 시간 0에서부터 진행하지 않고 해당 체크포인트에서부터 다시 시작하면 됨으로 덤프를 동반하는 시뮬레이션을 훨씬 빠르게 수행(이것은 시뮬레이션 시간단축이라 생각할 수 있다)할 수 있을 뿐만 아니라, DUT 하위계층에 존재하는 특정 설계객체들의 입력과 출력에 대한 덤프가 사전에 시뮬레이션 전체 시간 구간에 걸쳐서 진행되었다고 한다면 덤프를 동반하는 시뮬레이션의 실행은 설계오류의 원인을 내포하고 있는 특정 설계객체만을 대상으로 덤프를 통하여 저장된 특정 설계객체의 입력을 입력 스티뮬러스로 사용하여 훨씬 빠르게 수행(이것은 시뮬레이션 공간단축이라 생각할 수 있다)할 수 있다. 따라서, 사전에 체크포인트 생성과 설계객체들의 입력과 출력에 대한 덤프를 동시에 진행한 경우에 덤프를 동반하는 시뮬레이션은 시뮬레이션 시간단축과 시뮬레이션 공간단축이 동시에 가능하게 됨으로서, 시뮬레이션 시간 0에서부터 시작하여 시뮬레이션 시간 전체에 걸쳐서 DUT와 TB 전체 설계객체를 대상으로 하여 덤프를 동반하는 시뮬레이션보다 훨씬 빠르게 진행할 수 있다. 요약한다면, 설계오류 발견을 위한 시뮬레이션 과정에서 진행되는 체크포인트 생성과 설계객체들의 입력과 출력에 대한 덤프는 설계오류 수정후 시뮬레이션을 인크리멘탈 시뮬레이션으로 수행하기 위하여 필요할 뿐 아니라, 설계오류 발견을 위한 시뮬레이션 과정 자체에서 덤프를 동반하는 시뮬레이션을 빠르게 수행하기 위해서도 필요한 과정임을 알 수 있다.

본 논문에서 제안되는 이벤트-구동 인크리멘탈 시뮬레이션은 세 단계로 구성되어 있는데, 그림 4는 이와 같은 세 단계로 이루어진 이벤트-구동 인크리멘탈 시뮬레이션의 각 단계를 상위수준의 가상코드로 프로시듀어화 한 것이다. 이 세 단계의 시뮬레이션을 통하여 이루어지는 인크리멘탈 시뮬레이션을 통하여 DUT와 TB의 전체 설계객체를 대상으로 시뮬레이션 시간구간 전체에 걸쳐서 진행되는 통상적인 통상적 시뮬레이션과 동일한 결과를 훨씬 신속하게 얻는 것을 기대할 수 있는데, 이들 각 단계별 좀 더 자세한 설명은 다음절에 있다.

3.2 제안된 이벤트-구동 인크리멘탈 시뮬레이션의 세 단계

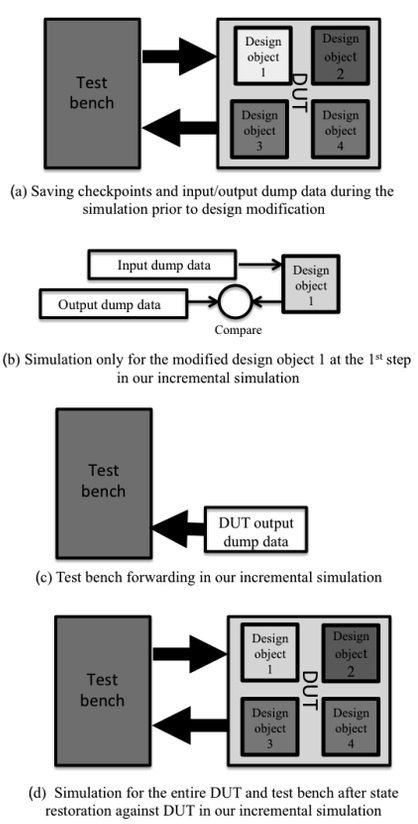
[1 단계]: 설계수정된 설계객체 MD만을 대상으로하는 부분 시뮬레이션

1 단계에서는 DUT와 TB의 전체 설계객체를 대상으로하는 시뮬레이션 대신에 설계수정이 이루어진 설계객체 MD만을 대상으로 시뮬레이션, 즉 부분 시뮬레이션을 시뮬레이션 시간 0에서부터 일정 시뮬레이션 시간 Td(추후에 설명되며, 이 시간 Td는 시뮬레이션 실행 과정에서 자동적으로 정해짐)까지 진행한다. 이 시뮬레이션을 위하여 설계수정이 이루어진 설계객체 MD의 입력 스티뮬러스가 필요하다. 그런데, 이 입력 스티뮬러스로 사용하기 위하여 설계수정 전의 시뮬레이션인 설계오류 발견을 위한 시뮬레이션 실행 과정에서 설계객체 MD의 입력과 출력에 대한 덤프를 이미 진행하였음으로, 이 중에서 저장된 MD의 입력 덤프 데이터를 설계객체 MD의 입력 스티뮬러스로 활용하고, 저장된 MD의 출력에 대한 덤프 데이터는 MD의 예상출력으로 지정하여 동시에 활용한다. 이상과 같이 설계오류 수정후에 적용되는 인크리멘탈 시뮬레이션의 1 단계 시뮬레이션에서 설계오류 발견을 위한 시뮬레이션 실행 과정에서 저장된 입력 스티뮬러스를 활용하여 설계객체 MD만을 대상으로하는 시뮬레이션을 진행하면서, 동시에 주기적으로 설계객체 MD에 대한 디자인 체크포인트(checkpoint)를 생성(단, 체크포인트를 생성하는 주기는 설계수정 전의 시뮬레이션 단계인 설계오류 발견을 위한 시뮬레이션에서 진행된 체크포인트 주기와 동일하여야 함)하여 재출발(restart)이 가능하도록 하면서 시뮬레이션을 진행함과 아울러, 설계객체 MD의 출력 포트에서의 출력 값을 계속하여 MD의 예상출력과 이벤트 단위로 비교하는 것을 설계객체 MD 출력 포트에서의 출력 값이 MD의 예상출력과 달라지는 시점(이 시점이 앞서서 언급된 일정 시뮬레이션 시간 Td임)까지만 진행하고 1 단계 시뮬레이션을 정지한 후, 2 단계로 진행한다.

[2 단계]: 테스트벤치 포워딩 및 상태복원

2 단계에서는 설계수정 전의 시뮬레이션 단계인 설계오류 발견을 위한 시뮬레이션에서 저장된 DUT의 출력을 이용하여서 테스트벤치를 시뮬레이션 시간 0에서부터 1 단계에서 구해진 일정 시뮬레이션 시간 T_d에서 제일 근접한 체크포인트 시점 T_r까지(단 시뮬레이션 시간 T_r은 T_d>T_r을 만족하는 시뮬레이션 시간 T_d보다 과거시간임) 테스트벤치만을 실행(이것을 본 논문에서는 테스트벤치 포워딩이라함)시킨 후에, T_r에서 DUT에 대한 상태복원 후에 재출발을 시행한다. 단 이 재출발을 위한 상태복원은 두 영역에서 구분되어서 이루어지는데, 첫번째 영역은 DUT에서 설계수정이 이루어진 설계객체 M_D를 제외한 설계영역(DUT-M_D)으로 이 영역에 대해서는 설계오류 발견을 위한 시뮬레이션에서 진행된 체크포인트에서 저장된 디자인 상태를 상태복원하는 영역이며, 두번째 영역은 설계수정이 이루어진 설계객체 M_D에 대해서 1 단계인 설계수정된 설계객체만을 대상으로하는 인크리멘탈 시뮬레이션에서 진행된 체크포인트에서 저장된 디자인 상태를 상태복원하는 설계영역(M_D)이다.

2 단계에서 테스트벤치에 대한 포워딩을 우선적으로 진행하는 이유는 다음과 같다. DUT에 입력 스티뮬러스를 인가하고 DUT의 출력을 모니터링하는 것이 주역할인 테스트벤치는 하드웨어를 동작을 합성가능한 형식으로 동시적(concurrent)으로 모델링하는 DUT와는 상이하게 일반적인 C/C++ 프로그램과 같이 순서적(sequential) 동작이 이루어지도록 모델링되는 것이 일반적이다. 이와 같이 모델링된 테스트벤치는 디자인 체크포인트를 수행하고, 이를 나중에 상태복원하더라도 재출발이 가능하지 않다. 따라서 디자인 체크포인트를 통한 재출발이 가능하지 않은 테스트벤치는 항상 최초 시뮬레이션 시간인 시뮬레이션 시간 0에서부터 수행되어야 하며, 이것이 2 단계에서 우선적으로 테스트벤치 포워딩이 실행되어야 하는 이유이다.



(a) Saving checkpoints and input/output dump data during the simulation prior to design modification

(b) Simulation only for the modified design object 1 at the 1st step in our incremental simulation

(c) Test bench forwarding in our incremental simulation

(d) Simulation for the entire DUT and test bench after state restoration against DUT in our incremental simulation

Fig. 5. The conceptual drawing for the proposed incremental simulation

[3 단계]: DUT와 TB 전체 설계객체를 대상으로하는 통상적 시뮬레이션

3 단계에서는 시뮬레이션 시간 T_r에서부터 DUT와 TB 전체 설계객체를 대상으로 하는 통상적인 시뮬레이션을 시뮬레이션 종료시간까지 진행한다.

그림 5는 인크리멘탈 시뮬레이션의 전체 과정을 그림으로 도식한 것이다. 그림 5에서 보는 것과 같이 제안되는 이벤트-구동 인크리멘탈 시뮬레이션을 통한 시뮬레이션 시간 단축은 1 단계와 2 단계를 통하여 이루어진다. 1 단계 시뮬레이션과 2 단계 시뮬레이션(그림 5의 (b)와 (c))은 통상적 시뮬레이션이 DUT와 TB 전체 설계객체를 대상으로하는 것과는 달리, 각각 설계수정이 이루어진 설계객체 와 TB만을 대상으로 함으로서 매우 빠르게 시뮬레이션이 가능하다. 반면에 3 단계 시뮬레이션은 통상적 시뮬레이션과 동일하게 DUT와 TB 전체 설계객체를 대상으로함으로서 3 단계(그림 5의 (d))에서는 시뮬레이션 시간 단축을 기대할 수는 없다.

3.3 추가적으로 고려해야할 사항들

본 논문에서 제안되는 이벤트-구동 인크리멘탈 시뮬레이션을 통하여 얻게되는 시뮬레이션 시간 단축은 어떤 변수들에 의하여 결정되는지를 살펴보자. 첫째로는, 상기 시뮬레이션 시간 T_d가 시뮬레이션 종료 시간에 가까우면 가까울수록 전체 시뮬레이션 시간 단축이 더 많이 됨을 알 수 있다. 그러나, 전체 시뮬레이션 시간 단축을 결정하는 것은 이것뿐만이 아니다. 전체 시뮬레이션 시간 단축은 1 단계 시뮬레이션의 속도에 의해서도 연관되어 있는데, 1 단계 시뮬레이션은 설계수정이 이루어진 설계객체 M_D를 어떤 규모로 설정하는가에 따라서도 이 1 단계 시뮬레이션 속도가 결정되기 때문이다. 즉, 설계수정이 이루어진 설계객체 M_D의 설정을 설계수정이 이루어진 지점을 중심으로 계층구조 상 제일 하위 계층의 제일 작은 설계객체로 설정하는 것이 1 단계 시뮬레이션 속도가 제일 빠르고, 계층구조 상 제일 상위 계층의 제일 큰 설계객체인 DUT로 설정하는 것이 1 단계 시뮬레이션 속도가 제일 느리다. 그러나 설계수정이 이루어진 설계객체 M_D의 설정은 1 단계 시뮬레이션 속도만 결정하지 않고, 시뮬레이션 시간 T_d가 시뮬레이션 종료 시간에 근접시키는 것에도 직접적으로 영향을 미친다. 즉 시뮬레이션의 속도와는 반대로 설계객체 M_D의 설정을 계층구조 상 제일 하위 계층의 제일 작은 설계객체로 설정하면 시뮬레이션 시

간 T_d가 시뮬레이션 종료 시간에서부터 멀어지며, 계층구조상 제일 상위 계층의 제일 큰 설계객체인 DUT로 설정하면 시뮬레이션 시간 T_d가 시뮬레이션 종료 시간에 가까워지는 것이 일반적이다. 이것은, 설계수정이 이루어진 지점을 중심으로 계층구조 상 상대적으로 큰 설계객체로 설정하는 것이, 1 단계 시뮬레이션의 속도를 떨어뜨리지만, 설계수정이 이루어진 지점을 설계객체 M_D 안으로 깊숙하게 위치하도록 함으로서 제어도(controllability)와 관측도(observability)가 저하되는 결과를 초래하여 설계객체 M_D의 출력 포트에서의 출력 값이 M_D의 예상출력과 달라지는 시점 T_d를 시뮬레이션 종료시점에 가까워지게 하기 때문이다.

그러면, 다음으로 답하여할 것은 "1 단계 시뮬레이션에서 설계객체 M_D를 되도록이면 크게 설정하는 것이 좋은가 작게 설정하는 것이 좋은가?"이다. 이는 앞서서 설명된 것과 같이 상반된 효과들이 공히 존재하기 때문에 일률적으로 이야기할 수는 없다. 그러나, 2.3.1절에서 언급한 것과 같이 1 단계 시뮬레이션에서 설계객체 M_D에 인가되는 입력 스티뮬러스는 설계오류가 존재하는 시점을 정확히 알지 못하는 설계오류 발견을 위한 시뮬레이션 실행 단계에서 저장되어야 함으로, 설계오류 발견을 위한 시뮬레이션 실행 단계에서 이미 추후에 설계객체 M_D로 선정될 가능성이 있는 설계객체들을 모두 선정하여 2.3.1절에서 이미 설명한 것과 같이 설계오류 발견을 위한 시뮬레이션 실행 단계에서 선정된 설계객체들의 입력과 출력을 덤프하여 두어야 함으로, 설계객체 M_D의 규모를 설계오류 발견을 위한 시뮬레이션 실행 단계에서 입력과 출력 덤프를 진행한 설계객체들과 동일하게 하는 것이 바람직하다.

4. 실험 결과

본 논문에서 제안된 새로운 인크리멘탈 시뮬레이션 방법을 통한 시뮬레이션 시간의 단축 정도를 실험적으로 확인하기 위하여 다수의 디자인들을 대상으로 하는 실험을 진행하였다. 앞서서 이미 언급한대로 본 논문에서 제안된 새로운 인크리멘탈 시뮬레이션 방법은 현재 제일 보편적으로 사용하고 있는 이벤트-구동 HDL 시뮬레이션에 적용할 수 있을 뿐만 아니라, 디자인의 크기에 상관없이 적용할 수 있는 장점이 있다. 실험을 위하여 사용된 이벤트-구동 HDL 시뮬레이터는 Cadence 사의 Verilog 시뮬레이터인 IUS이다.[10] IUS는 이미 다양한 최적화 기술들이 적용되어진 Synopsys 사의 VCS와 더불어 현존하는 제일 빠른 시뮬레이터이다. 본 논문에서 실험을 위하여 인크리멘탈 시뮬레이션을 각각 다음과 같은 순서로 진행되었다.

Ⅰ. 설계수정 전 시뮬레이션 실행에서 디자인 체크포인트 및 서브 디자인블럭의 입력/출력 덤프 수행

설계수정 전에 진행되는 설계오류 발견을 위한 시뮬레이션 실행과정에서는 DUT의 바로 밑 하위 계층으로 존재하는 모든 Verilog 모듈 인스턴스들 각각의 모든 입력과 출력들을 덤프하는 동시에 DUT에 대한 디자인 체크포인트와 DUT 출력에 대한 덤프를 진행하면서 시뮬레이션을 진행한다. DUT의 하위 계층으로 존재하는 모든 Verilog 모듈 인스턴스들을 대상으로 입력과 출력에 대한 덤프를 진행하는 것은, 설계오류가 발견될 수 있는 특정 Verilog 모듈 인스턴스를 이 시점에서 알 수 없기 때문이다.

Ⅱ. 설계오류 발견 후에 설계오류 제거를 위한 설계수정

Ⅲ. 설계수정 후 시뮬레이션을 인크리멘탈 시뮬레이션 방식으로 진행

i) 설계수정된 설계객체만을 대상으로하는 인크리멘탈 시뮬레이션

DUT에 서브블럭으로 존재하는 설계수정이 이루어진 특정 Verilog 모듈 인스턴스 M_D만을 대상으로 하는 1 단계 인크리멘탈 시뮬레이션을 설계수정 전에 진행된 설계오류 발견을 위한 시뮬레이션 실행 과정에서 저장한 M_D의 입력 덤프를 입력 스티뮬러스로 사용하여 M_D의 출력이 설계오류 발견을 위한 시뮬레이션 실행 과정에서 저장한 M_D의 출력 덤프 값과 달라지는 시점 T_d까지 진행한다.

ii) 테스트벤치 포워딩 및 상태복원 후 재출발

테스트벤치만을 대상으로하여 설계수정 전에 진행된 설계오류 발견을 위한 시뮬레이션 실행 과정에서 저장한 DUT의 출력 덤프를 테스트벤치의 입력 스티뮬러스로 사용하여 테스트벤치를 시뮬레이션 시점 0에서부터 상기 시뮬레이션 시간 T_d에서 제일 근접한 체크포인트 시점 T_r까지 진행한 후, DUT에 대한 상태복원을 진행한다.

iii) DUT와 TB 전체 설계객체를 대상으로 하는 인크리멘탈 시뮬레이션

상기 시뮬레이션 시간 T_r에서부터 통상적인 시뮬레이션을 시뮬레이션 종료시까지 진행한다.

표 1에서는 이와 같은 순서에 따라서 총 7개의 디자인들에 대하여 인크리멘탈 시뮬레이션을 진행하여 전체 시뮬레

Table 1. Experimental result

디자인명	일반적 시뮬레이션 방식을 적용한 경우 (단위:초)	본 논문의 인크리멘탈 시뮬레이션을 적용한 경우 (단위:초)				백분율 (%)
		설계수정 전 시뮬레이션	설계수정 후 인크리멘탈 시뮬레이션		전체 실행 시간	
			1 단계	2+3 단계		
SA(G*)	5,745.4	3,388.0	185.5	124.2	3,697.7	64
OpenSPARC(R*)	3,426.2	2,187.7	935.0	135.5	3258.2	95
IND1(G*)	5,920.2	3,212.2	3.9	208.6	3,424.7	58
IND2(R*)	870.4	593.1	234.2	531.3	1,358.6	156
IND3(M*)	2,344.3	1462.5	156.6	366.3	1985.4	85
IND4(G*)	12,644.0	7,379.7	2.5	208.6	7,590.8	60
IND5(G*)	18,854.0	12,915.0	1,037.0	0.0	13,952.0	74

* : R은 레지스터전송수준, G는 게이트수준, M은 레지스터전송수준과 게이트수준이 혼합된 수준임

이션 실행시간을 측정하고, 이것을 일반적인 시뮬레이션방식으로 진행한 경우의 전체 시뮬레이션 실행시간과 하였다. 7개의 디자인들 중, PCI IP를 포함한 시뮬레이션가속기를 위하여 자체 설계된 디자인(SA)과 과거 썬마이크로시스템사에 의하여 오픈 도메인에 공개된 오픈스파크 프로세서코어(OpenSPARC)를 제외한 나머지 5개의 디자인들은 실제 국내 대기업 A사에서 생산되는 반도체에 실제적으로 사용된 SOC급 디자인들(IND1, IND2, IND3, IND4, IND5)로서, 해당 실험은 보안상의 이유로 A사를 직접 방문하여 진행되었다. 이들 각 디자인들의 추상화수준은 표에 표시되어 있다.

실험에서 구체적으로 정량적 비교는 다음과 같이 하였다. 일반적 시뮬레이션 방식의 경우나 본 논문의 인크리멘탈 시뮬레이션 방식으로 진행한 경우 모두에 대하여 특정 설계수정을 기준으로 수정전 설계오류를 찾기 위한 설계수정 전 시뮬레이션 1회와 설계수정 후 시뮬레이션 1회의 총 2회의 시뮬레이션 실행 시간을 비교하였다. 일반적 시뮬레이션 방식의 경우에는 이 2회의 시뮬레이션을 모두 오버헤드가 제일 적은 덤프없는 시뮬레이션으로 진행한 것으로 가정하여 2회의 덤프없는 시뮬레이션 실행 시간을 측정(표에서 2번째 컬럼)하였다. 반면에, 본 논문의 인크리멘탈 시뮬레이션을 적용한 경우에는 설계수정 전 시뮬레이션(표에서 3번째 컬럼)은 DUT의 바로 밑 하위 계층으로 존재하는 모든 Verilog 모듈 인스턴스들 각각의 모든 입력과 출력들을 덤프하는 동시에 DUT에 대한 디자인 체크포인트와 DUT 출력에 대한 덤프를 진행하면서 시뮬레이션을 1회 진행하면서 진행한 시간을 측정한 것이다. 설계수정 후 진행되는 시뮬레이션은 본 논문에서 제안된 인크리멘탈 시뮬레이션 방식으로 진행하였는데, 인크리멘탈 시뮬레이션의 3 단계를 둘로 나누어 1 단계 실행시에 걸리는 시간(표에서 4번째 컬럼)과 2 단계와 3 단계 실행시에 걸리는 시간(표에서 5번째 컬럼)을 나타내었고, 마지막으로는 이 모든 시간을 더한 전체 실행시간(표에서 6번째 컬럼으로, 3번째와 4번째와 5번째 컬럼의 합)을 나타내었다. 마지막으로 7번째 컬럼은 6번째 컬럼을 2번째 컬럼으로 나눈 것을 백분율로 표시한 것으로 인크리멘탈 시뮬레이션을 적용한 경우의 시뮬레이션 시간을 얼마나 단축할 수 있는지를 알 수 있다.

표에서 같이 전체 실험의 총 7개의 사례에서 한개의 사례를 제외한 6 개의 사례에서 시뮬레이션 실행 시간이 단축됨을 확인할 수 있었다. 설계수정 후 시뮬레이션에 대하여 인크리멘탈 시뮬레이션을 적용하여 설계 시간이 증가한 한개의 사례(디자인4)는 레지스터전송수준의 설계이며 시뮬레이션 실행 시간이 다른 사례들에 비하여 매우 짧음을 알 수 있다. 이와 같은 경우에는 인크리멘탈 시뮬레이션을 위하여 유발되는 오버헤드인 설계수정 전의 시뮬레이션에서의 디자인 체크포인트 및 설계객체들의 입력과 출력에 대한 덤프, 저장된 입력 덤프 데이터를 입력 스티뮬러스로 활용하여 진행되는 시뮬레이션 등이 Verilog 시뮬레이터의 표준 API(Application Programming Interface)인 VPI(Verilog Procedural Interface)를 통하여 이루어지게 되는데, 이 VPI 오버헤드가 인크리멘탈 시뮬레이션을 통하여 얻게 되는 이득보다 매우 크기 때문일 것으로 판단된다. 반면에, 게이트수준의 설계들의 경우에는 시뮬레이션 과정에서 처리되어야 하는 시뮬레이션 대상 객체들이 레지스터전송수준에 비하여 매우 많음으로 인크리멘탈 시뮬레이션을 통한 시간 단축 효과를 크게 볼 수 있는 경우이다. 특히, 게이트수준의 설계에서도 시뮬레이션 시간이 긴 경우에는 더욱 큰 효과를 볼 수 있음을 확인할 수 있다. 마지막 사례인 디자인 7에서는 흥미로운 사실이 하나 더 있는데 3 단계의 인크리멘탈 시뮬레이션 중에서 1 단계만이 수행되고, 2 단계 및 3 단계는 수행할 필요가 없었다는 것이다. 즉, 이 디자인에서는 설계수정 전에 진행된 설계오류 발견을 위한 시뮬레이션 실행 과정에서 저장한 MD의 입력 덤프를 입력 스티뮬러스로 사용하여 MD의 출력이 설계오류 발견을 위한 시뮬레이션 실행 과정에서 저장한 MD의 출력 덤프 값과 달라지는 상황이 시뮬레이션 종료시점까지 나오지 않았다는 것으로, 이와 같은 경우에서도 인크리멘탈 시뮬레이션에 의한 시간 단축 효과가 매우 큼을 알 수 있다.

Table 2. Three Different Incremental Simulation Methods

공간적 방법[4]	조합회로, 단방향 연결구조에서 효과적이며 인크리멘탈 시뮬레이션을 위한 오버헤드가 제일 작으나 제일 비관적 접근법
시간적 방법[5]	제일 완벽하며 정확한 방법이나 상태표 생성/관리 비용이 제일 큼
본 논문에서 제안된 방법	공간적 방법에 비해서는 낙관적이며, 시간적 방법에 비해서는 비관적 접근법. 시간적 방법에 비하여 덜 완벽하며 정확하지 않으나 상태표 생성/관리 불필요. 인크리멘탈 시뮬레이션을 위한 오버헤드가 크지 않아 대규모 회로에 적용 용이

사실, 기존에 제안된 인크리멘탈 시뮬레이션 방법의 하나인 시간적 인크리멘탈 시뮬레이션 방법은 알고리즘 측면에서는 제일 완벽하며 정확하게 (설계변경 전 시뮬레이션의 결과를 가장 최대한으로 설계변경후 시뮬레이션에 활용하는 측면에서 제일 완벽하며 정확하다는 의미임) 인크리멘탈 시뮬레이션을 수행하는 방법이다. 그러나 시간적 인크리멘탈 시뮬레이션 방법은 이 완벽성 및 정확성을 위하여 상태표에 내부 상태가 반드시 포함되어야만 함으로, 회로가 커지게 되면 상태표도 같이 매우 커지고 이를 유지 및 보수하는 것에 큰 오버헤드가 발생함으로 큰 회로(예로 최근의 수천만 게이트급의 SOC 회로 등)에 대해서 적용하기가 어렵다. 반면에, 기존의 또 다른 인크리멘탈 시뮬레이션 방법인 공간적 인크리멘탈 시뮬레이션 방법은 너무 비관적 관점에서 인크리멘탈 시뮬레이션을 수행하는 방법이 되어서, 버스 등을 통하여 상호 연결성이 존재하는 일반적 회로구조에서는 인크리멘탈 시뮬레이션의 이득을 기대하기 어렵다. 본 논문에서는 이와 같은 기존의 인크리멘탈 시뮬레이션 방법들과 차별화하여서, 기존의 시간적 인크리멘탈 시뮬레이션과 달리 완벽성 및 정확성을 추가하지 않는 대신에 인크리멘탈 시뮬레이션을 위한 오버헤드를 최소화하며 동시에, 기존의 공간적 인크리멘탈 시뮬레이션과 비교해서는 훨씬 낙관적 방식을 통

하여 일반적 회로구조에서 대부분의 경우에 인크리멘탈 시뮬레이션의 이득을 기대할 수 현실적인 인크리멘탈 시뮬레이션 방법을 새롭게 제안하였다. 세가지 인크리멘탈 시뮬레이션 방법들의 특징점들을 정성적으로 요약하면 표 2와 같다.

5. 결 론

이벤트-구동 HDL 시뮬레이션을 통한 검증 과정에서는 다수의 디버깅 과정을 통하여 일련의 설계수정이 검증 완료 시까지 일련의 시뮬레이션 실행과 함께 연속적이며 지속적으로 이루어진다. 따라서 특정한 설계오류 수정을 위한 설계수정 전과 후에는 각각 설계오류 발견을 위한 시뮬레이션과 설계오류 수정 후 시뮬레이션이 진행되며, 이 과정은 검증이 완료되는 시점까지 반복된다. 본 논문에서는 이벤트-구동 HDL 시뮬레이션에서 전체 시뮬레이션 실행시간 단축을 통하여 검증 생산성을 높일 수 있는 새로운 인크리멘탈 이벤트-구동 HDL 시뮬레이션 방법을 제안하였다. 이 새로운 방법의 장점을 요약하면 첫째로는 시뮬레이션 대상이 되는 디자인의 복잡도 및 크기에 크게 제약받지 않고, 둘째로는 기존의 이미 산업체에서 널리 사용중인 이벤트-구동 시뮬레이터들을 변경 없이 그대로 활용하여 적용할 수 있다는 것이다. 현재 메이저 EDA 회사들 중 하나인 S사에서는 본 논문에서 제안된 인크리멘탈 시뮬레이션 방법의 유용성에 주목하여 자사의 Verilog 이벤트－구동 시뮬레이터에 이와 같은 기능을 추가하는 자체 연구 및 구현을 진행 중에 있음을 구두 접촉을 통하여 최근 확인할 수 있었다.

제안된 인크리멘탈 시뮬레이션 방법에서는 설계오류 발견을 위한 시뮬레이션에서 디자인에 존재하는 특정한 시그널들에 대한 덤프를 통하여 수집된 시뮬레이션 결과를 활용하여 설계오류 수정 후 시뮬레이션의 실행시간을 단축시키게 되는데, 이는 1 단계의 설계수정된 설계객체만을 대상으로하는 인크리멘탈 시뮬레이션, 2 단계의 테스트벤치 포워딩 및 상태복원 후 재출발, 3 단계의 DUT와 TB 전체 설계객체를 대상으로 하는 인크리멘탈 시뮬레이션의 세 단계로 구성된다. 제안된 새로운 인크리멘탈 시뮬레이션 방법은 실체 산업체에서 설계된 대규모 설계들을 대상으로한 실험을 통하여 시뮬레이션 실행시간 단축에 효과적임을 확인할 수 있었다.

Reference

[1] P. Rashinkar, P. Paterson, and L. Singh, System-on-a-chip Verification: Methodology and Technique, Kluwer Academic Publishers, Dec., 2000.

[2] B. Wile, J. Goss, and W. Roesner, Comprehensive Functional Verification: The Complete Industry Cycle (Systems on Silicon), Elsevier, June, 2005.

[3] Kyuho Shim et al., “Simulation method based on design checkpoint for efficient debugging”, Journal of KIPS-A Vol.19, No.3, Korean Information Processing Society (KIPS), pp.113-120, 2012.

[4] S. Y. Hwang, T. Blank, and K. Choi, “Incremental functional simulation of digital circuits”, Proc. International Conference of Computer-Aided Design, pp.392-395, Nov., 1987.

[5] S. Y. Hwang, T. Blank, and K. Choi, “Fast Functional Simulation: An Incremental Approach”, IEEE Transaction on CAD, Vol.7, pp.765-774, 1988.

[6] Kyuho Shim, Youngrae Cho, Namdo Kim, Hyuncheol Baik, Kyungkuk Kim, Dusung Kim, Jaebum Kim, Byeongun Min, Kyumyung Choi, Maciej Ciesielski, and Seiyang Yang, “A Fast Two-pass HDL Simulation with On-Demand Dump”, Asia and South Pacific Design Automation Conference, 2008. pp.422-427, 2008.

[7] Namdo Kim, Junhyuk Park, Byeong Min, K.M. Choi, Kyuho Shim and Seiyang Yang, “Smart Debugging Strategy for Billion-Gate SOCs”, User Track, 47th Design Automation Conference, June, 2010.

[8] J. Marantz, “Enhanced Visibility and Performance in Functional Verification by Reconstruction” Proc. 35th Design Automation Conference, pp.164-169, June, 1998.

[9] Yu-Chin Hsu, “Maximizing Full-Chip Simulation Signal Visibility for Efficient Debug”, International Symposium on VLSI Design, Automation and Test, pp.1-5, 2007.

[10] IUS Simulator Usermanual, Cadence Design Systems (http://www.cadence.com)

양 세 양

e-mail : syyang@pusan.ac.kr

1981년 고려대학교 전자공학과(학사)

1985년 고려대학교 컴퓨터공학과(공학석사)

1990년 University of Massachusetts, Amherst 컴퓨터공학과(공학박사)

1992년～현 재 부산대학교 정보컴퓨터공학부 교수

관심분야 : SoC 검증

심 규 호

e-mail : kyuho.shim@samsung.com

2003년 부산대학교 정보컴퓨터공학부(학사)

2011년 부산대학교 대학원 컴퓨터공학과 (공학박사)

2010년～현 재 삼성전자 SYS.LSI 기반 설계센터 책임연구원

관심분야 : SoC 검증