http://dx.doi.org/10.3745/KIPSTA.2012.19A.3.113

효율적 디버깅을 위한 디자인 체크포인트 기반 시뮬레이션 방법

심 규 호[†] · 김 남 도[††] · 박 인 학[†††] · 민 병 언[††††] · 양 세 양[†††††]

요 약

디지털시스템 설계에 대한 HDL 시뮬레이션을 통한 검증 과정에서는 설계에 대한 분석 및 디버깅을 위하여 설계에 존재하는 수많은 신호선들에 대하여 시뮬레이션 실행 중에 시그널 덤핑을 통한 가시도 확보가 필요하게 된다. 그러나 이와 같은 시그널 덤핑은 일반적으로 시뮬레이션의 속도를 크게 떨어뜨리는 문제점을 가지고 있거나, 시뮬레이션의 실행 횟수를 늘리는 문제점을 초래한다. 본 논문에서는 디자인 체크포인트를 활용하여서 시그널 덤핑을 효율적이며 신속하게 수행하는 시뮬레이션 방법을 제시하고, 이를 시스템반도체급의 대규모 회로인 산업체 설계들에 적용하여 제안된 방법이 효과적임을 확인하였다.

키워드 : 검증, 시뮬레이션, 디버깅

Simulation Method based on Design Checkpoint for Efficient Debugging

Kyuho Shim[†] · Namdo Kim[††] · Inhag Park[†††] · Byeongeon Min[††††] · Seiyang Yang[†††††]

ABSTRACT

The visibility for signals in designs is required for their analysis and debug during the verification process. It could be achieved through the signal dumping for designs during the execution of HDL simulation. However, such signal dumping, in general, degrades the speed of simulation significantly, or can result in the number of simulation runs. In this paper, we have proposed an efficient and fast simulation method for dumping based on the design checkpoint, and shown its effectiveness by applying it to industrial SOC designs.

Keywords : Verification, Simulation, Debugging

1. 서 론

디지털 반도체 설계의 규모가 커지고 해당 설계 복잡도가 증가함에 따라서, 최근에는 설계 시간의 70%가 검증에 사용되고 있다[1, 2]. 현재 검증에는 여러가지 다양한 기술들이 사용되고 있지만, 제일 많이 사용되는 검증 방법은 이벤트-구동(event-driven) HDL(Hardware Description Language) 시뮬레이터에 의한 이벤트-구동 시뮬레이션이다. 이와 같은 시뮬레이션을 수행하기 위해서는 HDL 언어로 구술된 2개의 성격이 다른 설계객체(Verilog에서는 module, VHDL에서는 entity)가 필요한데, 하나는 설계검증대상(DUT: Design Under Verification)이라 부르고 다른 하나는 테스트벤치(TB: Test Bench)라 한다. DUT는 설계 후에 최종적으로 반도체 칩으로 제조되어지는 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. 이와 같은 시뮬레이션을 통한 검증 과정에서 설계 오류들을 발견하고 수정하는 디버깅을 위해서는, 우선 DUT에 존재하는 수많은 신호선들이 시뮬레이션 실행 과정에서 가지게 되는 논리값 변화(이를 이벤트라 칭함)를 시뮬레이션 실행 과정 중에서 특정한 파형(waveform) 형태로 저장하고, 저장된 파형을 파형분석기(waveform viewer)를 통하여 분석하는 과정을 거치게 된다.

이와 같이 시뮬레이션 실행 과정 중에서 DUT에 존재하는 신호선들의 이벤트를 저장하는 과정을 시그널 덤프(signal dump) 또는 줄여서 덤프라고 한다. 그런데, 이 덤프에서는 다음과 같은 사항들이 고려되어야 한다. 첫째로, 덤프를 통하

※ 이 논문은 부산대학교 자유과제 학술연구비(2년)에 의하여 연구되었음.
† 정 회 원 : 삼성전자 System LSI 사업부 책임연구원
†† 정 회 원 : 삼성전자 System LSI 사업부 수석연구원
††† 정 회 원 : (주)시스템센트로이드 대표이사
†††† 정 회 원 : 삼성전자 System LSI 사업부 마스터
††††† 정 회 원 : 부산대학교 컴퓨터공학과 교수
논문접수 : 2012년 3월 28일
수 정 일 : 1차 2012년 5월 9일
심사완료 : 2012년 5월 12일

여 저장되는 신호선들의 이벤트에 설계 오류의 증상 및 이의 원인까지를 발견하고 수정할 수 있는 유용한 정보가 담겨져 있어야만 한다. 둘째로, 덤프 과정은 시뮬레이션 실행 과정 중에서 발생하는 대용량의 신호선들의 이벤트 데이터를 저장(일반적으로 하드디스크에 저장)하는 과정임으로 저장하고자 하는 이벤트 데이터량에 비례하여 시뮬레이션의 전체 실행 시간도 같이 증가시키게 된다. 셋째로, 설계 오류의 증상 및 원인을 찾아서 이를 수정하는 디버깅이 필요한지 아닌지를 판단하여서 이것에 필요한 덤프를 시뮬레이션 실행 중에 진행하는 것을 결정하는 시점은 시뮬레이션 실행 전이 아니고, 시뮬레이션 실행 후에 이 시뮬레이션의 실행 결과를 보고서야 결정할 수 있다. 이와같은 사항들을 종합하여 보면, "검증 과정에서 설계 오류를 찾고 수정하는 디버깅을 위한 덤프를 시뮬레이션 과정에서 언제 어떤 방식으로 할 것인가?"하는 문제가 검증 과정에서의 시뮬레이션 전체 실행 시간 및 횟수와 밀접한 관계가 있어서, 결과적으로 검증의 생산성에도 많은 영향을 미치게 됨을 알 수 있다.

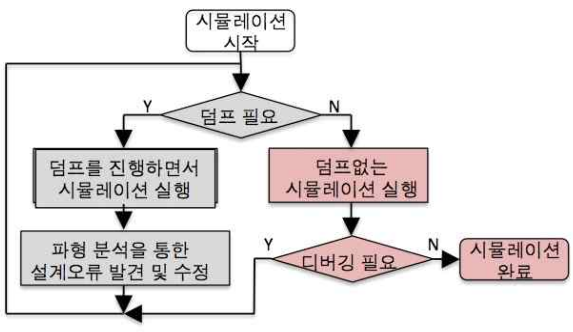
본 논문에서는 시뮬레이션에서 디자인 체크포인트를 활용한 신속하며 효율적인 덤프를 수행함으로서 검증 과정에서의 시뮬레이션 전체 실행 시간과 횟수를 효과적으로 줄일 수 있음으로 인하여 검증의 생산성을 높일 수 있는 새로운 방법을 제안하고, 이의 실질적 효과를 국내 산업체에서 실제 설계된 시스템급의 디지털 회로들에 대하여 적용하여 확인하였다.

2. 본 론

2.1 디버깅 과정을 포함한 일반적인 시뮬레이션 진행과정

시뮬레이션을 통한 검증 과정에서는 하나의 DUT에 대하여 다수의 TB들을 활용하여서 수많은 횟수의 시뮬레이션 실행을 하게 된다. 개개의 TB는 DUT의 특정한 함수적 기능 혹은 올바른 타이밍을 검증하는 역할을 수행한다. (그림 1)은 이와 같은 특정한 하나의 TB를 DUT와 결합하여서 디버깅 과정이 포함된 일반적인 시뮬레이션 실행 흐름도를 보여주고 있다.

그림 오른편 하단에 보이고 있는 시뮬레이션 완료는 특정한 함수적 기능 혹은 타이밍에 대한 검증이 완료된 경우에



(그림 1) 일반적인 시뮬레이션 흐름도

해당된다. 그림의 왼편 부분들(회색 부분들)은 시뮬레이션 실행 과정에서 디버깅을 위해서 덤프를 수행하는 것과 연관되어 있는데, 여기서는 오른편 부분들과는 달리 루프(이를 본 논문에서는 디버깅 루프라고 칭하기로 함)가 형성되어짐으로서 이 디버깅 루프 상에서 수행되는 일들은 수차례 반복적으로 진행되어질 수 있음을 알 수 있는데, 이로 인하여 전체 시뮬레이션 시간을 크게 늘릴 수있는 요인을 가지고 있다. 따라서 디버깅을 위해서 이 디버깅 루프 상에서 수행되는 일들에 소요되는 시뮬레이션의 전체 실행시간을 줄이는 것이 검증 생산성을 높이는 것에 큰 영향을 미치게 된다. 그러면, 다음에는 이를 이루고자 지금까지는 통상적으로 어떤 방법들이 있어 왔는지, 그리고 이 방법들에서 각각 어떤 문제점들이 있는지를 설명하기로 한다.

첫번째 방법은, 디버깅 루프의 실행 횟수를 최소한으로(즉 디버깅 루프의 실행이 오직 한번만으로 해당 디버깅이 가능하도록) 줄이는 것이다. 디버깅 루프의 실행이 오직 한번만으로 해당 디버깅이 반드시 가능하도록 되기 위해서는 디버깅 루프의 존재하는 단 한번의 시뮬레이션 실행 중에 진행되는 덤프를 통하여 확보된 덤프 데이터에 설계 오류의 증상 및 원인까지를 찾아낼 수 있는 디버깅을 가능하게 하는 충분한 시뮬레이션 실행 정보가 포함되어져 있어야만 한다. 따라서, 해당 덤프는 DUT에 존재하는 모든 신호선들을 대상으로 하여서 실행되는 시뮬레이션 시간 전체에서 덤프를 진행하여야 하는데, 본 논문에서는 이와 같은 덤프 방법은 전체 덤프(full dump) 방법이라고 칭한다. 즉, 전체 덤프 방법을 사용하게 되면 디버깅 루프가 존재하지만 어떠한 경우라도 디버깅 루프가 2회 이상 수행되는 것을 방지할 수 있다. 그러나, 전체 덤프 방법의 문제점은 한번의 시뮬레이션 실행 중에 진행되어져야 하는 덤프의 양이 상상을 초월할 정도로 커지게 됨(시스템급 반도체 설계에서는 최소 수백 기가바이트 이상)으로서 이 한번의 시뮬레이션 실행 시간이 작게는 약 2-3배 많게는 20배 이상 늘어난다는 것이다[3, 4].

두번째 방법은, 한번의 디버깅 루프 실행에 수행되어지는 한번의 시뮬레이션 실행 시간을 가능한 최소한으로 줄이는 것이다. 이를 위해서는 시뮬레이션 실행 중에 함께 진행되는 덤프의 오버헤드를 감소시킬 필요가 있다. 즉, DUT에 존재하는 모든 신호선들 중에서 디버깅에 유용한 특정한 신호선들만을 부분적으로 선정하여서 시뮬레이션 시간 전체에서 덤프를 진행하거나, 또는 모든 신호선들에 대하여 특정 일부분 시뮬레이션 시간대에서만 덤프를 진행하거나, 또는 특정한 신호선들만을 부분적으로 선정하여 이들만을 대상으로 특정 시뮬레이션 시간대에서만 덤프를 진행하는 방법을 통하여 덤프의 오버헤드를 적절하게 줄일 수 있다. 이와 같은 덤프 방법은 본 논문에서는 부분 덤프(partial dump) 방법이라 칭한다. 이 부분 덤프 방법에서 중요한 것은 "부분적 영역을 어떻게 선정하여야 할 것인가?"라는 것이다. 만일 이미 진행된 덤프 데이터에 디버깅을 가능하게 하는 충분한 시뮬레이션 실행 정보가 포함되어 있지 않은 경우에는 또 한번의 디버깅 루프 실행을 통하여서 또 한번의 시뮬레이션 실행을 통하여 추가적인 부분 덤프를 진행하여서 새로운 덤

프 데이터를 확보하여야 한다. 따라서 이 방법의 문제점은 시뮬레이션 실행을 통하여 확보된 부분 덤프 데이터에 디버깅을 가능하게 하는 충분한 시뮬레이션 실행 정보가 포함되어 있지 않은 경우에, 이것이 확보될 때까지 시뮬레이션을 반복적으로 진행하여야 한다는 것이다.

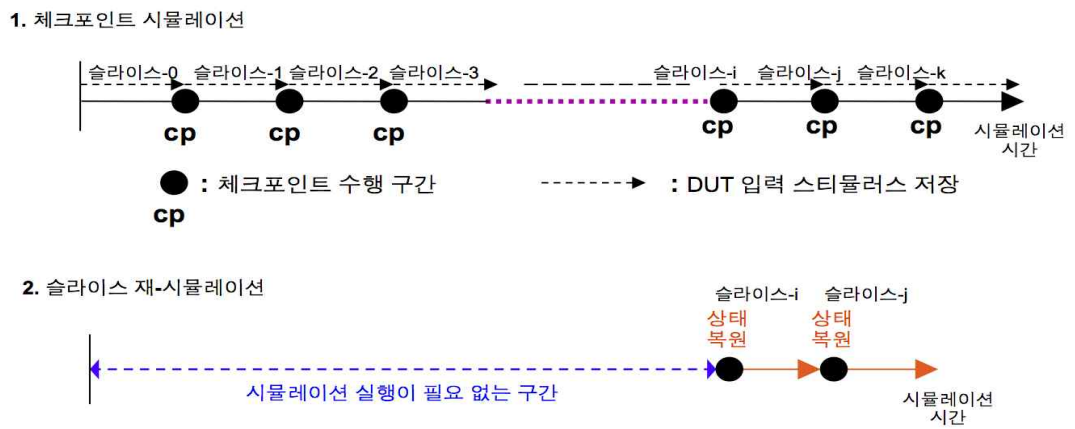
2.2 관련 연구

참고문헌 [5, 6, 7]에서는 앞서 설명한 통상적인 두가지 덤핑 방법들의 문제점을 해결하기 위하여 DUT에 존재하는 특정 신호선들만을 시뮬레이션 실행 중에 덤프하고, 이 이외의 신호선들의 값은 특정 신호선들의 값에서 디버깅 과정에서 실시간으로 계산하는 방법을 제안하였다. 해당 논문들에서 필수 신호선(essential signal)이라 불렸던 이들 특정 신호선들은 DUT의 모든 입력 신호선들과 내부에 존재하는 기억소자(플립플롭, 래치, 메모리) 신호선들을 말한다. 따라서 DUT에 존재하는 모든 신호선들은 필수 신호선과 비 필수 신호선(non-essential signal)로 구분될 수 있는데, 특정 시뮬레이션 시간에서 임의의 비 필수 신호선의 논리값은 DUT의 내부구조 및 함수적 기능을 알고있으면, 이 시간에서의 모든 필수 신호선의 논리값을 알고 있다면 항상 계산을 통하여 구할 수 있다.

따라서 참고문헌 [5, 6, 7]에서는 시뮬레이션 실행 과정에서 DUT에 존재하는 모든 필수 신호선들에 대한 덤핑을 전체 시뮬레이션 시간에 걸쳐서 진행하고, 디버깅이 수행되는 과정에서 비 필수 신호선들에 대한 논리값이 필요한 경우에는 추가적인 시뮬레이션을 진행하지 않고 해당 시뮬레이션 시간에서의 필수 신호선 값들로부터 이를 계산하는 기능을 파형분석기에 내장시켜서 비 필수 신호선 값들을 실시간으로 얻어내는 방법(해당 논문에서는 이 방법을 신호 재구성 방법이라 불렀음)을 연구하였다. 따라서 신호 재구성 방법은 부분 덤프 방법의 특별한 경우로서 앞서서 설명된 부분 덤프 방법의 단점인 디버깅 루프의 시뮬레이션 실행이 상당 횟수로 반복될 수 있다는 점을 효과적으로 제거할 수 있기 때문에 일견 보아서는 매우 효과적인 덤프 방법으로 보여진다. 그러나 신호 재구성 방법은 다음과 같은 문제점을 가지고 있다. 신호 재구성 방법이 전체 덤프 방법과 비교하여서 효과적이기 위해서는 모든 신호선들을 덤프하는 것과 비교하여서 필수 신호선들만을 덤프하는 것이 전체 덤프 방법과 달리 시뮬레이션 시간을 크게 증가시키지 않아야만 한다. 이와 같이 되기위해서는 DUT에 존재하는 모든 신호선들의 총 수에서 비 필수 신호선들의 총 수가 차지하는 비율이 상대적으로 커야만 한다. 게이트수준 디자인들은 일반적으로 이와 같은 조건이 만족되기가 쉽지만, RTL(Register Transfer Level) 디자인들에서는 이와 같은 조건이 만족되기가 쉽지않다. 따라서 DUT에 존재하는 신호선들 중에서 필수 신호선들이 상대적으로 많은 비율을 차지하는 RTL 디자인들에 대한 덤프 방법으로는 신호 재구성 방법은 적합하지 않음을 알 수 있다.

2.3 디자인 체크포인트 기반 시뮬레이션 방법

본 절에서는 본 논문에서 새롭게 제안하는 덤핑 방법을 구체적으로 설명하기로 한다. 이를 위하여 우선 디자인 체크포인트에 대하여 설명한다. HDL 시뮬레이션에서 체크포인트란 시뮬레이션 실행 도중에 시뮬레이션 실행 이미지를 저장하여 추후에 이를 업로드하여 재실행 할 수 있도록 하는 동작이나, 또는 이를 위하여 저장된 시뮬레이션의 실행 이미지(run-time image) 파일을 말한다. 이것의 제일 유용한 사용 사례로는, 매우 긴 시뮬레이션 실행 도중에 예외적 상황 등을 만나서 시뮬레이션의 재 실행이 처음부터 다시 시작되어야 하는 경우에 존재한다. 만일 이전의 시뮬레이션 실행 도중에 체크포인트를 생성하였다 하면 이와 같은 상황에서 시뮬레이션의 재 실행을 처음부터 반복하는 대신에 이 체크포인트에서부터 다시 진행하는 것이 가능함으로서 재실행 시뮬레이션의 시간을 단축할 수 있다. 그러나 이 통상적 체크포인트의 단점으로는 시뮬레이션 실행 이미지 전체를 저장하는 것임으로 시스템급 디자인을 대상으로 하는 경우에는 이미지 크기가 수 기가바이트가 되어 하나의 체크포인트를 생성하거나 이를 후에 업로드할 때에 상당한 시간이 걸리게 된다는 것이다. HDL 시뮬레이션에서는 체크포인트를 위한 또 다른 방법을 생각할 수 있는데, 합성가능한



(그림 2) 본 논문에서 제안된 디자인 체크포인트 기반의 2 단계 시뮬레이션

DUT에 존재하는 모든 순차소자(플립플롭, 래지, 메모리)들의 값(이를 본 논문에서는 상태정보가 칭함)을 저장하는 방법이 그것이다. 본 논문에서는 시뮬레이션의 실행 이미지 전체를 저장하는 통상적인 체크포인트에 대비하여서, 이를 디자인 체크포인트라 부르기로 한다. 디자인 체크포인트는 우선 DUT의 상태정보만을 저장함으로 시뮬레이션의 실행 이미지 전체를 저장하는 체크포인트 보다 크기가 우선 비교할 수 없을 정도로 작고, 따라서 체크포인트 생성 및 업로드를 신속하게 진행할 수 있다.

지금부터는 본 논문에서 제안된 효과적인 새로운 덤핑 방법에 대하여 설명한다. 본 방법은 2 단계의 시뮬레이션 실행을 활용한 덤핑 방법이다. 2 단계의 시뮬레이션 실행 과정에서는 각각 앞서서 설명된 부분 덤프를 진행하게 되는데, 이 두번의 부분 덤프는 각각 목표하는 것이 다르다. 이를 구체적으로 설명하면 다음과 같다. 우선 첫번째 시뮬레이션 실행에서는 시뮬레이션 전체 시간 구간에 걸쳐서 DUT의 모든 입력들에 대한 덤프를 진행함과 동시에, DUT의 디자인 상태정보를 특정 시뮬레이션 시간 소구간(아래에서 시간 소구간에 대해서는 더 자세히 설명됨)에서 주기적으로 저장하여서 다수의 디자인 체크포인트를 생성하는 부분 덤프를 진행한다. 이와 같은 첫번째 시뮬레이션을 본 논문에서는 체크포인트 시뮬레이션(checkpoint simulation)이라고 칭하기로 한다. 이 체크포인트 시뮬레이션에서 수행되는 부분덤프(이를 앞으로는 체크포인트 시뮬레이션 부분덤프라고 칭함)에 대하여 좀 더 자세히 분석할 필요가 있다. (그림 2)에서는 체크포인트 시뮬레이션의 실행 과정과 이 과정을 통하여 진행되는 체크포인트 시뮬레이션 부분덤프를 도식적으로 나타내고 있다. 그림에서 보는 것과 같이, 이와 같은 체크포인트 시뮬레이션 실행을 통하여 n-1개의 디자인 체크포인트가 만들어지면 체크포인트 시뮬레이션이 진행된 전체 시뮬레이션 시간 구간은 n개의 시간 구간으로 나누어지게 되는데, 본 논문에서는 이 n개의 시간 구간을 슬라이스(slice)라고 부르기로 한다. 이 n개의 슬라이스 각각마다 디자인 체크포인트를 통하여 만들어진 시작 구간에서의 상태정보(그림에서 검은원으로 표시됨)와 슬라이스 전체 구간에서 저장된 DUT의 입력값들(그림에서 점선 화살표로 표시됨)을 가지고 있다.

두번째 시뮬레이션을 본 논문에서는 슬라이스 재-시뮬레이션(slice re-simulation)이라고 칭하기로 한다. 두번째 시뮬레이션의 제일 큰 특징은 시뮬레이션 시작을 시뮬레이션 시간 0에서부터 시작하지 않고 n-1개의 디자인 체크포인트 중에 하나를 선정하여 이를 DUT의 초기상태로 활용하고 특정 슬라이스 전체 구간에서 저장된 DUT의 입력값들을 테스트벤치로 활용하여서 특정 슬라이스에 해당하는 시간 구간에 대해서만 시뮬레이션을 진행한다는 것이다. 이와 같은 시뮬레이션 실행 중에서 DUT에 존재하는 모든 신호선들에 대한 덤프를 진행함으로서 해당 시간 구간에 대해서만은 디버깅을 위하여 추가적인 덤프를 필요치 않게 한다.

이와 같은 2 단계에 걸쳐서 진행되는 시뮬레이션에 의한 덤프 방법의 장점은 명확하다. 즉, (그림 2)에서 보이는 것과 같이 본 논문에서 제안되는 2 단계에 걸쳐서 진행되는 시뮬레이션에 의한 덤프 방법에서는 전통적인 부분 덤프 방법과는 달리 어떠한 경우라도 시뮬레이션 전체 시간에 걸쳐서 수행되는 시뮬레이션(첫번째 시뮬레이션)은 오직 1번만 수행하고, 1 이상의 특정 슬라이스들에서만 실행되는 시뮬레이션(두번째 시뮬레이션)은 시뮬레이션 전체 시간과 비교하여서는 매우 짧은 1 이상의 특정 시뮬레이션 시간구간들, 즉 슬라이스들에서만 시뮬레이션을 신속하게 실행하면서 DUT에 존재하는 모든 신호선들에 대하여 덤프를 가능하게 한다. 최근에는 이와 같은 디자인 체크포인트를 이용한 시뮬레이션을 다른 추상화 단계의 시뮬레이션들간에 적용하거나[8], 또는 다른 검증플랫폼들간에 적용하는 시도들[9]도 나타나고 있다.

2.3.1 풀어야 할 문제들

앞 절에서 본 논문에서 제안된 체크포인트 시뮬레이션과 슬라이스 재-시뮬레이션의 2 단계에 걸쳐서 진행되는 시뮬레이션에 의한 새로운 덤프 방법에 대하여 설명하였다. 본 절에서는 이와 같은 새로운 덤프 방법을 이벤트-구동 HDL 시뮬레이션에 적용하는 경우에 풀어야 하는 문제들 및 해결방안들에 대하여 설명한다.

(A) 이벤트-구동 HDL 시뮬레이션에서의 디자인 체크포인트 문제점

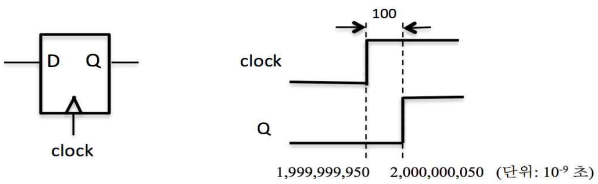
논리 시뮬레이션은 사이클-기반(cycle-based) 시뮬레이션과 이벤트-구동 시뮬레이션으로 나눌 수 있다[1, 2]. 사이클-기반 시뮬레이션은 클럭의 한주기 동안 오직 한번만 평가(evaluation) 과정을 거치는 것에 비하여, 이벤트-구동 시뮬레이션은 디자인에 존재하는 신호선에서 논리값 변화, 즉 이벤트가 생기게 되면 이 이벤트를 이벤트 스케쥴러에 예약을 하고 시뮬레이션 시간 상에서 앞선 순서대로 평가를 하게 된다. 따라서, 사이클-기반 시뮬레이션은 기본적으로 단일클럭을 사용하고 지연시간(delay)를 고려하지 않는 동기식 설계에만 적용할 수 있는 반면에, 이벤트-구동 시뮬레이션은 사이클-기반 시뮬레이션과 같은 제약들 없이 임의의 논리회로에 대하여 적용 가능함으로 최근의 모든 상용 HDL 시뮬레이터들은 모두 이벤트-구동 시뮬레이션 방식을 채용하고 있다.

따라서 본 논문에서 제안된 체크포인트 시뮬레이션과 슬라이스 재-시뮬레이션의 2 단계에 걸쳐서 진행되는 시뮬레이션에 의한 새로운 덤프 방법도 이벤트-구동 시뮬레이션에서 적용할 수 있어야 한다. 그런데 이벤트-구동 시뮬레이션 방식에서는 디자인 체크포인트 생성 시에 다음과 같은 문제점이 존재한다. 즉, 이벤트-구동 시뮬레이션에서는 지연시간을 특별한 제약없이 고려할 수 있음으로 논리회로에 존재하는 다양한 지연시간 모델을 다양한 추상화 수준에서 반영할 수 있다. 이와 같은 지연시간 모델이 반영된 DUT에 대한 디자인 체크포인트를 하고자 하는 경우에 제일 중요한 질문은 “디자인 체크포인트를 통하여 상태정보를 저장하는 특정한 시뮬레이션 시점이 언제여야 하는 것인가?”라는 것이다.

순수한 사이클-기반 시뮬레이션에서는 이 질문에 대한 답은 단순하며 명확한데, 그것은 임의의 시뮬레이션 시점 어디에서나 DUT에 대한 디자인 체크포인트를 수행하게 되면 항상 올바른 상태정보를 저장하게 된다. 이와 같은 이유는 순수한 사이클-기반 시뮬레이션에서는 지연시간을 고려하지 않기 때문이다. 그러나 도 3의 예에서와 같이 플립플롭의 전파 지연시간(propagation delay)인 clock-to-Q 지연시간을 모델링한 이벤트-구동 시뮬레이션 경우를 생각해 보자. 그림에서의 예와 같이 clock-to-Q 지연시간이 100 ps이고 클럭의 엑티브되는 시뮬레이션 시간이 1,999,999,950 ps이면 입력 D의 이벤트에 의하여 출력 Q의 값이 변하는 시뮬레이션 시간은 2,000,000,050 ps이 된다. 만일 디자인 체크포인트를 위한 상태정보 저장을 시뮬레이션 시간 2,000,000,000 ps에서 하면 Q의 이전 값인 논리값 0이 디자인 체크포인트에 포함됨으로 올바르지못한 상태정보(2,000,000,000 ps는 이미 클럭이 엑티브된 후임으로 논리값 1이 올바른 상태정보임)가 저장됨으로 문제(슬라이스 재-시뮬레이션이 올바르게 수행되지 못함)가 된다.

(B) 비동기적 클럭들이 2이상 존재하는 설계에 대한 디자인 체크포인트 문제점

앞서서 언급된 플립플롭 또는 래치의 지연시간이 고려된 이벤트-구동 시뮬레이션에서 디자인 체크포인트를 어느 시점에 수행하여야 올바른 상태정보를 저장할 수 있는지에 대한 문제는 DUT에 비동기적 클럭들이 2 이상 존재하는 설계에서는 더욱 어려워진다. 즉 DUT에 클럭이 오직 1개만 있는 경우에서는 클럭이 엑티브된 시뮬레이션 시점에서부터 플립플롭이나 래치의 출력이 변화되는 시뮬레이션 시점 사이에서만 디자인 체크포인트를 수행하지 않으면 항상 올바른 상태정보를 저장할 수 있다. 그러나, 만일 DUT에 다수의 비동기적 클럭들이 존재하는 경우에는 이 클럭들간의 위상차(phase difference)가 시뮬레이션 시간에 걸쳐서 계속적으로 변하게 됨으로 인하여 이와 같은 올바른 상태정보를 저장하는 시뮬레이션 시점을 찾는 것이 불가능하게 된다. 이와 같은 기술적 어려움때문에 지금까지는 이벤트-구동 시뮬레이션에서는 상태정보 저장을 통한 디자인 체크포인트를 시도한 사례가 존재하지 않았다.



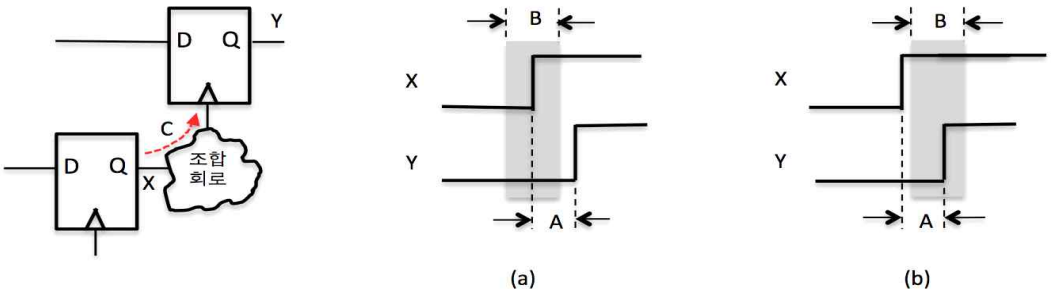
(그림 3) 플립플롭에서 상태정보 저장의 문제점

(C) TB에 대한 디자인 체크포인트 문제점

디자인 체크포인트에서 또 다른 문제는 TB에 대한 체크포인트 여부이다. 시뮬레이션 관점에서 보면, DUT에 입력 스티뮬러스를 인가하고 DUT의 출력을 모니터링하는 것이 주역할인 TB는 하드웨어를 모델링한 DUT와 달리 동시적(concurrent) 동작이 되도록(다른말로 하면 합성가능하도록) 모델링되지 않고, C/C++ 프로그램 코드와 순서적(sequential) 동작이 이루어지도록(다른말로 하면 합성불가능하게) 모델링되는 것이 일반적이다. 따라서 TB에 대한 디자인 체크포인트를 수행하여 상태정보를 저장하고, 이를 나중에 복원하더라도 TB에 대한 올바른 재실행은 가능하지 않다.

(D) 본 논문에서 제안된 디자인 체크포인트 해법

앞서서 언급된 디자인 체크포인트들의 문제점들을 본 논문에서는 다음과 같이 해결하였다. 우선 이벤트-구동 시뮬레이션에서의 디자인 체크포인트 문제 및 다수의 비동기적 클럭들이 존재하는 상황에서의 디자인 체크포인트 문제는 디자인 체크포인트를 특정 시뮬레이션 시점에서 수행하는 대신에 일정한 시뮬레이션 시간구간(구체적으로 얼마의 시뮬레이션 시간구간이어야 하는지는 추후 설명됨)에 수행함으로 올바른 디자인 체크포인트가 가능하다. 위에서의 동일한 예에만 국한하여서 설명한다면, clock-to-Q 지연시간이 100 ps이라고 하면 하나의 체크포인트를 위하여 DUT의 상태정보를 저장하는 시뮬레이션 시간구간을 100ps보다 조금이라도 크게 하면 이 예에서는 항상 올바른 상태정보 저장이 가능하다. 뿐만 아니라, 이와 같이 일정한 시뮬레이션 시간구간에 걸쳐서 디자인 체크포인트를 수행하게 되면 다수의 비동기적 클럭들이 존재하는 경우에서도 항상 올바른 상태정보 저장이 가능하다.



$A = D_{longest\text{-}clock\text{-}Qpath} + D_{clock\text{-}to\text{-}Q}$, $B > D_{longest\text{-}clock\text{-}Qpath} + D_{clock\text{-}to\text{-}Q}$, $C = D_{longest\text{-}clock\text{-}Qpath}$

(a)와 (b)에서 회색부분이 상태정보를 진행하는 시뮬레이션 시간구간

(그림 4) 이벤트-구동 시뮬레이션에서 올바른 상태정보 저장 방법

TB에 대한 디자인 체크포인트 문제의 해결책은 TB에서 DUT의 입력단으로 인가되는 입력 스티뮬러스를 체크포인트 시뮬레이션 실행 과정에서 시뮬레이션 전체 시간구간에 걸쳐서 저장하고, 이와 같이 저장된 것을 슬라이스 재-시뮬레이션 실행에서 입력 스티뮬러스로 사용하는 것이다. 즉, 체크포인트 시뮬레이션 실행 과정에서는 TB를 제외한 DUT에 대해서만 디자인 체크포인트를 진행하고, 슬라이스 재-시뮬레이션 실행 과정에서는 오리지널 테스트벤치를 사용하지 않는 것이다.

그러면 마지막으로 이벤트-구동 시뮬레이션에서의 디자인 체크포인트에서 제일 중요한 문제인 “올바른 디자인 체크포인트를 위하여 DUT의 상태정보를 저장하는 일정한 시뮬레이션 시간구간을 얼마로 하여야 할 것인가?”에 대한 답을 찾아보도록 하자. DUT에 존재하는 모든 플립플롭들 혹은 래치들의 클럭입력들 중에서 이 클럭입력들이 조합회로를 거쳐서 DUT의 입력들에 연결되지 않고 플립플롭 혹은 래치의 출력 Q에 연결된 것들 가운데 이 연결된 경로상의 지연시간이 제일 큰 것을 $D_{longest\text{-}clock\text{-}Qpath}$라고 하고 플립플롭/래치의 clock-to-Q 지연시간 중에서 제일 큰 것을 $D_{clock\text{-}to\text{-}Q}$라고 한다면, 올바른 디자인 체크포인트를 위하여 DUT의 상태정보를 저장하는 일정한 시뮬레이션 시간구간은 항상 ($D_{longest\text{-}clock\text{-}Qpath}$+$D_{clock\text{-}to\text{-}Q}$)보다 조금이라도 크기만 하면 된다. (그림 4)는 이를 가능한 두가지 경우로 나누어서 왜 상태정보 저장을($D_{longest\text{-}clock\text{-}Qpath}$+$D_{clock\text{-}to\text{-}Q}$)보다 큰 시뮬레이션 시간구간에 걸쳐서 하게 되면 항상 올바른 상태정보 저장이 되는지를 보여주고 있다. (그림 4(b))는 플립플롭의 클럭 입력의 엑티브 이벤트가 디자인 체크포인트를 통하여서 저장되는 상태정보에 포함되어짐으로 인하여 이 클럭 입력의 엑티브 이벤트가 발생되는 시점에서부터 clock-to-Q 시간 후에 반드시 플립플롭의 출력에서 이벤트가 발생될 수 있게되는 상황이고, (그림 4(c))는 플립플롭의 출력에서의 이벤트 자체가 디자인 체크포인트를 통하여서 저장되는 상태정보에 직접 포함되어짐으로 인하여 올바른 상태정보가 저장되는 상황인데, 이 두 상황 모두 슬라이스 재-시뮬레이션 실행 초기단계에서 체크포인트 시뮬레이션 실행 과정에서 이와 같이 저장된 상태정보를 복원하게 되면 올바른 값들로 상태정보 복원이 일어나게 됨으로서, 해당 시뮬레이션 시점에서부터 올바른 시뮬레이션을 재현할 수 있게된다.

위와 같은 올바른 디자인 체크포인트 생성을 위한 상태정보 저장 방법은 DUT에 다수의 비동기적 클럭들이 존재하는 경우에도 그대로 적용될 수 있다. 즉 이벤트-구동 시뮬레이션에서 DUT의 상태정보 대상이 되는 플립플롭 또는 래치의 클럭 입력에서의 이벤트가 원인이 되어서 출력에서 이벤트가 결과로서 나타나게 되는데, 이들 두 이벤트 사이의 지연시간 차이가 최대 ($D_{longest\text{-}clock\text{-}Qpath}$+$D_{clock\text{-}to\text{-}Q}$)가 되는 상황은 단일 클럭의 경우나 다수의 비동기적 클럭들이 존재하는 경우 모두에서 동일하기 때문이다. 즉 다수의 비동기적 클럭들이 존재하는 경우에도 상태정보를 저장하는 시뮬레이션 시간구간을($D_{longest\text{-}clock\text{-}Qpath}$+$D_{clock\text{-}to\text{-}Q}$)보다 조금만이라도 크게 하면 항상 (그림 4(a)) 또는 (b)와 같은 상황에서 올바른 상태정보를 저장하는 것이 항상 가능하다.

2.3.2 기존 방법과의 비교

본 절에서는 본 논문에서 제안된 방법과 기존 연구들에서 제안된 방법(신호 재구성 방법)[5, 6, 7]과의 차이점을 살펴보기로 하자. 신호 재구성 방법에서는 첫번째 시뮬레이션 실행 과정에서 DUT의 상태정보를 시뮬레이션 전체 실행시간에 걸쳐서 저장하지만, 본 논문에서 제안된 방법에서는 DUT의 상태정보를 시뮬레이션 전체 실행시간에 걸쳐서 저장하지 않고 오직 체크포인트들에서만 주기적으로 저장함으로 첫번째 시뮬레이션 실행에서 상태정보를 저장하는 것에 의한 오버헤드를 최소화할 수 있다. 특히, 최근의 수천만 게이트의 시스템반도체에 존재하는 플립플롭/래치/메모리의 갯수는 매우 크기 때문에 이와 같은 것들을 모두 포함하여야 하는 상태정보를 시뮬레이션 전체 실행시간에 걸쳐서 저장하여야 하는 기존의 방법들에 비하여, 오직 주기적으로 체크포인트 구간들에서만 저장하도록 하는 본 논문에서의 방법이 훨씬 발전된 방법임을 쉽게 알 수 있다.

3. 실 험

본 논문에서 제시된 방법의 효율성을 실험적으로 보이기 위하여 실제 산업체에서 최근에 설계되어진 7개의 Verilog 디자인들을 대상으로 디자인 체크포인트의 갯수를 50개 설정하여서 실험을 수행하였다. <표 1>은 이들 실험 대상이 된 설계들에 대하여 덤프없는 시뮬레이션(No dump), 전통적인 전체 덤프 시뮬레이션(Full dump), 그리고 본 논문에서 제안된 체크포인트 시뮬레이션(Checkpoint sim.)에 걸리는 각각의 시뮬레이션 실행 시간과 덤프없는 시뮬레이션과 체크포인트 시뮬레이션 실행시간들간의 비율(C/A)과 전체 덤프 시뮬레이션과 체크포인트 시뮬레이션 실행시간들간의 비율(C/B)을 보여준다. 비율 C/A는 덤프없는 시뮬레이션 대비하여서 체크포인트 시뮬레이션을 수행하기 위한 수행시간 측면에서의 오버헤드를 나타내는 수치인데, 이 오버헤드가 19% ~ 80%이다. 비율 C/B는 전체 덤프 시뮬레이션과 비교하여 체크포인트 시뮬레이션을 수행함으로서 시뮬레이션을 얼마만큼 빨리 수행할 수 있는지에 관한 효율성을 나타내는 수치인데, 이것으로부터 체크포인트 시뮬레이션을 수행하기 위해서는 전체 덤프 시뮬레이션의 수행시간의 36% ~ 74% 만을 사용하면 된다는 것을 알 수 있다. 또한 디자인 체크포인트의 갯수를 50개로 한 경우에 한 슬라이스에 대한 슬라이스 재-시뮬레이션(해당 슬라이스 시간구간에서는 전체 덤프를 진행하면서 진행함)은 7개의 모든 디자인들에 대해서 197초 ~ 709초 사이에서 매우 빠르게 수행되었다. 물론 디자인 체크포인트의 갯수를 증가시키면 체크포인트 시뮬레이션의 수행시간은 증가하게 되며, 슬라이스 재-시뮬레이션의 수행시간은 감소하게 될 것이다. 다만, 디자인 체크포인트의 갯수를 2배 증가시키더라도 디자인 체크포인트를 저장하는 시뮬레이션 시

〈표 1〉 실험 결과

디자인명	상태 갯수	시뮬레이션 수행 시간 (단위: 초)			비 율		디스크사용량(Byte)	
		No dump (A)	Full dump (B)	CP sim. (C)	C/A	C/B	Full dump	CP sim.
D1	270M	39,644	68,477	50,481	1.27	0.74	165G	3.82G
D2	30M	12,329	26,317	14,760	1.20	0.56	21G	0.40G
D3	62K	10,952	28,393	16,312	1.49	0.57	46G	0.22G
D4	20K	3,676	10,984	6,626	1.80	0.60	29G	0.20G
D5	61K	25,346	51,395	31,286	1.23	0.61	35G	0.64G
D6	49K	23,728	43,069	28,327	1.19	0.66	40G	0.64G
D7	74K	19,008	68,472	24,381	1.28	0.36	48G	0.66G

간구간은 시뮬레이션 전체시간 구간에 비하여서는 그래도 매우 작은 시간 구간이라는 것과, 디자인 체크포인트의 갯수를 증가시키지 않은 경우나 2배 증가시킨 경우 모두에서 DUT 입력 스티뮬러스의 저장은 동일하게 시뮬레이션 전체시간 구간에서 진행되어야 하기 때문에 디자인 체크포인트의 갯수가 체크포인트 시뮬레이션의 수행시간에 영향을 미치는 정도는 크지 않다. 이상과 같은 실험을 통하여 본 논문에서 제안된 시뮬레이션 방법이 덤프를 요하는 시뮬레이션에 매우 효과적임을 확인할 수 있었다.

또한 본 논문에서 제안된 시뮬레이션 방법이 전체 덤프 시뮬레이션 대비하여 매우 적은 저장공간을 사용함을 알 수 있다. 따라서 본 논문에서의 디자인 체크포인트 기반의 시뮬레이션을 통한 덤프 데이터 확보 방법에서는 적은 공간만을 사용하는 디자인 체크포인트 결과를 우선 보존하고, 실제 덤프 데이터의 확보가 필요한 경우에서만 신속한 슬라이스 재-시뮬레이션 실행 과정을 통하여서 원하는 덤프 결과를 얻는 것이 가능하다.

4. 결 론

본 논문에서는 검증 방법 중 가장 많이 사용되는 이벤트-구동 시뮬레이션에서 검증 생산성을 높이기 위한 디자인 체크포인트에 기반한 효율적인 시뮬레이션 방법을 연구하여 제안하였다. 본 논문에서 제안된 방법은 2회의 시뮬레이션 실행으로 구성될 수 있는데, 항상 수행되어지는 첫번째의 체크포인트 시뮬레이션 실행 중에 DUT에 대하여 주기적으로 디자인 체크포인트들을 덤프없는 시뮬레이션 대비 매우 적은 시뮬레이션 시간증가를 통하여 생성 한 후에, DUT에 대한 디버깅이 필요함으로 인하여 DUT에 존재하는 시그널들에 대한 덤프가 필요한 경우에서만 선택적으로 수행되는 두번째 슬라이스 재-시뮬레이션의 실행에서는 첫번째 체크포인트 시뮬레이션 실행 과정에서 저장되어진 디자인체크포인트를 활용하여서 DUT에 존재하는 모든 신호선들에 대한 덤프를 수행하게 된다. 이와 같은 슬라이스 재-시뮬레이션은 DUT에 존재하는 모든 신호선들에 대한 덤프를 수행하는 시뮬레이션이더라도, 시뮬레이션 시작을 시뮬레이션 시간 0에서부터 시작하지 않고 해당 체크포인트가 이루어진 시뮬레이션 시간에서부터 진행할 뿐만 아니라 슬라이스의 시간구간도 시뮬레이션 전체 시간구간에 비하여 매우 짧다. 따라서 어떠한 경우라도 두번째 슬라이스 재-시뮬레이션은 매우 빠르게 수행하는 것이 가능함을 알 수 있다. 실제 산업체에서 설계된 대규모 디자인들을 대상으로한 실험을 통하여 본 논문에서 제안되는 방법이 매우 효과적임을 확인할 수 있었다.

참 고 문 헌

[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, 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.

[4] 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.

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

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

[7] Visibility Enhancement Technology for Simulation Whitepaper, Novas Software (http://www.springsoft.com), 2009

[8] Dusung Kim, Maciej Ciesielski, Kyuho Shim, and Seiyang Yang, "Temporal parallel simulation: A fast gate-level HDL simulation using higher level models", *Proc. DATE Conference*, March, pp.1584-1589, 2011.

[9] Making Hardware/Software Co-Verification Easier for ARM Cortex™ - A Series Processor-based Designs, On-Demand Web Seminar, Mentor Graphics (http://www.mentor.com), 2012.

심 규 호

e-mail : kyuho.shim@samsung.com
2003년 부산대학교 정보컴퓨터공학부(학사)
2011년 부산대학교 컴퓨터공학과 (Ph.D., 석박사통합)
2010년~현 재 삼성전자 System LSI 사업부 책임연구원
관심분야 : Front-end Design & Verification Methodology

김 남 도

e-mail : nd.kim@samsung.com
1994년 부산대학교 컴퓨터공학과(학사)
1996년 부산대학교 컴퓨터공학과(공학석사)
2002년 부산대학교 컴퓨터공학과(Ph.D.)
2002년~현 재 삼성전자 System LSI 사업부 수석연구원
관심분야 : Front-end Design & Verification Methodology

박 인 학

e-mail : ihpark@systemcentroid.com
1980년 고려대학교 전자공학과(학사)
1983년 고려대학교 전자공학과(석사)
1992년 국립폴리테크연구소(INPG), France (Ph.D.)
1982년~2000년 ETRI, 책임연구원, 실장
2000년~현 재 (주)시스템센트로이드 대표이사
관심분야 : EDA, 상위수준합성, 임베디드시스템 설계 등

민 병 언

e-mail : byeong.min@samsung.com
1982년 고려대학교 전기공학과(학사)
1984년 고려대학교 전기공학과(공학석사)
1985년~현 재 삼성전자 System LSI 사업부 마스터
2001년 Texas A&M University (Ph.D. in Computer Engineering)
관심분야 : Front-end Design & Verification Methodology

양 세 양

e-mail : syyang@pusan.ac.kr
1981년 고려대학교 전자공학과(학사)
1985년 고려대학교 전자전산기공학과 (공학석사)
1990년 University of Massachusetts Amherst(Ph.D.)
1990년~1991년 MCNC R&D staff
1991년~현 재 부산대학교 컴퓨터공학과 교수
관심분야 : HDL 시뮬레이션, 병렬 시뮬레이션, HW 디버깅 등 검증 전분야