KIPS Trans. Comp. and Comm. Sys.
Vol.8, No.3 pp.57~64 pISSN: 2287-5891

https://doi.org/10.3745/KTCCS.2019.8.3.57

Prediction-Based Parallel Gate-Level Timing Simulation Using Spatially Partial Simulation Strategy

Jaehoon Han† · Seiyang Yang††

ABSTRACT

In this paper, an efficient prediction-based parallel simulation method using spatially partial simulation strategy is proposed for improving both the performance of the event-driven gate-level timing simulation and the debugging efficiency. The proposed method quickly generates the prediction data on-the-fly, but still accurately for the input values and output values of parallel event-driven local simulations by applying the strategy to the simulation at the higher abstraction level. For those six designs which had used for the performance evaluation of the proposed strategy, our method had shown about 3.7x improvement over the most general sequential event-driven gate-level timing simulation, 9.7x improvement over the commercial multi-core based parallel event-driven gate-level timing simulation, and 2.7x improvement over the best of previous prediction-based parallel simulation results, on average.

Keywords : Verification, Event-Driven Logic Simulation, Parallel Logic Simulation, Timing Simulation

공간적 부분시뮬레이션 전략이 적용된 예측기반 병렬 게이트수준 타이밍 시뮬레이션

한 재 훈† · 양 세 양††

요 약

본 논문에서는 이벤트구동 게이트수준 타이밍 시뮬레이션의 성능 향상 및 디버깅 효율성 크게 높일 수 있는 공간적 부분시뮬레이션 전략이 적용된 효율적인 예측기반 병렬 시뮬레이션 기법을 제안한다. 제안된 기법은 병렬 이벤트구동 로컬시뮬레이션들의 입력값과 출력값에 대한 빠르면서도 정확한 예측을 달성하기 위해서, 공간적 부분시뮬레이션 전략을 추상화 상위수준 시뮬레이션에 적용하여 정확한 예측 데이터를 빠르고 즉각적으로 생성해낸다. 공간적 부분시뮬레이션 전략이 적용된 예측기반 병렬 게이트수준 타이밍 시뮬레이션은 성능 평가를 위하여 사용된 6개의 벤치마크 설계들에 대하여 제일 일반적인 순차 이벤트구동 게이트수준 타이밍 시뮬레이션에 비하여 평균 약 3.7배, 상용화된 멀티코어 기반의 병렬 이벤트구동 게이트수준 타이밍 시뮬레이션에 비해서는 평균 9.7배, 그리고 기존의 가장 우수한 예측기반 병렬 이벤트구동 게이트수준 타이밍 시뮬레이션 결과에 비해서도 평균 2.7배의 시뮬레이션 성능이 향상됨을 확인할 수 있었다.

키워드 : 검증, 이벤트구동 로직 시뮬레이션, 병렬 로직 시뮬레이션, 타이밍 시뮬레이션

1. 서 론

게이트수준 타이밍 시뮬레이션은 전단계(front-end) 설계에서 제일 마지막으로 진행되는 검증 작업으로 타이밍과 관련된 설계오류를 찾아서 수정하는 것에 제일 효과적인 검증 방법으로서, 최근 들어서 다시 중요성이 커지고 있다[12]. 게이트수준 타이밍 시뮬레이션 대신으로 타이밍 관련된 버그를 찾아내는 방법으로는 정적 타이밍 분석(STA: Static Timing Analysis) 방법을 과거에는 많이 사용하였다. 게이트수준 타이밍 시뮬레이션과 비교하여서 정적 타이밍 분석의 장점은 시뮬레이션 입력값들이 필요하지 않다는 것과 수행시간이 매우 빠르다는 것이다. 그러나, 최근의 대부분의 SOC(System On Chip) 설계들이 초미세 공정을 사용하고 설계 자체에 수많은 비동기 클럭들이 존재하는 상황으로 인하여 정적 타이밍 분석만을 가지고는 타이밍 관련된 버그들을 찾아내는 것이 거의 불가능하게 되었다.

게이트수준 타이밍 시뮬레이션의 제일 큰 문제점은 매우 낮은 시뮬레이션 성능으로 인하여 수행시간이 크게 늘어난다

※ 이 논문은 부산대학교 기본연구지원사업(2년)에 의하여 연구되었음.
† 비 회 원 : 한화시스템 해양연구소 연구원
†† 정 회 원 : 부산대학교 정보컴퓨터공학부 교수
Manuscript Received : October 12, 2018
First Revision : January 2, 2019
Accepted : January 3, 2019
* Corresponding Author : Seiyang Yang(syyang@pusan.ac.kr)

는 것이며, 이를 해결할 수 있는 효과적인 대안이 존재하지 않는다는 것이다. 게이트수준 함수적 시뮬레이션이나 레지스터전송수준(RTL: Register Transfer Level) 함수적 시뮬레이션은 게이트수준 타이밍 시뮬레이션에 비하여 추상화수준이 높음으로 인하여 시뮬레이션의 수행시간이 각각 대략 1/10과 1/100 단축될 수 있을 뿐만 아니라, 하드웨어 에뮬레이터라는 별도의 시뮬레이션 가속장비를 통하여 수행시간을 추가적으로 단축시키는 것이 가능하다. 그러나, 게이트수준 타이밍 시뮬레이션은 SDF(Standard Delay Format)을 통한 타이밍 연산에 엄청난 횟수의 복잡한 부동소숫점 연산들이 필요함으로 하드웨어 에뮬레이터를 활용하는 것도 불가능하다. 따라서 게이트수준 타이밍 시뮬레이션의 수행시간을 단축시킬 수 있는 유일한 방법은 다중 코어 내지는 다중 마이크로프로세서에 다수의 이벤트구동 로직 시뮬레이터들을 병렬적으로 연동시켜서 게이트수준 타이밍 시뮬레이션의 성능을 높이는 것이다. 사실 병렬 로직 시뮬레이션은 이미 오래전부터 최근까지 많은 연구가 진행되어져 왔으며[1-5, 9-11], 최근에는 상용 로직 시뮬레이터들도 대부분 멀티코어를 지원하고 있어서 멀티코어를 활용한 병렬 로직 시뮬레이션이 가능하다[6, 7]. 그러나, 이와 같은 병렬 로직 시뮬레이션들도 모두 RTL 함수적 시뮬레이션이나 게이트수준 함수적 시뮬레이션에 국한하여 사용되고있고, 게이트수준 타이밍 시뮬레이션에는 사용되지 못하고 있는 실정이다. 이의 핵심적 원인으로는 병렬 게이트수준 타이밍 시뮬레이션에서 존재하는 동기 오버헤드와 통신 오버헤드가 병렬 RTL 함수적 시뮬레이션과 병렬 게이트수준 함수적 시뮬레이션에 비하여서 매우 과도하게 존재하기 때문이다[8]. 최근에 이와 같은 과도한 동기 오버헤드와 통신 오버헤드를 줄이는 새로운 방법으로 예측기반 병렬 이벤트구동 로직 시뮬레이션 기법[13]이 제안되었다. [13]에서 제안된 병렬 시뮬레이션 기법은 예측을 통하여 동기 오버헤드와 통신 오버헤드를 획기적으로 감소시킬 수 있음으로 통상적인 기존의 병렬 시뮬레이션 기법들을 통해서는 이룰 수 없는 시뮬레이션 성능 향상을 이룰 수 있다. 그러나, 이의 실제적 달성 여부는 예측의 정확도 및 신속성에 크게 좌우된다. 즉, 예측기반 병렬 이벤트구동 로직 시뮬레이션을 통한 시뮬레이션 성능 향상 정도는 예측의 2가지 중요한 요소인 정확도와 신속성에 전적으로 종속되어진다.

본 논문에서는 이와 같이 예측기반 병렬 이벤트구동 게이트수준 타이밍 시뮬레이션의 성능에 결정적 요소인 높은 예측정확도와 빠른 예측을 동시에 달성할 수 있을 뿐만 아니라, 디버깅 효율성까지도 높일 수 있는 새로운 방법인 공간적 부분시뮬레이션 전략이 적용된 예측기반 병렬 게이트수준 타이밍 시뮬레이션 기법을 제안하고, 이의 효능을 실험을 통하여 보인다.

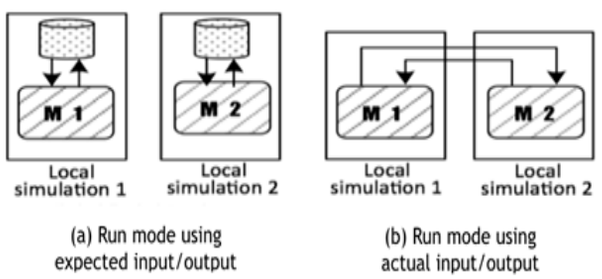
2. 배경 및 관련 연구[13, 14]

병렬 게이트수준 타이밍 시뮬레이션의 성능 향상을 방해하는 제일 큰 요소는 병렬 이벤트구동 로직 시뮬레이션에 필수적인 동기(synchronization)와 통신(communication)에 과도한 오버헤드가 시뮬레이션 실행 과정에서 발생한다는 것인데, 이를 각각 동기 오버헤드와 통신 오버헤드라 칭한다. 병렬 이벤트구동 로직시뮬레이션에서 동기란, 병렬 시뮬레이션에서 모든 로컬시뮬레이션들 각각이 자체의 시뮬레이션 시간들인 로컬시뮬레이션 시간들을 가지고 있는데 이들간에 시뮬레이션 시간상에서 인과(casauality) 관계를 올바르게 유지하는 과정이다(로컬시뮬레이션이란, 병렬 이벤트구동 로직 시뮬레이션에서 동기와 통신을 하면서 병렬적으로 동작하는 개개의 시뮬레이션을 가르킴). 시간 상에서 인과 관계가 올바르게 유지되지 못한 일 예로, 로컬시뮬레이션 1이 병렬 시뮬레이션의 실행으로 로컬시뮬레이션 시간 1000에 로컬시뮬레이션 2로 전달하여야 하는 이벤트를 발생을 한 경우 로컬시뮬레이션 2가 이미 로컬시뮬레이션 시간 1001에 벌써 도달한 상황을 생각할 수 있다. 올바른 인과 관계를 유지하는 대표적 기법으로 비관적(pessimistic) 동기화와 낙관적(optimistic) 동기화가 있다[1-3]. 그러나, 이들은 각각 고유의 약점으로 인하여 동기 오버헤드를 발생하게 되는데, 비관적 동기 기법에서는 인과 관계 오류를 원천적으로 봉쇄하기 위한 너무 빈번한 동기화로 인한 오버헤드, 낙관적 동기 기법에서는 일시적으로 발생한 인과 관계 오류를 수정하기 위한 체크포인트와 롤백 오버헤드로 인한 오버헤드가 그것이다. 또한, 병렬 이벤트구동 로직시뮬레이션에서 통신이란 동기를 맞추면서 병렬적으로 실행되는 로컬시뮬레이션들에서 시뮬레이션되는 로컬설계객체들 사이에서 일어나게 되는 시뮬레이션 이벤트 값들의 실제 전달 과정이다. 병렬 게이트수준 이벤트구동 로직 시뮬레이션의 병렬화가 어려운 이유는 병렬화를 통하여 기대할 수 있는 성능 향상의 대부분 혹은 그 이상을 병렬화를 위하여 필수적인 동기 과정과 통신 과정에서 발생되는 과도한 동기 오버헤드와 통신 오버헤드로 인한 성능 저하가 상쇄시켜 버린다는 것이며, 이와 같은 이유 때문에 현재까지도 병렬 게이트수준 타이밍 시뮬레이터의 상용화가 이루어지지 못하고 있다.

게이트수준 타이밍 시뮬레이션에서 동기 오버헤드가 과도한 이유는 시뮬레이션 시간 상에서 미래에 발생하게되는 이벤트를 올바르게 알 수 없다는 일반적인 이유에 더하여, 게이트수준 타이밍 시뮬레이션에서는 시뮬레이션 시간 상에서 미래에 발생하게 되는 이벤트의 시간단위가 RTL이나 함수적 게이트수준에 비해서 극히 작다는 것에서도 연유한다. 뿐만 아니라, 게이트수준 타이밍 시뮬레이션에서 통신 오버헤드 또한 과도하게 발생되는 이유는 일반적으로 로컬설계객체들 사이에서 경로 상에 지연시간과 관련되어져서 엄청나게 많은 수의 이벤트들이 발생하기 때문이며(반면, RTL이나 함수적 게이트수준에서는 지연시간이 존재하지 않음으로 상대적으로 훨씬 적은 수의 이벤트들이 발생함), 이를 최소화하는 설계 전체를 로컬설계객체들로 최적 분할하는 문제 또한 이론적으로 지극히 어려운 문제로서 최적 분할 혹은 준최적 분할 조차도 현실적으로 불가능하기 때문이다. 따라서, 최근의 병렬 이벤트구동 로직 시뮬레이션 연구들[5, 10]은 GPU(Graphics Processing Unit)을 활용하여 함수적 시뮬레이션의 병렬적 실행을 극대화시킨 것인데, 이를 위하여 CMB(Chandy-Misra-Bryant) 알고리즘[15]을 사용하였다. 그러나 수많은 GPU들을 사용하더라

도 GPU들간에 동기와 통신이 과도하게 발생하게 되는 병렬 게이트수준 타이밍 시뮬레이션의 특성때문에 GPU를 활용한 병렬적 실행을 게이트수준 타이밍 시뮬레이션으로까지 확대하는 것은 성공적으로 이루어지지 않고 있다.

그런데, 병렬 게이트수준 타이밍 시뮬레이션의 문제점인 과도한 동기 오버헤드 및 통신 오버헤드에 인한 병렬 시뮬레이션 성능 제약을 효과적으로 해결할 수 있는 예측기반의 병렬 이벤트구동 로직 시뮬레이션 기법이 최근 새롭게 제안되었다[9, 13]. 제안된 예측기반 게이트수준 타이밍 시뮬레이션은 예상입출력이용-런 모드와 실제입출력이용-런 모드의 두 가지 실행 모드가 상황에 따라서 번갈아 가면서 다음과 같이 실행되어진다. 처음, 로컬시뮬레이션들 각각은 예상입출력이용-런 모드로서 자신이 저장하고 있는 예상입력을 가지고 다른 로컬시뮬레이션들과의 동기 및 통신없이 개별적으로 따로따로 실행되어진다. 이 경우 각 로컬설계객체의 출력에서는 타이밍 시뮬레이션이 진행되면서 실제출력들이 생성되게 되는데, 이들 실제출력들은 타이밍 시뮬레이션 진행 과정에서 각 로컬시뮬레이션에 저장되어 있는 예상출력들과 비교가 진행된다. 만일 이 비교에서 실제출력이 예상출력과 일치하게 되면 각 로컬시뮬레이션은 다른 로컬시뮬레이션들과의 동기 및 통신없이 계속하여 개별적으로 따로따로 실행되어진다. 단, 시뮬레이션이 실행되는 과정 중에 실제출력과 예상출력이 일치하지 않게 되는 시점에서 요구되는 올바른 인과 관계 유지를 위하여 각 로컬시뮬레이션들은 정해진 시뮬레이션 시간에서 체크포인트(checkpoint)를 주기적으로 생성하여야 한다[13, 14]. 이와 같이 진행되는 예상입출력이용-런 모드에서는 병렬 시뮬레이션 성능 제약을 유발시키는 동기 오버헤드 및 통신 오버헤드가 전혀 존재하지 않음을 주목할 필요가 있다. 이와 같은 이유로 인하여 예상입출력이용-런 모드에서는 병렬화 정도에 선형적으로 비례하는 큰 폭의 시뮬레이션의 성능 향상을 기대할 수 있다. 그러나, 실제출력과 예상출력의 비교에서 이들간에 불일치가 발생하게 되면 제일 가까운 체크포인트로 롤백(roll-back)을 진행한 후에, 불일치가 발생한 시점까지는 예상입출력이용-런 모드로 진행한 후, 불일치가 발생한 시점에서부터는 모든 로컬시뮬레이터들은 동기 및 통신 과정을 통하여 통상적인 병렬 게이트-수준 타이밍 시뮬레이션을 진행하게 된다. 이와 같이 모든 로컬시뮬레이터들 사이에 동기 및 통신 과정을 통하여 통상적인 병렬 게이트-수준 타이밍 시뮬레이션을 진행하는 것을 실제입출력이용-런 모드라 한다. 따라서, 병렬 게이트수준 타이밍 시뮬레이션의 성능 향상을 극대화시키기 위해서는 실제입출력이용-런 모드로 전환된 후에는 예상입출력이용-런 모드로 최대한 신속히 재전환하는 것이 절대적으로 바람직하다. 이를 위해서 실제입출력이용-런 모드로 각 로컬시뮬레이션들이 실행되는 경우에서는 각 로컬설계객체의 출력에서 생성되는 실제출력들은 각 로컬시뮬레이션에 저장되어 있는 예상출력들과 비교가 시뮬레이션이 실행시 지속적으로 진행된다. 그리고, 만일 이와 같은 비교에서 실제출력과 예상출력이 일치되는 횟수가 사전에 정한 특정한 횟수와 같아지면 이 시점에서부터는 다시 모든 로컬시뮬레이션들은 동기 및 통신이 필요치 않는 예상입출력이용-런 모드로 재전환되어 타이밍 시뮬레이션을 진행하게 된다. 그림 1은 예측기반의 병렬 이벤트구동 로직 시뮬레이션의 예상입출력이용-런 모드 실행과 실제입출력이용-런 모드 실행 상황 각각을 개념적으로 설명한 그림이다. Fig. 1(a)처럼 예상입출력이용-런 모드 실행에서는 각 로컬시뮬레이션이 다른 로컬시뮬레이션들과의 동기 및 통신 없이 완전 독립적으로 실행됨으로 병렬화를 통한 대폭적 성능 향상이 게이트수준 타이밍 시뮬레이션에서도 가능함을 잘 보여주고 있다.



(a) Run mode using expected input/output　　(b) Run mode using actual input/output

Fig. 1. Two Execution Modes of Prediction-based Parallel Logic Simulation [14]

3. 공간적 부분시뮬레이션 전략이 적용된 예측기반 병렬 게이트수준 타이밍 시뮬레이션

3.1 예측기반 병렬 게이트수준 타이밍 시뮬레이션을 통한 타이밍 검증 시의 문제점

예측기반 병렬 게이트수준 타이밍 시뮬레이션을 통하여 타이밍 시뮬레이션의 성능 향상을 성공적으로 달성할 수 있는 핵심은 바로 “어떤 예측방법을 통하여 시뮬레이션 실행 과정에서 높은 예측정확도를 지속적으로 유지할 수 있는가?”이다. 그러나 이뿐만 아니라, 게이트수준 타이밍 시뮬레이션을 통한 타이밍 검증의 전 과정에 걸쳐서 타이밍 시뮬레이션의 성능을 최대한 높이는 것도 중요하다. 이는 타이밍 검증 과정에서 여러번의 타이밍 에러 관련된 설계수정이 일어나게 되며, 이로 인하여 시뮬레이션의 실행이 수없이 반복되어져서 일어나게 되기 때문이다. 예측기반 병렬 게이트수준 타이밍 시뮬레이션에서 시뮬레이션의 성능 향상을 얻는데 있어서 제일 큰 문제는 이와 같이 타이밍 에러 관련된 설계수정이 일어난 경우에 높은 예측정확도를 가지는 예측 데이터가 존재하지 않을 수 있다는 것이다. 즉, 통상적인 예측기반 병렬 시뮬레이션에서는 바로 이전 시뮬레이션에서 얻어질 수 있는 각 로컬설계객체의 입력값과 출력값을 예측 데이터로서 사용하게 되는 것이 일반적인데, 앞서서 실행된 이전 시뮬레이션을 통하여 타이밍 에러가 발견되어지고 이를 제거하기 위한 설계수정이 일어나게 되면, 변경된 설계에 대한 시뮬레이션을 예측기반 병렬 시뮬레이션에서 변경전 설계에 대한 시뮬레이션에서 얻어진 예측 데이터를 사용하게 되면 예측정확도가 크게 낮아질 가능성이 높아지고, 실제 예측정확도가 떨어진 예측 데이터를 사용한 설계 변경후의 예측기반 병렬 시뮬레이션에서 동기 및 통신 오버헤드가 전혀 존재하지 않는 예

상입출력이용-런 모드 보다는 극심한 동기 및 통신 오버헤드가 유발되는 실제입출력이용-런 모드로 실행되는 게이트-수준 타이밍 시뮬레이션 구간이 상대적으로 커지게 됨으로서 예측기반 병렬 게이트수준 타이밍 시뮬레이션에서 기대할만한 시뮬레이션의 성능 향상을 이루어낼 수 없게 된다.

3.2 기존에 제안된 방법 [14]

이와 같은 예측기반 병렬 게이트수준 타이밍 시뮬레이션의 문제점을 해결하기 위하여 [14]에서는 추상화상위수준의 모델(RTL 모델 또는 함수적 게이트수준 모델)을 활용한 동적예측 방법을 제안하여, 일정한 성능 향상을 얻는 것에 성공하였다. 즉, 설계 변경에 의하여 설계가 변경된 이후에 진행되는 시뮬레이션을 위해서는 해당 설계 변경 전의 게이트수준 타이밍 시뮬레이션 실행에서 얻어진 예측데이터(1차 예측데이터)와 더불어서 설계변경이 이루어진 설계의 상위추상화 모델에 대한 빠른 시뮬레이션을 추가적으로 실행하여 얻어진 예측데이터(2차 예측데이터)를 함께 사용하는 것이 그것이다.

그런데, 2차 예측데이터를 얻기 위하여 실행되는 RTL 설계에 대한 RTL 시뮬레이션은 게이트수준 타이밍 시뮬레이션에 비하여 추상화 수준이 높음으로 훨씬 빠르게 시뮬레이션이 진행될 수 있어서, 2차 예측데이터를 상대적으로 빠른 시간 내에 확보하는 것이 가능하다. 그러나, 이 시뮬레이션은 RTL 설계 전체에 대하여 RTL 시뮬레이션을 진행함으로 2차 예측데이터 생성시간은 RTL 설계 전체에 대하여 RTL 시뮬레이션 실행시간에 전적으로 한정(bound) 되어진다.

3.3 새롭게 제안되는 방법

본 논문에서는, [14]에서 제안된 방법을 더욱 개선시킨 공간적 부분시뮬레이션 전략이 적용된 예측기반 병렬 게이트수준 타이밍 시뮬레이션 방법을 제안한다. 우선, 본 논문에서 제안되는 방법과 [14]에서 제안된 방법의 공통점은 타이밍 에러를 수정하기 위하여 설계가 변경된 이후에 진행되는 게이트수준 타이밍 시뮬레이션을 예측기반 병렬 시뮬레이션으로 수행하기 위하여 필요한 새로운 예측 데이터를 얻기 위해서 추상화상위수준의 모델(RTL 모델 또는 함수적 게이트수준 모델)을 활용하는 것이다. 그러나 [14]에서 제안된 방법과 다른점 및 개선점은, [14]에서는 새로운 예측 데이터를 얻기 위하여 추상화 상위수준의 설계 전체를 시뮬레이션 전체 시간에 걸쳐서 실행시키는 것에 비하여, 본 논문에서 제안되는 방법은 우선 추상화 상위수준의 설계의 일부분만을 대상으로하는 시뮬레이션을 우선 시도하고, 필요 시에 추가적으로 설계의 다른 부분까지를 점진적으로 포함시켜 가면서 시뮬레이션을 진행하는 공간적 부분 시뮬레이션 전략을 사용한다는 것이다. 이와 같은 새로운 방법의 장점은 예측정확도가 동일하게 높은 새로운 예측 데이터를 기존의 방법보다 더 빠르게 얻어낼 수 있다는 것으로, 이를 통하여 추가적인 성능 향상이 가능하다는 것이다. 이제부터는 공간적 부분시뮬레이션 전략을 통한 높은 예측정확도를 가진 새로운 예측 데이터를 어떻게 빠르게 얻어낼 수 있는지를 좀 더 자세히 설명하기로 한다.

최근의 설계는 높은 설계 복잡도로 인하여 설계에는 반드시 최상위층에서부터 최하위층까지 여러단의 계층구조가 존재하며, 이는 시스템온칩(SOC: System On Chip) 설계에서는 더욱 더 그러하다. 공간적 부분 시뮬레이션에서는 이와 같이 설계에 존재하는 여러단의 계층구조를 효과적으로 이용한다. 즉, 앞선 예측기반의 병렬 게이트수준 타이밍 시뮬레이션을 통하여 특정 설계오류(타이밍 에러 또는 함수적 에러)가 발견되고 설계의 특정 부분 M이 수정되어진 경우를 생각해보자. 이 상황에서 새로운 예측 데이터를 얻기 위해서, 본 논문에서 제안되는 방법을 단계별로 구술하면 다음과 같다.

(1) 부분 시뮬레이션 대상 PS 최초 선정.
수정된 설계 전체에 대한 시뮬레이션 대신에 설계의 계층구조 상에 설계의 수정된 부분 M을 포함하는 서브모듈 SM(i)을 시뮬레이션 대상 PS로 설정, PS <= {SM(i)}. 현재 시뮬레이션 시간 ST = 0.

(2) PS에 대하여 공간적 부분시뮬레이션을 다음과 같이 진행.
현재 시뮬레이션 시간 ST에서부터 PS에 설계변경 전의 시뮬레이션에서 저장된 입력값 SI(i)를 인가하며 동시에 설계변경 전의 시뮬레이션에서 저장된 출력값 SO(i)과 시뮬레이션 과정에서 얻어지는 {SM(i)}의 실제 출력값 RO(i)를 실시간으로 비교하여 SO(i)와 RO(i)와 제일 처음으로 다르게 되는 시뮬레이션 시간 Tx까지 또는 시뮬레이션 종료 시점까지 시뮬레이션을 진행. 현재 시뮬레이션 시간 ST <= Tx 또는 시뮬레이션 종료 시점으로 변경. ST가 시뮬레이션 종료 시점과 같게 되면 시뮬레이션을 종료.

(3) 부분 시뮬레이션 대상 PS 확장
시뮬레이션 시간 ST에서 SO(i)와 RO(i)가 달라지는 PS의 출력들과 연결되어 있는 “서브모듈 {SM(k)}의 상태복원”(바로 아래에서 설명)을 실행한 후, 서브모듈 {PS, {SM(k)}}을 시뮬레이션 대상 PS로 재설정, PS <= {PS, {SM(k)}}. 단계 (2)로 진행.

위 과정의 (3) 단계에서 수행되는 서브모듈 {SM(k)}의 상태복원은 다음과 같이 진행한다.

(a) 서브모듈 {SM(k)}를 현재 시뮬레이션 시간 ST에서 제일 가까운 과거시간의 체크포인트 상태 {SM(k)}′로 복원

(b) 제일 가까운 과거시간의 체크포인트 시점에서부터 현재 시뮬레이션 시간 ST까지 설계변경 전의 시뮬레이션에서 저장된 입력값 SI(k)를 인가하여 서브모듈 {SM(k)}에 대한 부분 시뮬레이션을 진행하여 서브모듈 {SM(k)}에 대한 현재 시뮬레이션 시간 ST에서 상태복원

Fig. 2는 이와 같은 부분적 시뮬레이션 실행이 시뮬레이션 시간 상에서 진행되어지는 과정을 예로써 도식화한 것으로서, 설계 전체는 M1, M2, M3, M4의 서브모듈을 가지고 있고 설계변경이 M1 내부에서 이루어진 것으로 설정되어 있다. 이 경우에 정확도가 높은 예측 데이터를 빠르게 생성하기 위한 본 논문에서의 공간적 부분시뮬레이션은 설계 전체에 대한 시뮬레이션 대신에 시뮬레이션 시간 0에서부터 서브모듈 M1

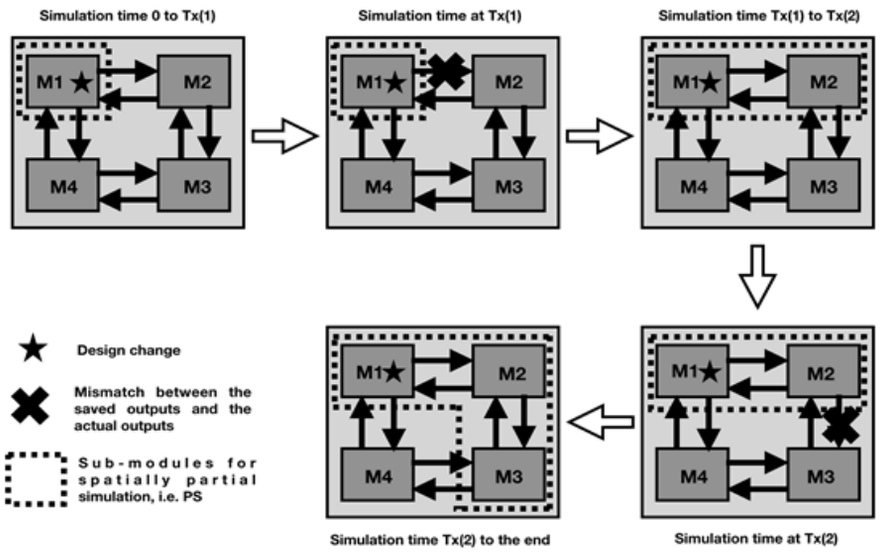


Fig. 2. An Example of Proposed Prediction-based Parallel Simulation Using Spatially Partial Simulation Strategy

만에 대한 시뮬레이션으로 시작하여서 시뮬레이션 시간이 증가되면서 시뮬레이션 시간 Tx(1)에서부터(Tx(1)은 서브모듈 M1의 출력에서 SO(M1)와 RO(M1)가 제일 처음으로 다르게 되는 시뮬레이션 시간) 서브모듈 M1과 M2만에 대한 시뮬레이션으로 확장되고, 최종적으로는 시뮬레이션 시간 Tx(2)에서부터(Tx(2)은 서브모듈 M2의 출력에서 SO(M2)와 RO(M2)가 제일 처음으로 다르게 되는 시뮬레이션 시간) 서브모듈 M1과 M2와 M3만에 대한 시뮬레이션으로 예측 데이터를 생성시키는 상황을 묘사하고 있다 (즉, 모듈 M4에 대해서는 시뮬레이션 전체 시간 동안에 시뮬레이션을 전혀 수행할 필요가 없음). 이와 같은 부분적 시뮬레이션 실행을 진행하는 과정에서 특정 로컬 시뮬레이션의 입력과 출력이 부분적 시뮬레이션 실행에 포함되어져 있으면, 새로운 예측 데이터는 부분적 시뮬레이션 실행을 통하여 얻어지게 된다. 그렇치 않으면, 설계 변경으로 해당 특정 로컬 시뮬레이션의 예측 데이터가 변경되지 않은 것이 됨으로 설계 변경 전의 예측 데이터를 그대로 사용하면 된다.

다음으로는 이와 같은 공간적 부분시뮬레이션 전략이 적용된 예측기반 병렬 게이트수준 타이밍 시뮬레이션 실행을 위하여 이전 시뮬레이션 단계에서 추가적으로 수행되어야 하는 사항들에 대하여 살펴보기로 한다. 우선 서브모듈 {SM(k)}의 상태복원의 (a) 단계를 위하여 이전 시뮬레이션 단계에서 시뮬레이션 전체 시간 동안에 걸쳐서 주기적인 체크포인트 생성이 필요하다. 그러나, 예측기반 병렬 시뮬레이션은 올바른 인과 관계 유지를 위하여 각 로컬시뮬레이션들이 이미 체크포인트를 진행하며 시뮬레이션을 실행하여야 함으로 체크포인트 생성은 본 논문에서의 공간적 부분시뮬레이션 전략이 적용된 예측기반 병렬 게이트수준 타이밍 시뮬레이션 실행을 위하여 새롭게 요구되어야 하는 것은 아니다. 공간적 부분시뮬레이션 전략이 적용된 예측기반 병렬 게이트수준 타이밍 시뮬레이션 실행을 위하여 이전 시뮬레이션 단계에서 추가적으로 수행되어야 하는 단 한가지는 다음과 같다. 즉, 설계 변경 전에 설계에서 서브 모듈별로 분할을 진행하여서 모든 서브 모듈들에 대한 입력과 출력들에 대해서 시뮬레이션 시간 전체에 걸쳐서 시그널 덤프를 진행하여야 한다는 것이 그것이다. 이와 같은 모든 서브 모듈들의 입력과 출력들에 대한 시그널 덤프는 시뮬레이션 시간을 소폭 증가시키지만(대략 5% 내외), 이 덤프 데이터는 본 논문에서의 공간적 부분시뮬레이션 전략을 통하여 예측기반 병렬 시뮬레이션의 성능 향상을 가능하게 할 뿐만 아니라, 설계 오류를 찾는 디버깅 시에 매우 유용한 입출력 파형 데이터이기도 하다.

만일 설계 오류를 찾는 디버깅 과정에서 분할이 된 설계의 모든 서브 모듈들에 대한 입력과 출력들에 대해서 디버깅에 앞서서 진행된 직전 시뮬레이션에서 시뮬레이션 시간 전체에서 시그널 덤프를 진행한 입출력 파형 데이터들만으로는 설계 오류를 찾기 어려운 경우, 즉 설계 오류의 원인이 설계의 특정 서브 모듈의 내부에 깊숙히 자리잡은 경우에는 특정 서브 모듈에 존재하는 모든 신호선들에 대한 완전 덤프까지도 필요할 수 있다. 그런데, 여기서 어려운 점은 특정 서브 모듈이 구체적으로 설계의 어느 모듈이냐 하는 것과 해당 서브 모듈에 대한 덤프를 시뮬레이션 전체 시간 구간대에서 어느 구간대에 걸쳐서 하여야만 하는지를 시뮬레이션 실행 전에 미리 알 수 없다는 것이다. 따라서 최악의 경우에는 단 한번의 시뮬레이션에서 특정 서브 모듈이 아닌 설계의 전체에 존재하는 모든 신호선들에 대한 완전 덤프를 시뮬레이션 전체 시간 구간대에서 진행하거나(설계 전체에 대한 완전덤프를 하는 단 한번의 시뮬레이션), 수차례의 시뮬레이션에서 특정 서브 모듈에 존재하는 신호선들에 대한 덤프를 반복적으로 진행하여야만 한다(특정 모듈에 대한 부분덤프를 하는 여러번의 시뮬레이션). 그러나, 설계 전체에 대한 완전덤프를 하는 단 한번의 시뮬레이션 방법이나 특정 모듈에 대한 부분덤프를 하는 여러번의 시뮬레이션 방법은 모두 시뮬레이션의 실행 시간이

극히 길어지고 설계 내부에 존재하는 신호선들이 극히 많아서 설계 전체에 대한 완전덤프의 오버헤드로 인하여 시뮬레이션 시간이 크게 증가하는 게이트수준 타이밍 시뮬레이션에서 디버깅의 효율성을 크게 저하시키게 만드는 요인으로 작용하게 된다. 특히 최근의 시스템온칩의 설계들은 설계복잡도가 매우 높음으로 설계 전체에 대한 완전덤프를 하는 단 한번의 시뮬레이션 방식은 대폭적으로 증가하는 시뮬레이션 실행시간 뿐만 아니라 덤프 데이터의 용량이 수백기가 바이트를 쉽게 넘어감으로 현실적으로도 적용할 수 없다.

본 논문에서의 공간적 부분시뮬레이션 전략이 적용된 예측기반 병렬 게이트수준 타이밍 시뮬레이션 기법을 활용하게 되면, 설계 오류를 찾기위한 덤프 데이터를 얻기 위한 추가적인 시뮬레이션을 설계 전체가 아니라 설계 오류의 원인을 가진 것으로 의심이 되는 설계의 일부분에 대해서만 (공간적으로 한정하여) 시뮬레이션이 가능하게 할 뿐만 아니라, 시뮬레이션 전체 시간 구간이 아니라 (시간적으로 한정하여) 특정 시간 구간에만 걸쳐서 할 수 있게 함으로서 디버깅의 효율성을 크게 높이는 매우 중요한 추가적인 장점을 제공하게 된다. 이를 개념적으로 도식화한 것이 Fig. 3이다. 첫번째 그림은, 서브모듈 M1, M2, M3, M4로 이루어진 설계 전체에 대한 시뮬레이션 실행(첫번째 시뮬레이션)을 본 논문에서의 예측기반 병렬 시뮬레이션으로 진행하는 과정에서 주기적으로 체크포인트를 진행하는 동시에, 분할이 이루어진 서브모듈들의 입력과 출력을 저장하는 상황을 나타낸 것이다. 첫번째 시뮬레이션의 목적은 설계가 설계자의 의도대로 동작하는지를 여부를 판단하는 목적으로, 따라서 당연히 시뮬레이션 전체 시간과 설계 전체에 대하여 진행되어야만 한다. 만일, 이와 같은 첫번째 시뮬레이션에서 설계가 설계자의 의도대로 동작하지 않는다면 설계에 대한 디버깅 과정이 필요한데, 이 디버깅 과정에서도 1회 이상의 시뮬레이션(디버깅을 위한 시뮬레이션)이 필요한 것이 일반적이다. 즉 디버깅을 위한 시뮬레이션에서는 설계에 존재하는 설계오류의 위치와 원인을 찾기위한 파형 데이터를 생성하는 것이 주목적이다. 여기에서 중요한 것은 앞서서 이미 언급한대로 설계의 어느 부분에 대하여 어느 시뮬레이션 구간에 대한 파형 데이터를 만들어 내어야 특정 설계오류의 위치와 원인을 찾을 수 있느냐는 것인데, 문제는 디버깅을 위한 시뮬레이션 전에는 이를 미리 정확히 알 수 없다는 것이다. 두번째 그림은, 이와 같은 디버깅을 위한 시뮬레이션을 본 논문에서의 공간적 부분시뮬레이션 전략이 적용된 예측기반 병렬 게이트수준 타이밍 시뮬레이션 기법을 활용하여서

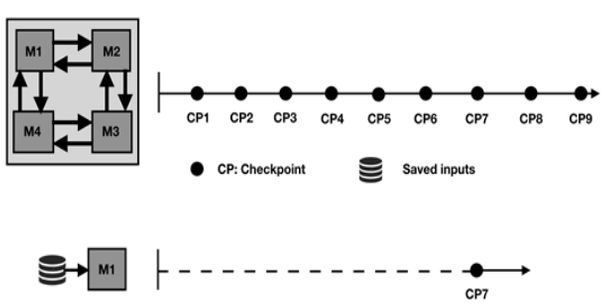


Fig. 3. Spatially and Temporally Partial Simulation for Debugging

디버깅을 위한 시뮬레이션을 설계 전체가 아닌 설계오류의 위치를 가진 것으로 의심되는 서브모듈 M1에 대하여 시뮬레이션 전체 시간이 아닌 설계오류의 현상을 파형 데이터에 기록할 수 있다고 생각하는 특정 시뮬레이션 시간 구간대에 한정하여서만 실행하게 되는 상황을 묘사하는 그림이다. 당연히, 이와 같은 공간적/시간적으로 축소된 부분 시뮬레이션 방식은 공간적/시간적으로 축소가 되지않은 통상적 시뮬레이션보다 훨씬 빠르게 실행되어 짐으로서 디버깅을 위한 시뮬레이션을 위하여 제일 적합한 방식이다. 이상과 같은 공간적 부분시뮬레이션 전략은 설계수정이 된 후의 RTL의 함수적 시뮬레이션를 예측기반 병렬 시뮬레이션으로 실행하는 경우에서도 적용가능 하지만, 설계수정이 된 후의 게이트수준의 타이밍 시뮬레이션을 예측기반 병렬 시뮬레이션으로 실행하는 경우에서 더 효율적이며, 이의 이유로는 다음과 같은 것들이 있다. 첫째, RTL의 함수적 시뮬레이션 또는 게이트수준의 함수적 시뮬레이션과 비교하여서 게이트수준 타이밍 시뮬레이션의 실행 시간은 이들과 비교하여 평균적으로 각각 100배 또는 10배 이상 오래걸리지만, 하드웨어 에뮬레이터 등의 하드웨어기반의 가속장비를 통하여 시뮬레이션 성능을 향상시킬 수 있는 RTL의 함수적 시뮬레이션 또는 게이트수준의 함수적 시뮬레이션과는 달리, 게이트수준 타이밍 시뮬레이션은 전통적인 이벤트-구동 로직 시뮬레이션을 통해서만 가능하며 다른 대안이 존재하지 않는다. 둘째, 설계 수정 후에 진행되는 게이트수준 타이밍 시뮬레이션에서, 본 논문에서의 방법은 높은 예측정확도를 가지는 예측 데이터를 빠르게 생성할 수 있음으로 인하여 게이트수준 타이밍 시뮬레이션의 실행 시간 자체를 단축시키는 것이 가능하다. 셋째, 설계 수정 전에 진행되는 디버깅 과정에서, 본 논문에서의 방법은 설계에 오류가 존재하여서 오류의 원인은 발견하고 수정하는 디버깅을 위하여 필요한 추가적인 디버깅을 위한 시뮬레이션의 실행 시간 자체도 대폭적으로 단축시키는 것이 가능하다.

4. 실험 결과

본 논문에서 제안된 공간적 부분시뮬레이션 전략이 적용된 예측기반 병렬 게이트수준 타이밍 시뮬레이션을 통한 시뮬레이션 성능 향상의 정도를 실험적으로 확인하기 위하여 다수의 설계들을 대상으로 설계에 존재하는 설계오류를 찾는 디버깅을 위한 게이트수준 타이밍 시뮬레이션을 1회 진행하여 설계 오류를 찾은 후 설계 오류를 제거하는 설계 변경을 수행하고, 이후 디버깅이 올바르게 진행되었는지를 최종 확인하는 게이트수준 타이밍 시뮬레이션을 한번 더 실행(즉 게이트수준 타이밍 시뮬레이션을 2회 실행)하는 것에 대한 전체 시뮬레이션 실행 시간 측정을 위한 실험을 진행하였다.

기존의 순차 게이트수준 타이밍 로직 시뮬레이션(A)과 기존 통상적 병렬 게이트수준 타이밍 로직 시뮬레이션(B)은 순차 시뮬레이션과 병렬 시뮬레이션 모두를 지원하는 상용 베릴로그 시뮬레이터를 각각 싱글 코어 상에서 순차 시뮬레이션과 멀티 코어(각 설계별로 병렬 시뮬레이션들에서 사용된 코어수는 표의 3번째 컬럼 참조)상에서 병렬 시뮬레이션으로 실행하여 시뮬레이션 수행시간을 측정한 것이다. 이 실험에서 JPEG과

3serial_atpg에서는 병렬 시뮬레이션이 순차 시뮬레이션에 비하여 각각 1.4(10884/7550)배, 1.1(14272/12820)배의 성능 향상을 얻을 수 있었으나 AC97, AES, PCI, Doorlock에서는 오히려 병렬 시뮬레이션이 순차 시뮬레이션에 비하여 각각 30.8(10664/346)배, 2.7(36940/13914)배, 13.7(25758/1882)배, 15.3(18678/1222)배의 큰 폭의 성능 저하가 관찰되었다. 이는 통상적 병렬 게이트수준 타이밍 로직 시뮬레이션의 경우에는 로컬시뮬레이션들 사이에 발생하는 과도한 동기 오버헤드와 통신 오버헤드가 병렬 실행을 통하여 기대할 수 있는 성능 향상 효과를 완전히 무력화시키고 이와 같은 오버헤드가 오히려 순차 시뮬레이션에 비하여 속도 저하를 유발하게 됨을 실험을 통해서도 확인한 것이다. 다음으로는 예측정확도를 높이기 위하여 제일 성능적으로 우수한 기존의 방법인 설계변경 후에 새로운 예측데이터를 생성하기 위하여 추상화 상위수준에서의 시뮬레이션을 전체 시뮬레이션 시간 구간에 걸쳐서 진행하게 되는 예측기반 병렬 이벤트 구동 게이트수준 타이밍 로직 시뮬레이션 방법[14]을 적용한 경우(C)에 대해서 성능을 측정하였다. 마지막으로, 본 논문에서 제안된 공간적 부분시뮬레이션 전략이 채용된 예측기반 병렬 게이트수준 타이밍 시뮬레이션 기법(D)을 사용한 전체 시뮬레이션 수행시간 측정 결과를 볼 수 있다 (대괄호안의 숫자 2개는 각각 설계오류를 찾는 게이트수준 타이밍 시뮬레이션(1st)과 디버깅 후에 디버깅이 올바르게 진행되었는지를 최종 확인하는 게이트수준 타이밍 시뮬레이션(2nd)을 가르킴).

표에서와 같이 본 논문에서 제안된 방법을 통하여 실질적인 게이트수준 타이밍 시뮬레이션의 성능이 기존의 전통적인 순차적 게이트수준 타이밍 시뮬레이션(A), 그리고 기존의 멀티코어 기반의 병렬적 게이트수준 타이밍 시뮬레이션(B) 모두에 비하여 모든 설계 사례들에서 일관되게 큰 폭으로 향상됨을 볼 수 있으며, 평균적으로는 각각 3.7배와 9.7배의 시뮬레이션 성능 향상이 관측되었다. 또한, 표에서 보는 것과 같이 모든 설계들에서 기존의 방법(C)과 동일한 높은 예측정확도(5번째 컬럼과 대비하여 7번째 컬럼)를 얻을 수 있음을 알 수 있으며, 시뮬레이션 수행시간이 기존의 제일 성능적으로 우수한 예측기반 병렬 이벤트구동 게이트수준 타이밍 로직 시뮬레이션 방법과 비교해서도 평균적으로 2.3배 더 향상되었음을 알 수 있다. 특히, AC97은 3.8배, JPEG은 3.4배의 큰 성능 향상을 얻을 수 있었고 3serial_atpg와 PCI도 2.2배 이상의 성능 향상을 보였다. 추가적으로 주목할 점은 기존 방법[14]과 비교하여 본 논문에서의 방법 또한 동일한 높은 예측정확도를 유지하고 있을 뿐만 아니라, 이와 같은 높은 예측정확도가 설계들의 게이트 복잡도와 시뮬레이션 실행 시간과는 거의 무관하게 얻어지고 있다는 것이다. 즉 본 논문에서 제안된 방법은 설계의 게이트 복잡도 또는 시뮬레이션 실행 시간과 독립적으로 높은 예측정확도를 빠르게 확보하는 것을 가능하게 함으로서 병렬 게이트수준 타이밍 시뮬레이션의 성능 향상에 효과적인 방법임을 실험을 통해서도 확인할 수 있다.

즉, 본 논문에서 제안된 공간적 부분시뮬레이션 전략이 적용된 예측기반 병렬 게이트수준 타이밍 시뮬레이션 기법은 기존의 제일 성능이 우수한 예측기반 병렬 게이트수준 타이밍 시뮬레이션 기법과 비교하여서 어떠한 경우에서도 시뮬레이션의 성능이 떨어지지 않음으로 기존의 예측기반 병렬 게이트수준 타이밍 시뮬레이션 방법을 무조건적으로 대체할 수 있다. 특히, Table 1에서 수행시간을 측정한 시뮬레이션은 앞서 3장 (3)절의 후반부에서 설명한 디버깅을 위한 시뮬레이션을 오직 1회만으로 한정한 것으로서, 여기에 추가적인 디버깅을 위한 시뮬레이션 실행을 고려한다면 본 논문에서 제안된 공간적 부분시뮬레이션 전략이 적용된 예측기반 병렬 게이트수준 타이밍 시뮬레이션 기법을 통한 전체 시뮬레이션 시간의 단축은 앞서 얻어진 수치보다도 훨씬 클 것이다. 뿐만 아니라, 성능 평가를 위한 실험에 사용된 설계들은 베릴로그 코드가 퍼블릭 도메인에 공개되어져서 쉽게 구할 수는 있지만, 상대적으로 설계 규모가 크지 않은 소/중 규모급의 설계들이다. 그런데 대규모급 이상의 설계들은 소/중 규모급 설계들에 비하여 게이트수준 타이밍 시뮬레이션에서 처리하여야 하는 게이트들이 훨씬 많을 뿐만 아니라 발생되는 이벤트의 수도 크게 증가하게 되며, 이로 인하여 순차적 게이트수준 타이밍 시뮬레이션의 실행시간은 더욱 크게 증가하게 된다. 그러나 본 논문에서 제안된 기법을 통하여 얻게 되는 높은 예측정확도는 위 실험을 통하여서도 알 수 있는 것과 같이 설계 규모와는 거의 무관함을 알 수 있다. 따라서 실제 산업체에서 진행되는 AP(Application Processor)와 같은 초대규모의 시스템온칩 설계들에 본 논문에서 제안된 시뮬레이션 기법이 적용되는 경우에는 시뮬레이션 성능 향상에 있어서 더 좋은 결과를 낼 수 있을 것으로 판단된다.

Table 1. Experimental Result

Design name	**A** Execution time of traditional sequential logic sim (sec) [6]	**B** Traditional multi-core parallel logic sim [6]		**C** Previous best predicition-based parallel logic sim [14]		**D** Proposed approach	
		# of cores	Execution time (sec)	Prediction accuracy	Execution time (sec)	Prediction accuracy	Execution time (sec) [1st, 2nd]
AC97	346	4	10,664	95.1%	306	95.1%	81 [72, 9]
JPEG	10,884	3	7,550	100%	6,534	100%	1,927 [1751, 176]
3serial_atpg	14,272	4	12,820	99.5%	7,064	99.5%	2,701 [2451, 250]
AES	13,914	4	36,940	100%	10,668	100%	5,867 [5331, 536]
PCI	1,882	4	25,758	100%	1,480	100%	659 [574, 85]
Doorlock	1,222	2	18,678	100%	662	100%	361 [314, 47]
Total (sec)	42,520	NA	112,410	NA	26,714	NA	11,596

5. 결 론

본 논문에서는 공간적 부분시뮬레이션 전략이 적용된 예측기반 병렬 게이트수준 타이밍 시뮬레이션 기법을 새롭게 제안하였다. 최근들어서 여러 이유들로 인하여 게이트수준 타이밍 시뮬레이션의 중요성이 다시 커지고 있지만, 설계 복잡도에 비례하여서 게이트수준 타이밍 시뮬레이션의 성능은 더욱 낮아지고 있는 것이 현실이다. 그러나, 함수적 시뮬레이션의 성능 향상을 위하여 광범위하게 사용 중인 하드웨어 에뮬레이터 등을 사용하는 하드웨어 기반의 시뮬레이션은 게이트수준 타이밍 시뮬레이션에 적용할 수 없다는 근본적인 문제점이 있다.

본 논문에서 새롭게 제안된 기법은 새로운 예측 데이터를 얻기 위하여 추상화 상위수준의 설계 전체를 시뮬레이션 전체 실행 시간에 걸쳐서 실행시키는 대신, 추상화 상위수준의 설계의 일부분만을 대상으로하는 시뮬레이션을 우선 시도하고, 필요시에 추가적으로 설계의 다른 부분까지를 점진적으로 포함시켜 가면서 시뮬레이션을 진행하는 공간적 부분 시뮬레이션 전략을 사용한다. 이와 같은 새로운 방법을 통하여 예측정확도가 동일하게 높은 새로운 예측 데이터를 기존의 방법보다 훨씬 빠르게 얻어낼 수 있어서 설계변경 후의 게이트수준 타이밍 시뮬레이션의 성능 향상을 가능하게 할뿐만 아니라, 설계변경 전의 디버깅 과정에서 디버깅에 필요한 설계에 대한 덤프 데이터를 얻기 위해 필요한 디버깅 시뮬레이션의 실행 시간도 크게 단축할 수 있음으로 인하여 디버깅의 효율성도 함께 높일 수 있다. 제안된 공간적 부분시뮬레이션 전략이 적용된 예측기반 병렬 게이트수준 타이밍 시뮬레이션 기법은 다수의 설계들을 대상으로한 실험을 통해서도 게이트수준 타이밍 시뮬레이션의 성능 향상에 매우 효과적임을 확인할 수 있었다.

References

[1] R.M. Fujimoto, “Parallel Discrete Event Simulation,” *Communication of the ACM*, Vol.33, No.10, pp.30-53, Oct. 1990.

[2] D.M. Nicol, “Principles of Conservative Parallel Simulation,” in *Proceedings of the 28th Winter Simulation Conference*, pp. 128-135, 1996.

[3] R.M. Fujimoto, “Time Warp on a Shared Memory Multiprocessor,” *Transactions of the Society for Computer Simulation*, Vol.6, No.3, pp.211-239, Jul. 1989.

[4] L. Li and C. Tropper, “A design-driven partitioning algorithm for distributed Verilog simulation,” in *Proc. 20th International Workshop on Principles of Advanced and Distributed Simulation(PADS)*, pp.211-218, 2007.

[5] D. Chatterjee, A. DeOrio, and V. Bertacco, “Event-driven gate-level simulation with general purpose GPUs,” in *Proc. of Design Automation Conference(DAC09)*, pp. 557-562, Jun. 2009.

[6] IUS Simulator Usermanual, Cadence Design Systems [Internet], http://www.cadence.com.

[7] VCS Simulator Usermanual, Synopsys [Internet], http://www.synopsys.com.

[8] K. Chang and C. Browy, “Parallel Logic Simulation: Myth or Reality?” *Computer*, Vol.45, No.4, pp.67-73, Apr. 2012.

[9] Jaehoon Han et al, “Predictive parallel event-driven HDL simulation with a new powerful prediction strategy,” in *Proc. of Design, Automation and Test in Europe Conference and Exhibition(DATE)*, pp.1-3, Mar. 2014.

[10] Yuhao Zhu, Bo Wang, and Yangdong Deng, “Massively Parallel Logic Simulation with GPUs,” *ACM Transactions on Design Automation of Electronic Systems(TODAES)*, Vol.16, No.3, pp.1-20, Jun. 2011.

[11] James Gross et al, “Multi-Level Parallelism for Time- and Cost-efficient Parallel Discrete-Event Simulation on GPUs,” in *Proc. of 26th ACM/IEEE Workshop on Principles of Advanced and Distributed Simulation 2012(PADS 2012)*, Jun. 2012.

[12] Gate-level Simulation Methodology, Whitepaper, Cadence Design Systems [Internet], www.cadence.com.

[13] Seiyang Yang, “A New Prediction-based Parallel Event-driven Logic Simulation,” *Journals of KIPS/Computer and Communication Systems*, Vol.4. No.3, pp. 85-90, Mar. 2015.

[14] Seiyang Yang, “Performance Improvement of Prediction-based Parallel Gate-Level Timing Simulation Using Prediction Accuracy Enhancement Strategy,” *Journals of KIPS/Computer and Communication Systems*, Vol.5. No.12, pp.439-446, Dec. 2016.

[15] Chandy, K. M. and Misra, J., “Distributed simulation: A case study in design and verification of distributed programs,” *IEEE Trans. Softw. Engin.*, SE-5, 5, pp.440-452, 1979.

한 재 훈

https://orcid.org/0000-0002-0031-6836

e-mail : jh.han@hanwha.com

2013년 부산대학교 정보컴퓨터공학부(학사)

2015년 부산대학교 컴퓨터공학과(공학석사)

2015년～현 재 한화시스템 해양연구소 연구원

관심분야 : SoC 설계 및 검증, 미들웨어

양 세 양

https://orcid.org/0000-0002-4386-8593

e-mail : syyang@pusan.ac.kr

1981년 고려대학교 전자공학과(학사)

1985년 고려대학교 컴퓨터공학과(공학석사)

1990년 University of Massachusetts, Amherst 컴퓨터공학과(공학박사)

1991년～현 재 부산대학교 정보컴퓨터공학부 교수

관심분야 : 설계자동화, 특히 SoC 검증