KIPS Tr. Comp. and Comm. Sys.
Vol.4, No.3 pp.85~90 pISSN: 2287-5891

http://dx.doi.org/10.3745/KTCCS.2015.4.3.85

A New Prediction-Based Parallel Event-Driven Logic Simulation

Seiyang Yang†

ABSTRACT

In this paper, anew parallel event-driven logic simulation is proposed. As the proposed prediction-based parallel event-driven simulation method uses both prediction data and actual data for the input and output values of local simulations executed in parallel, the synchronization overhead and the communication overhead, the major bottleneck of the performance improvement, are greatly reduced. Through the experimentation with multiple designs, we have observed the effectiveness of the proposed approach.

Keywords : Verification, Event-Driven Simulation, Parallel Simulation

새로운 예측기반 병렬 이벤트구동 로직 시뮬레이션

양 세 양†

요 약

본 논문에서는 새로운 병렬 이벤트구동 로직 시뮬레이션 기법을 제안한다. 제안한 예측에 기반한 병렬 이벤트구동 시뮬레이션 기법은 병렬 이벤트구동 시뮬레이션에서 다른 로컬시뮬레이션과의 연동 과정에서 사용되는 입력값과 출력값에 실제값과 예측값을 함께 사용함으로써 성능 향상의 제약 요소인 동기 오버헤드 및 통신 오버헤드를 크게 감소시킬 수 있다. 본 논문에서 제안한 예측기반 병렬 이벤트구동 로직 시뮬레이션의 유용함은 다수의 디자인들에 적용한 실험을 통하여 확인할 수 있었다.

키워드 : 검증, 이벤트구동 시뮬레이션, 병렬 시뮬레이션

1. 서 론

최근의 모든 마이크로프로세서는 코어가 여러 개인 멀티코어 구조를 가지고 있다. 병렬 이벤트구동 로직 시뮬레이션은 이와 같은 다중 코어 내지는 다중 마이크로프로세서에 다수의 이벤트구동 로직 시뮬레이터들을 병렬적으로 연동시켜서 시뮬레이션의 성능을 높이고자 하는 것으로, 이미 오래전부터 최근까지 많은 연구가 진행되어 왔으며[1-5, 9-11], 최근에는 상용 로직 시뮬레이터들도 멀티코어를 지원하고 있다[6, 7]. 최근의 SOC(System On Chip) 설계에서도 이벤트구동 로직 시뮬레이터를 활용한 로직 시뮬레이션을 제일 많이 사용하고 있지만, 수억 게이트급인 최근 디자인들에 대한 로직 시뮬레이션의 실행 성능은 매우 낮아서 시뮬레이션의 성능 향상을 높이는 시도는 이론적으로뿐만 아니라 현실적으로도 매우 중요한 과제이다. 병렬 이벤트구동 로직 시뮬레이션에서는 시뮬레이션 하고자 하는 디자인 전체를 하나의 시뮬레이터로 실행시키지 않고 디자인을 여러 개의 설계객체(이를 본 논문에서는 로컬설계객체라 칭함)들로 분할(partition)한 후에 각각의 설계객체에 하나의 시뮬레이터를 할당하여 이들을 연동하여 병렬적으로 실행시킴으로써, 하나의 시뮬레이터로 실행시키는 것에 비하여 높은 시뮬레이션 성능을 얻고자 한다. 그러나 지금까지의 연구 결과와 상용화 시도는 그리 성공적이라 할 수 없는데, 이의 핵심적 원인은 병렬 로직 시뮬레이션에서 존재하는 동기 오버헤드와 통신 오버헤드가 매우 과도하게 존재하기 때문이다[8]. 본 논문에서는 예측에 기반한 새로운 병렬 이벤트구동 로직 시뮬레이션 기법을 제안하고, 이의 효능을 실험을 통하여 보인다. 본 논문의 구성은 2절에서 배경 및 관련 연구를 언급하고, 본론에 해당하는 3절에서 제안한 예측기반 병렬 이벤트구동 로직 시뮬레이션을 자세히 설명하고, 4절에서 실험 결과를 보이고, 마지막 절에서 결론을 이야기한다.

2. 배경 및 관련 연구

서론에서 언급한 병렬 이벤트구동 로직 시뮬레이션에서

※ 이 논문은 부산대학교 자유과제 학술연구비(2년)에 의하여 연구되었음.
† 정 회 원 : 부산대학교 정보컴퓨터공학부 교수
Manuscript Received : October 6, 2014
First Revision : December 9, 2014
Accepted : December 9, 2014
* Corresponding Author : syyang@pusan.ac.kr

제일 큰 문제점인 동기 오버헤드와 통신 오버헤드 문제를 좀 더 분석하기 위하여 우선 병렬 이벤트구동 로직 시뮬레이션 기술에서 사용되는 용어들을 설명한다.

- 로컬시뮬레이션 : 병렬 이벤트구동 시뮬레이션에서 분할된 로컬설계객체들 각각에 대한 시뮬레이션
- 로컬시뮬레이터 : 병렬 이벤트구동 시뮬레이션에서 로컬시뮬레이션을 수행하는 시뮬레이터
- 로컬시뮬레이션 시간 : 병렬 시뮬레이션을 진행하는 과정에서 각 로컬시뮬레이션들이 가지게 되는 시뮬레이션 시간

병렬 이벤트구동 로직 시뮬레이션의 결과는 단일 이벤트구동 로직 시뮬레이션의 결과와 항상 동일하여야만 한다. 그런데 하나의 로직 시뮬레이터에 의하여 수행되는 단일 시뮬레이션에서는 오직 하나의 시뮬레이션 시간이 시뮬레이션 과정에 존재하는 것과는 달리, N개의 로직 시뮬레이터로 수행되는 병렬 시뮬레이션에서는 N개의 로컬 시뮬레이션 시간이 따로따로 존재함으로써 이벤트들의 인과(causality) 관계가 로컬시뮬레이션 간에 어긋남으로 틀려진 이벤트 순서에 의하여 전체 시뮬레이션이 올바르지 않을 수 있다. 동기(synchronization)란 이와 같은 다수의 로컬 시뮬레이션 시간들이 존재하는 경우에도 이벤트들의 인과 관계를 올바르게 유지하는 과정이다. 통신(communication)이란 로컬 시뮬레이션들 사이에 로컬설계객체들에 의하여 생성되는 이벤트를 실제적으로 주고받는 과정으로, 병렬 시뮬레이션 과정에서 일어나는 통신의 양은 전체 디자인을 어떻게 분할하였는지에 크게 좌우된다. 병렬 이벤트구동 시뮬레이션에서 동기 방식은 크게 비관적(pessimistic) 방식과 낙관적(optimistic) 방식으로 나누어진다[1-3]. 비관적 동기 방식에서는 로컬 시뮬레이션들 간에 절대 인과 관계 위반이 일어나지 않도록 모든 로컬 시뮬레이션 시간들을 항상 일치시키면서 시뮬레이션을 진행하게 됨으로, 이를 록-스텝(lock-step) 방식이라고도 한다. 이와 같은 비관적 동기 방식의 문제점은 모든 로컬 시뮬레이션 시간들을 록-스텝으로 항상 일치시키기 위하여 과도한 동기 오버헤드가 발생하게 되고 이로 인하여 병렬 시뮬레이션의 성능이 크게 제약된다는 것이다. 반면, 낙관적 동기 방식에서는 우선 각 로컬 시뮬레이션이 자신의 로컬 시뮬레이션 시간에 맞추어 독립적으로 진행될 수 있도록 허용함으로, 일시적으로 로컬 시뮬레이션들 간에 인과 관계 위반이 일어날 수 있다. 따라서 낙관적 동기 방식에서는 인과 관계 위반이 일어난 경우에 이를 바로잡아 주는 메커니즘이 필요한데, 이는 주기적인 체크포인트(checkpoint) 생성과 롤백(rollback)을 통하여 가능하다[3]. 체크포인트를 위해서는 시뮬레이션 프로그램의 실행 이미지 전체를 저장하거나 디자인 이미지 전체 내지는 일부를 보조기억장치에 저장하여야 하므로, 주기적인 체크포인트 생성은 디자인이 큰 경우(최근 디자인들의 시뮬레이션 프로그램 실행 이미지는 수십 기가바이트 규모임)에는 시뮬레이션의 성능을 크게 저하시키는 별도의 문제점을 초래한다. 뿐만 아니라 낙관적 동기 방식에서 인과 관계 위반이 빈번하게 발생하게 되면 빈번한 롤백 후 재실행에 의한 오버헤드가 추가되므로 시뮬레이션의 성능은 더욱 저하되게 된다.

병렬 이벤트구동 로직 시뮬레이션에서는 앞서 설명한 과도한 동기 오버헤드뿐만 아니라, 통신 오버헤드도 큰 문제가 된다. 특히 통신 오버헤드는 병렬화의 정도를 증가시킬수록(즉, 로컬 시뮬레이터의 수를 늘릴수록) 급격하게 증가하게 되는데, 이는 디자인 전체를 로컬 시뮬레이터의 수만큼의 로컬 설계객체로 분할하는 과정에서 로컬 설계객체의 입력단과 출력단에 존재하는 시그널들의 수가 매우 많아지는 것이 일반적인 상황이기 때문이다. 최근 국내 유수의 반도체회사에서 상용 멀티코어용 병렬 이벤트구동Verilog 시뮬레이터들 2개에 대한 벤치마크 테스트에서 2~3개의 적은 코어 수를 사용한 경우에서는 어느 정도의 성능 향상이 가능하지만 코어 수를 그 이상 증가시키면 오히려 단일 시뮬레이션에 비교하여 뒤떨어지는 결과가 도출되었는데, 이와 같은 결과는 산업체에서 설계되는 큰 규모의 회로들에 대한 병렬 시뮬레이션 과정에서는 코어 수가 조금만 늘어나면 실제동기 오버헤드와 통신 오버헤드가 급격하게 커지기 때문이다.

3. 새로운 예측기반 병렬 이벤트구동 로직 시뮬레이션

3.1 예측기반 병렬 이벤트구동 로직 시뮬레이션

본 논문에서 새롭게 제안하는 병렬 이벤트구동 로직 시뮬레이션의 핵심은 병렬 로직 시뮬레이션에서의 제일 큰 문제점인 과도한 동기 오버헤드 및 통신 오버헤드를 효과적으로 감소시키는 것이다. 각 로컬 시뮬레이터에서 실행되는 로컬 시뮬레이션을 위해서는 로컬 설계객체의 시뮬레이션 실행 과정에 필요한 올바른 입력 이벤트들이 계속적으로 필요한데, 이 올바른 입력 이벤트들은 병렬 시뮬레이션 실행 과정에서 다른 로컬 시뮬레이션으로부터 특정 시뮬레이션 시점에서 공급되는 것으로 이를 위하여 기존의 모든 병렬 이벤트구동 로직 시뮬레이션 방법들에서는 통신 및 동기가전체 시뮬레이션 시간에 걸쳐서 반드시 필요하게 된다. 즉 기존의 모든 병렬 이벤트구동 로직 시뮬레이션 기술에서 시뮬레이션 성능 향상의 병목으로 작용하는 통신 오버헤드 및 동기 오버헤드는 이 올바른 입력 이벤트들을 특정 로컬 시뮬레이션에서부터 또 다른 특정 로컬 시뮬레이션으로 특정(혹은 올바른) 시뮬레이션 시점에 전달하기 위하여 발생하게 된다(본 논문에서는 이와 같이 특정 로컬 시뮬레이션에서부터 또 다른 특정 로컬 시뮬레이션으로 특정 시뮬레이션 시점에 실제 전달되는 입력 이벤트를 “실제입력”이라 칭함).

새롭게 제안한 병렬 이벤트구동 로직 시뮬레이션 기법에서는 각 로컬 시뮬레이션의 실행에 필요한 이와 같은 입력 이벤트로 실제입력 이벤트 아닌 자체적으로 예측한 입력 이벤트(본 논문에서는 이들 “예상입력”이라 칭함)를 우선적으로 사용하고, 예상입력이 실제입력과 다른 상황에서만 실제

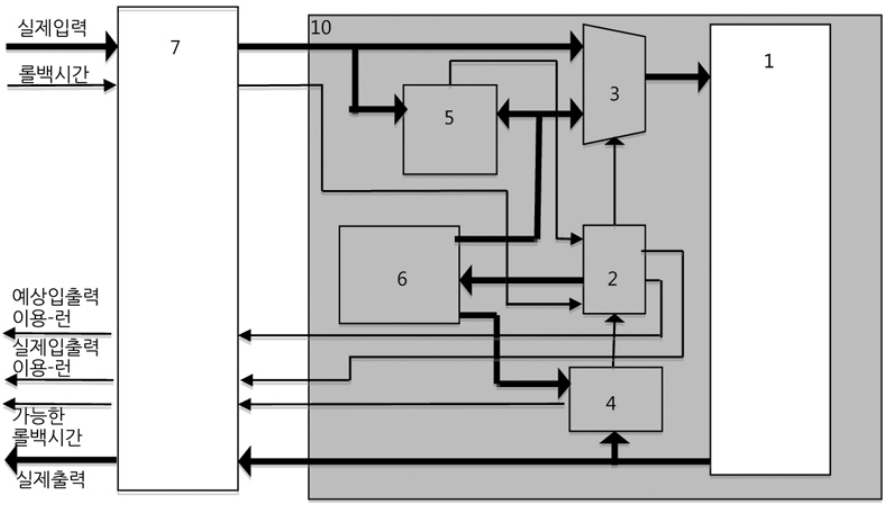

Fig. 1. Structure of local simulation in prediction-based parallel event-driven logic simulation

입력을 사용한다는 것이 핵심 아이디어이다. 여기서 주목할 것은 예상입력이 실제입력과 동일하다면 해당 시뮬레이션 시점에서 해당 로컬 시뮬레이션의 실행을 위해서는 다른 로컬 시뮬레이션과의 동기 및 통신이 전혀 필요하지 않다는 것이다. 따라서 만일 모든 로컬 시뮬레이션들에서 예상입력이 항상 또는 대부분 맞다면 병렬 시뮬레이션의 실행에서 각 로컬 시뮬레이션 간의 동기 오버헤드와 통신 오버헤드가 전혀 또는 거의 발생하지 않게 되며, 이는 병렬 시뮬레이션 성능 향상의 병목 요소가 완전히 또는 대부분 제거됨을 의미하므로 큰 시뮬레이션 성능 향상을 기대할 수 있게 된다. 물론, 예상입력이 항상 맞을 수는 없으므로 예상입력이 실제입력과 다른 경우도 본 논문에서 제안하는 예측기반 병렬 이벤트구동 로직 시뮬레이션 기법에서 고려되어야만 한다. 이를 위해서 우선 필요한 개념을 설명하기로 한다. 설계객체의 상태정보란 특정 시뮬레이션 시점에 설계객체에 존재하는 플립플롭출력이나 래치출력 혹은 메모리 혹은 궤환루프를 이루는 조합적 경로를 구술하고 있는 모든 변수나 시그널들의 값들을 말한다. 또한 설계객체의 입력정보란 시뮬레이션 전체 시간구간에서의 설계객체의 모든 입력(input) 및 입출력(inout) 상에서의 값들을 말하고, 설계객체의 출력정보란 시뮬레이션 전체 시간구간에서의 설계객체의 모든 출력(output) 및 입출력(inout) 상에서의 값들을 말한다.

본 논문에서 제안한 예측기반 병렬 이벤트구동 로직 시뮬레이션은 예상입출력 이용-런 모드와 실체입출력 이용-런 모드의 두 가지 실행 모드가 번갈아 가면서 진행되는데, 이 두 가지 실행 모드를 설명하면 다음과 같다. 예상입출력 이용-런 모드에서는 로컬 설계객체들 각각이 예상입력을 활용하여서 각 로컬 시뮬레이션을 독립적으로 실행시켜서 각 로컬 설계객체들의 출력에서 실제출력들을 구하게 되고, 동시에 이들 실제 출력값들 각각을 예상출력들과 시뮬레이션 진행 과정 중에 실시간으로 비교하면서 시뮬레이션을 진행한다. 그리고 이들(실제출력과 예상출력) 비교가 일치하게 된다면 로컬 시뮬레이터들 간에 통신 및 동기화를 완전히 생략하고 각 로컬 시뮬레이션 시간을 전진시킴으로써 시뮬레이션 속도를 크게 높이는 것이 가능하다. 그러나 만일 실제출력을 예상출력과 비교하여 이들이 일치하지 않으면(이 시뮬레이션 시점을 예상출력/실제출력 불일치시점이라 칭함), 이 예상출력/실제출력 불일치시점에서부터는 모든 로컬 시뮬레이터들 간에 통신 및 동기화과정을 수행하면서 실제입력을 활용하는 실제입출력 이용-런 모드로 분산병렬 시뮬레이션을 진행하게 된다. 즉, 이 시뮬레이션 시점에서부터는 로컬 시뮬레이터들 간에 통신 및 동기화 과정을 수행하게 되는 실제입출력 이용-런 모드로 실행 모드가 전환(이 전환이 이루어지는 시뮬레이션 시점을 실제입출력 이용-런 적용 시점이라고 칭함)된다. 그런데 특정한 1 이상의 로컬 시뮬레이션은 상기 예상출력/실제출력 불일치를 일으킨 로컬 시뮬레이션보다 빠르게 진행하여서 이들의 로컬 시뮬레이션 시간이 상기 예상출력/실제출력 불일치시점을 이미 지나쳐버린 상황에서는 이들의 로컬 시뮬레이션 시간을 되돌려서 상기 예상출력/실제출력 불일치시점으로 되돌린 후 재실행(restart after rollback)시킬 필요성이 있다. 이를 위하여 각 로컬 시뮬레이션들은 예상입출력 이용-런 모드로 진행하는 경우에는 일정 간격마다 체크포인트(checkpoint)를 만들면서 진행되어야 한다. 예상출력/실제출력 불일치가 발생하고 하나 이상의 로컬 시뮬레이션의 로컬 시뮬레이션 시간이 예상출력/실제출력 불일치시점을 넘어선 경우에는 상기 예상출력/실제출력 불일치시점에서 제일 가까운 체크포인트로 모든 로컬 시뮬레이션들이 롤백을 수행하고 난 후 상기 예상출력/실제출력 불일치 시점 직전까지 각 로컬 시뮬레이션들은 예상입출력 이용-런 모드로 진행한 후에 모든 로컬 시뮬레이션들의 로컬 시뮬레이션 시간을 예상출력/실제출력 불일치 시점으로 맞춘 후에 이 시점부터는 실제입력들을 사용하는(실제입력들은 특정 로컬 시뮬레이션의 실제출력임) 실제입출력 이용-런 모드로 모든 로컬 시뮬레이션들이 실행된다. 실제입출력 이용-런 모드에서 각 로컬 시뮬레이션은 다른 로컬 시뮬레이션들과 통신 및 동기화를 초래하는 시뮬레

이션을 진행하면서, 동시에 실제출력과 예상출력을 실시간으로 비교한다. 이 실제출력이 예상출력과 일정 횟수만큼 일치하는 상황이 모든 로컬 시뮬레이션들에서 일어나게 되면, 각 로컬 시뮬레이션들은 다시 예상입출력 이용-런 모드로 재전환하여 예상입력을 사용하여 동기 및 통신 과정 없이 시뮬레이션을 진행한다.

Fig. 1은 예측기반 병렬 이벤트구동 로직 시뮬레이션을 수행하는 각 로컬 시뮬레이션의 개략적인 구성도를 구조적으로 나타낸 것이다. 병렬 시뮬레이션환경을 구성하는 각 로컬 시뮬레이터에서 수행되는 로컬 시뮬레이션(10)은 로컬 설계객체(1)와 추가적인 컴포넌트들인 예상입출력 이용-런/실제입출력 이용-런제어모듈(2), 예상입력/실제입력선택모듈(3), 예상출력/실제출력비교모듈(4), 예상입력/실제입력비교모듈(5), 예상입력/예상출력 데이터생성/저장모듈(6)을 포함한다. 상기 각 컴포넌트의 동작 행태를 설명하면 다음과 같다. 예상입출력 이용-런/실제입출력 이용-런제어모듈(2)은 예상출력/실제출력비교모듈(4)과 예상입력/실제입력비교모듈(5)과 분산병렬 시뮬레이션을 위한 통신 및 동기화모듈(7)로부터 입력들을 받아서 이 입력들의 값들과 현재 로컬 시뮬레이션이 예상입출력 이용-런 방식으로 진행되는지 혹은 실제입출력 이용-런 방식으로 진행되는지에 따라서 예상입력/실제입력선택모듈(3)로 출력을 생성하여 예상입력/실제입력선택모듈(3)이 예상입력을 선택하게 하든지, 또는 실제입력을 선택하게 하든지, 또는 실제입력을 선택하기 전에 롤백이 필요하게 되면 롤백수행을 제어하게 된다. 즉, 현재 로컬 시뮬레이션이 예상입출력 이용-런모드로 진행하면서(따라서 예상입출력 이용-런/실제입출력 이용-런제어모듈(2)은 현재 예상입력/실제입력선택모듈(3)이 예상입력을 선택하도록 하는 출력을 예상입력/실제입력선택모듈(3)에 보내어 제어 중), 동시에 예상출력/실제출력비교모듈(4)에서부터 예상출력과 실제출력이 다르다는 판단이 입력으로 오게 되면 예상입출력 이용-런/실제입출력 이용-런제어모듈(2)은 예상입력/실제입력선택모듈(3)이 실제입력을 선택하도록 출력을 예상입력/실제입력선택모듈(3)에 보내고 내부의 현재 상태변수를 예상입출력 이용-런에서 실제입출력 이용-런으로 스위칭 하고 동시에 분산병렬 시뮬레이션을 위한 통신 및 동기화모듈(7)로부터 롤백을 위한 특정 롤백시간이 입력되면 상기 특정 롤백시간으로 롤백을 수행할 수 있게 제어도 수행하며, 현재 로컬 시뮬레이션이 실제입출력 이용-런모드로 진행되고 있으면서(따라서 예상입출력 이용-런/실제입출력 이용-런제어모듈(2)은 현재 예상입력/실제입력선택모듈(3)이 실제입력을 선택하도록 하는 출력을 예상입력/실제입력선택모듈(3)에 보내어 제어 중) 동시에 예상입력/실제입력비교모듈(4)에서부터 예상입력과 실제입력이 일정횟수 이상 같다는 판단이 입력으로 오게 되면 예상입출력 이용-런/실제입출력 이용-런제어모듈(2)은 예상입력/실제입력선택모듈(3)이 예상입력을 선택하도록 출력을 예상입력/실제입력선택모듈(3)에 보내고 동시에 내부의 현재 상태변수를 실제입출력 이용-런에서 예상입출력 이용-런으로 스위칭 하게 된다. 뿐만 아니라, 예상입출력 이용-런/실제입출력 이용-런제어모듈(2)은 내부의 현재 상태를 분산병렬 시뮬레이션을 위한 통신 및 동기화모듈(7)을 통하여 다른 로컬 시뮬레이션들에게 알리기 위한 두 개의 출력(실제입출력 이용-런 필요, 예상입출력 이용-런 가능)을 분산병렬 시뮬레이션을 위한 통신 및 동기화모듈(7)로 보내고, 예상입력/예상출력 데이터생성/저장모듈(6)을 제어하여서 예상입력/예상출력 데이터생성/저장모듈(6)을 맞는 예상입력 내지는 예상출력을 맞는 타이밍에 출력하도록 한다. 예상출력/실제출력비교모듈(4)은 예상입력/예상출력 데이터생성/저장모듈(6)에 저장된 예상출력과 로컬 설계객체(1)에서부터 로컬 시뮬레이션 실행과정에서 실제적으로 나오게 되는 실제출력을 비교하여, 일치하면 예상입출력 이용-런/실제입출력 이용-런제어모듈(2)에 일치함을 출력으로 보내고 불일치하면 예상입출력 이용-런/실제입출력 이용-런제어모듈(2)에 불일치함을 출력으로 보내는 동시에 분산병렬 시뮬레이션을 위한 통신 및 동기화모듈(7)로 롤백을 위한 현재의 시뮬레이션 시간을 보내어 다른 로컬 시뮬레이션들에게 이들 정보(롤백을 위한 현재의 시뮬레이션 시간)를 보낼 수 있게 한다. 예상입력/실제입력비교모듈(5)은 예상입력/예상출력 데이터생성/저장모듈(6)에 저장된 예상입력과 분산병렬 시뮬레이션을 위한 통신 및 동기화모듈(7)로부터 오는 다른 1 이상의 로컬 시뮬레이션으로부터의 실제입력을 비교하여 비교가 일정횟수만큼 일치하면 이를 예상입출력 이용-런/실제입출력 이용-런제어모듈(2)에 출력으로 보내게 된다. 예상입력/실제입력선택모듈(3)은 예상입출력 이용-런/실제입출력 이용-런제어모듈(2)의 출력으로써 통신 및 동기화모듈(7)로부터 오는 실제입력과 예상입력/예상출력 데이터생성/저장모듈(6)에 저장된 예상입력 중에서 하나를 선택하여서 이 로컬 시뮬레이터에서 로컬 설계객체(1)에 입력으로 인가한다.

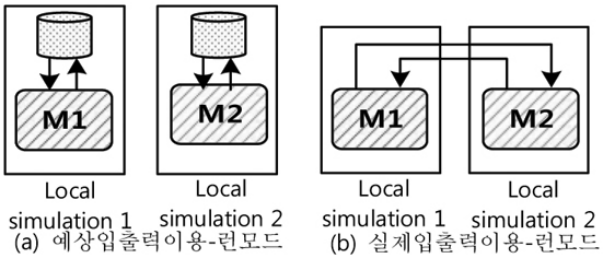

Fig. 2. Two Execution Modes of Proposed Method

Fig. 2는 Fig. 1과 같은 구조를 가지는 2개의 로컬 시뮬레이션들이 각각 예상입출력 이용-런모드(통신/동기 오버헤드 없음)와 실제입출력 이용-런모드(통신/동기 오버헤드 존재)로 실행되는 상황을 도식적으로 예시한 그림이다.

3.2 예측정확도 및 설계플로우에서 적용시점

앞에서 설명한 대로, 예측기반 병렬 이벤트구동 로직 시뮬레이션은 예측이 맞는 경우에는 로컬 시뮬레이션들 간에 동기 및 통신을 완전히 제거하고 로컬 시뮬레이션들이 완벽

Table 1. Experimental result

디자인명	케이스 유형	순차적 시뮬레이션 실행시간(초)	본 논문의 예측기반 병렬시뮬레이션			성능향상비
			실행시간(초)	코어 수	예측정확도(%)	
JPEG	2	4,249	1,573	3	100	2.70
AC97	2	169	52	4	100	3.25
AES	1	3,521	981	4	100	3.59
serial_atpg1	2	9,401	9,145	4	80	1.03
serial_atpg2	2	10,896	1,905	5	99	5.1
mobile_ap	2	16,349	2,125	8	99	7.69

히 독립적으로 시뮬레이션을 병렬적으로 실행하게 되므로 병렬화에 의한 높은 시뮬레이션 성능 향상을 기대할 수 있다. 따라서 본 논문에서 제안한 예측기반 병렬 이벤트구동 로직 시뮬레이션의 핵심은 “어떤 방법을 통하여 병렬 시뮬레이션 실행 과정에서 높은 예측정확도를 계속 유지할 수 있는가?”이다.

현재 SOC 등과 같은 비메모리반도체 설계는 Verilog 또는 System Verilog와 같은 하드웨어구술언어를 사용하여 레지스터전송수준 내지는 시스템수준에서 설계하여 게이트수준을 거쳐서 레이아웃수준까지 진행되는데 이 여러 추상화 수준 단계들에서 로직 시뮬레이션들이 반복적으로 실행되게 된다. 따라서 특정 로직 시뮬레이션 실행(예로 게이트-수준에서 타이밍 시뮬레이션을 실행)을 예측기반 병렬 시뮬레이션으로 실행하고자 하는 경우에 사용할 수 있는 예측데이터(로컬 시뮬레이션들 각각의 실행에 필요한 예상입력 시퀀스와 예상출력 시퀀스)는 바로 이전 시뮬레이션 실행(이 이전 시뮬레이션 실행도 예측기반 병렬 시뮬레이션으로 실행될 수 있음) 과정에서 상기 로컬 시뮬레이션들 각각에서 시뮬레이션되는 로컬 설계객체들 각각의 입력정보와 출력정보를 저장한 것인데 이 두 시뮬레이션들 사이에는 매우 높은 유사성이 존재하므로 예측데이터는 대부분의 경우에 매우 높은 예측정확도를 가지게 될 것이다. 이를 좀 더 자세히 보면, 설계 과정에서 특정 시뮬레이션 실행이 반복되는 경우는 다음과 같은 3가지 케이스로 나누어진다.

- 케이스 1 : 추상화 수준이 달라져서 특정 시뮬레이션 실행을 반복하는 경우(예 : 레지스터전송수준에서 레지스터전송수준 디자인에 대한 특정 시뮬레이션 실행을 한 후에 논리합성 과정을 통하여 얻어진 게이트수준 디자인에 대하여 게이트수준에서 동일한 특정 시뮬레이션을 실행하는 경우)
- 케이스 2 : 동일한 추상화 수준에서 디자인의 일부가 변경된 후에 특정 시뮬레이션 실행을 반복하는 경우(예 : 게이트수준에서 디자인에 대한 타이밍 시뮬레이션을 실행하고, 디자인의 일부분에 ECO(Engineering Change Order) 변경을 한 후에 ECO 변경이 바르게 이루어졌는지를 확인하기 위하여 변경된 디자인에 대한 타이밍 시뮬레이션을 실행하는 경우)
- 케이스 3 : 동일한 추상화 수준에서 동일한 디자인에 대한 시뮬레이션 재실행

케이스 3에서는 당연히 항상 100%의 예측정확도를 얻을 수 있고, 케이스 1의 경우에서도 100%에 거의 근접한 예측정확도를 기대할 수 있는데, 그 이유는 일반적으로 게이트수준의 네트리스트는 논리합성 과정을 통하여 레지스터전송수준의 디자인 구술로부터 완전 자동화된 방식으로 생성되기 때문에 게이트수준의 디자인과 레지스터전송수준의 디자인은 함수적으로 등가이어서 특정 테스트벤치를 통한 특정한 시뮬레이션 시나리오하에서 레지스터전송수준의 디자인과 게이트수준의 디자인은 함수적으로 동일한 행태(behavior)를 보이기 때문이다. 케이스 2는 디자인 검증 과정에서 설계오류가 발견되어서 디자인이 수정되는 상황이다. 그런데 일반적으로 설계오류를 바로잡기 위한 설계 변경은 디자인의 행태를 전반적으로 변화시키기보다는 국지적으로 변화시키는 경우가 대부분이므로 케이스 2에서도 디자인의 일부가 국지적으로 변경된 경우, 설계 변경 전의 시뮬레이션 실행 과정에서 얻은 예측데이터의 예측정확도 역시 상당히 높을 것으로 예상된다. 이들 각 케이스들(케이스 3은 당연히 예측정확도가 100%이므로 제외)에 대해서는 다음 절의 실험을 통해서 좀 더 자세히 논의하기로 한다.

4. 실험 결과

본 논문에서 제안된 새로운 예측기반 병렬 이벤트구동 로직 시뮬레이션을 통한 시뮬레이션 성능 향상의 정도를 실험적으로 확인하기 위하여 다수의 디자인들을 대상으로 하는 실험을 진행하였다. 실험을 위하여 사용된 이벤트구동 HDL 시뮬레이터는 Cadence 사의 Verilog 시뮬레이터인 IUS이다. IUS는 이미 다양한 최적화 기술들이 적용되어진 Synopsys 사의 VCS와 더불어 현존하는 제일 빠른 시뮬레이터이다. 표 1에서는 총 6개의 디자인들에 대하여 순차적 시뮬레이션과 본 논문에서 제안한 예측기반 병렬 시뮬레이션을 각각 진행하여 각각의 시뮬레이션 실행시간 및 병렬 시뮬레이션에서 사용한 프로세서 코어수(로컬시뮬레이션 총수) 및 예측정확도를 측정하고,최종적으로 본 논문의 방법을 통하여 시뮬레이션의 성능향상비(세번째컬럼/네번째컬럼)를 구하였다. 6개의 디자인들중 serial_atpg1, serial_atpg2, mobile_ap는 실제 산업체에서 설계된 대규모 디자인들이다. 표 1에서

두번째 컬럼(케이스유형)은 각 사례가 논문 제 3장의 예측 데이터가 확보되는 이전 시뮬레이션의 유형 세가지 케이스 중에 어느 케이스에 해당하는지를 표시한 것이다.

표 1에서 보는것과 같이 모든사례들에서 예측기반 병렬 시뮬레이션을 통한 시뮬레이션 성능 향상이 관찰되었다.특히 serial_atpg2의 경우에는 코어를 5개를 사용하였음에도 성능향상은 5.1 배가 되었는데, 이는 거의 100%에 근접하는 높은 예측정확도로 시뮬레이션 시간의 대부분을 동기 오버헤드 및 통신 오버헤드가 완전히 제거되어져서 5개의 로컬 시뮬레이션들이 독립적으로 실행되는 상황에서 각 로컬설계 객체들의 크기가 전체 디자인보다 작음으로서 각 로컬시뮬레이션 실행 과정에서 트레싱(thrashing)의 발생빈도가 낮아졌기 때문으로 짐작되며, 이 결과가 본 기법의 장점을 극명하게 보여주는 사례의 하나이다. Serial_atpg1의 경우에는 성능향상이 3% 밖에 되지 않는데, 이는 다른 사례들에 비하여 상대적으로 낮은 예측정확도에서 추론할 수 있듯이 설계 변경이 디자인의 행태를 많이 변하게 함으로 많은 횟수의 롤백과 재실행 때문이다.

5. 결 론

본 논문에서는 예측기반의 병렬 이벤트구동 로직 시뮬레이션을 새롭게 제안하였다. 제안된 기법은 다른 로컬 시뮬레이션와의 동기 및 통신을 수반하는 실제입력을 사용하는 것에 앞서 해당 로컬 시뮬레이션이 자체적으로 가지고 있는 예상입력을 우선적으로 사용하여 로컬 시뮬레이션을 다른 로컬 시뮬레이션들과의 동기 및 통신을 배제하고 실행하고, 예상입력이 틀린 상황에서만 동기 및 통신을 통하여 다른 로컬 시뮬레이션에서 제공되는 실제입력을 사용하여서 로컬 시뮬레이션이 실행될 수 있도록 함으로써 병렬 이벤트구동 로직 시뮬레이션에서 핵심 병목으로 성능 향상을 제약하는 요소는 동기 오버헤드 및 통신 오버헤드 모두를 효과적으로 최소화시키는 것이 가능하여 병렬화를 통한 의미 있는 성능 향상이 가능하다. 제안된 새로운 병렬 이벤트구동 로직 시뮬레이션 방법은 실체 산업체에서 설계된 대규모 다자인들을 포함한 다수의 디자인들을 대상으로 한 실험을 통하여 시뮬레이션 성능 향상에 매우 효과적임이 확인되었다. 향후에는 설계변경이 디자인의 행태를 상당한 폭으로 변경하는 경우(예측정확도가 상대적으로 낮은 경우)에서도 높은 시뮬레이션 성능을 유지하는 것에 대한 추가적인 연구가 필요하다.

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," 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, 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," Proceedings 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, C. Browy, "Parallel Logic Simulation: Myth or Reality?," *Computer*, Vol.45, No.4, pp.67-73, Apr., 2012.

[9] D. Chatterjee, A. DeOrio, and V. Bertacco, "High-performance gate-level simulation with GP-GPUs," Proceedings of the Conference on Design Automation and Test in Europe (DATE'09), pp.1332-1339, Apr., 2009.

[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] Nanjundappa, M., Patel, H., Jose, B. A., and Shukla, S., "SCGPSim: A Fast SystemC Simulator on GPUs," In Proceedings of the 15th Asia and South Pacific Design Automation Conference(ASPDAC'10), pp.145-154, 2010.

양 세 양

e-mail : syyang@pusan.ac.kr

1981년 고려대학교 전자공학과(학사)

1985년 고려대학교 컴퓨터공학과 (공학석사)

1990년 University of Massachusetts, Amherst 컴퓨터공학과(공학박사)

현 재 부산대학교 정보컴퓨터공학부 교수

관심분야 : 설계자동화, 특히SoC검증