Synopsys Milkyway Database환경에서 구현된 Post-layout Gate-Level Logic Simulator

김명진[1], 방성용[1], 정의영[1], 윤성로[2]
[1]연세대학교 전기전자공학부, [2]고려대학교 전기전자전파공학부
전화: (02)3290-4826, E-mail: sryoon@korea.ac.kr

Post-layout Gate-Level Logic Simulator Implemented in the Synopsys Milkyway Database Environment

Myeong-Jin Kim[1], Sung-Yong Bang[1], Eui-Young Chung[1], Sungroh Yoon[2]
[1]School of Electrical&Electronic Engineering, Yonsei University
[2]School of Electrical Engineering, Korea University

요 약

본 논문에서는 세계적으로 가장 많이 사용되는 EDA(Electronic Design Automation) Design Platform중의 하나인 Synopsys Milkyway Database(Galaxy)상에서 IEEE 표준 언어인 스킴(Scheme) 프로그램 언어를 사용하여 구현된 게이트 수준의 논리 시뮬레이터를 제안한다. Synopsys Milkyway Database는 지금까지 주로 Physical Design 환경으로 사용되거나, Verification의 경우도 주로 STA(Static Timing Analysis)정도로만 적용되어 왔다. 본 논문에서는 제시하는 시뮬레이터는 STA환경에서 자동으로 디자인을 읽고 사용된 로직의 모델을 생성하여 시뮬레이션 수행한다. 일반적으로 이벤트를 발생시간에 따라 스케줄링을 하기위해 선형 배열 형태의 스케줄러를 사용한다. 스케줄러는 발생시간에 따른 이벤트를 정렬하여 처리하는 속도를 높이기 위해 세밀한 시간분해능을 가져야한다. 그런데 이벤트 발생 빈도가 높으면 선형 배열이 효율적이지만, 반대로 이벤트 발생 빈도가 낮은 경우에는, 메모리 사용 효율이나 이벤트를 탐색하여 처리하는 속도가 상대적으로 떨어지게 된다. 이에 대해 Design Capacity를 고려하여 주기적으로 이벤트 예측을 통하여 이벤트 발생빈도가 낮은 경우에 인접 이벤트 정보를 이용하여 시뮬레이션 속도를 향상시킬 수 있었다.

Abstract

This paper presents an adaptively structured event scheduling gate-level logic simulator. This simulator, which was implemented with the Scheme programming language in the Synopsys Milkyway database environment, can generate simulation models and construct design automatically based upon the information extracted from the Milkyway database. Generally, logic simulators have an array-style event scheduler in order to speed up the scheduling time. This linear array style scheduler is normally required to have a fine timing resolution for event-sorting and works well for a high density event scheduling task. However, this type of scheduler tends to be inefficient when scanning an area of no event or a low-density area. In such cases, using the next event's information is helpful because we can skip empty time slots and reduce the simulation time. However, without a smart event estimation technique, the overhead of getting the next event's information will be too high. In this paper, we propose an estimation based adaptively structured scheduler, which can reduce the simulation time substantially, without having too much overhead.

Keywords : Gate level logic simulator, STA, Synopsys, scheme program language

Ⅰ. 서 론

Logic Simulator는 Design의 오류를 찾아내거나 정확성을 검증하는 기본적인 도구이다. 그런데 Design의 규모나 복잡도가 증가함에 따라 다양한 수준에서 검증 방법들이 연구되어 왔다. 검증의 대상도 Abstraction

Level에 따라 Behavioral (Algorithmic) Level, RTL (Register Transfer Level), Gate Level등으로 나눌 수 있고, 이에 따른 Simulation의 형태도 Behavior Simulation, Cycle-based Simulation, Event-driven Simulation등으로 나눌 수 있다.[1] 그리고 Simulation의 검증 대상과 목적에 따라 Functional Simulation, Pre-layout Simulation, Post-layout Simulation으로도 나눌 수 있다. 본 논문에서는 주로 Post-layout Gate Level Event driven Simulation에 대해서 논한다.

Post-layout simulation은 Physical Design이 완성된 후 할 수 있는 단계로써, 대부분의 Design Sign-off Flow에 포함되며, IC가 만들어진 상태에 가장 가까운 Simulation 결과를 얻기 위해 Physical Design의 정확한 Timing 정보로부터 Simulation이 수행된다. 그런데, 공정 기술의 발전으로 집적화와 미세화에 따른 Yield 향상을 위한 DFM (Design For Manufacturability)을 고려하게 되고,[2] 휴대기기 증가로 인한 LOP (Low Operation Power)와 LSTP (Low Stand-by Power)의 적용으로 인한 설계 및 검증 Flow의 복잡화와 Quality측면에서 Guard band 확보가 점점 어려워지고 있다. 이에 따라 검증을 위해 Physical Design으로부터 추출된 Data의 양도 점점 증가하고, 전체 Design Cycle에서 검증시간도 상대적으로 점점 길어지고 있다.

이와 같은 Design의 추세에서, 본 논문에서 제안하는 Simulator는 IEEE 표준인 Scheme 프로그래밍 언어[3]를 사용하여 Synopsys Milkyway Database 상에서 구현되어 기존의 상용 Tool과 Interface가 용이하며, Logical Library 환경과 Physical Database를 직접 읽어 Simulation Model을 생성하고 정확한 Timing 정보를 통해 Simulation을 수행한다. Simulation 수행시 각각의 Event에 대해 보다 효율적인 Simulation을 진행하기 위해 Design의 Capacity를 고려하여 Event의 발생 빈도를 예측하여 보다 효율적인 스케줄러를 적용하였다.

Ⅱ. 본 론

1. 제안하는 Simulator의 Data Flow

제안하는 Post-layout Gate Level Simulator는 그림1과 같이 Synopsys Galaxy Design Platform (Milkyway Database)상에서 구현되었기 때문에 기존의 상용 Tool들과 같이 Milkyway Database를 공유함으로써 Tool간의 Data의 입출력을 최소화할 수 있을 뿐 아니라 Voltage Drop이나 Crosstalk과 같은 Physical 영역에서 해석되어지는 현상을 Simulation시에 반영이 용이해진다.

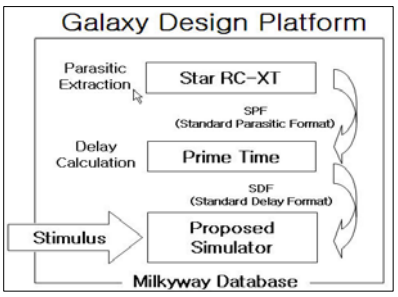

그림 1. Galaxy Design Plaform상의 Data Flow
Fig. 1. Data Flow on Galaxy Design Platform

2. Logic Level과 Simulation Model

Simulation Logic Level은 Scheme Language에 맞게 3가지의 Value(0,1,#f)를 사용한다. 여기서 #f는 unknown 'X' 값에 해당한다.

Simulation Model은 그림2와 같은 Scheme으로 기술된 Not, And, Or, Xor의 네 개의 기본 Function을 기초로 각각의 Gate들의 Function이 Modeling된다.

```
(define Not
 (lambda X
  (begin
   (cond
     ((null? X) #f)
     ((equal? 0 (car X)) 1)
     ((equal? 1 (car X)) 0)
     (else #f)))))

(define And
 (lambda X
  (begin
   (cond
     ((null? X) 1)
     ((equal? 0 (car X)) 0)
     ((equal? 1 (car X)) (apply And (cdr X)))
     ((and (equal? #f (car X))
       (not (boolean? (member 0 (cdr X))))) 0)
     (else #f)))))

(define Or
 (lambda X
  (begin
   (cond
     ((null? X) 0)
     ((equal? 1 (car X)) 1)
     ((equal? 0 (car X)) (apply Or (cdr X)))
     ((and (equal? #f (car X))
       (not (boolean? (member 1 (cdr X))))) 1)
     (else #f)))))

(define Xor
 (lambda X
  (begin
   (cond
     ((null? X) 0)
     ((not (boolean? (member #f X))) #f)
     ((odd? (apply + X)) 1)
     ((even? (apply + X)) 0)))))
```

그림 2. 스킴으로 기술된 Simulation Model
Fig. 2. Simulation Model described in Scheme

3. Simulator의 Database 구조

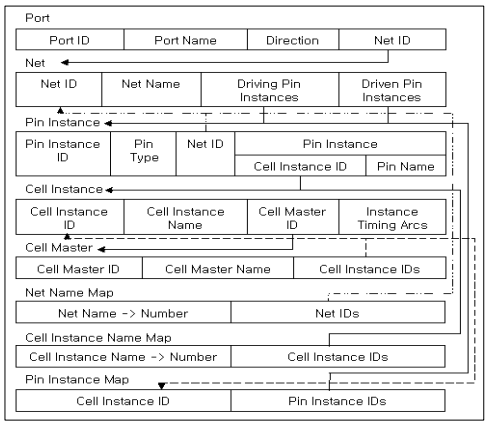

그림 3. Simulator의 Database 구조
Fig. 3. Database of Simulator

Simulator의 Database구조는 그림3과 같이 Port, Net, Pin Instance, Cell Instance, Cell Master의 기본 Data와 Timing Annotation을 위한 Net Name Map, Cell Instance Name Map, Pin Instance Map으로 구성된다.

4. Simulation Flow

Simulation의 Flow는 아래 6단계로 모두 자동으로 수행되어진다.

가. Library Setup 단계

Design이 완성된 Library와 Simulation Design명을 주면 Design에 사용된 Library정보, Library에 포함된 Cell 정보와 Timing 정보를 자동으로 추출하는 단계이다.

나. Function Setup 단계

Library에 Modeling된 Cell중 Design에 사용된 Cell들에 대해 Function을 Setup하는 단계로서 Infix notation "(!a & !b & c)"의 function model을 Scheme 언어에 맞게 Prefix notation "(And (Not a) (And (Not b) c))"으로 변형하고 Cascaded된 동일 function에 대해 logic collapsing[4] "(And (Not b) (Not b) c)"을 통해 간략화 시켜주는 단계이다.

다. Design Setup 단계

Simulation을 위해 Milkyway Database를 읽는 단계로 "Cell Master -> Cell Instance -> Pin Instance -> Net -> Port" 순으로 읽은 후 끝으로 Net Connection을 완성한다.

라. Timing Setup 단계

그림1과 같이 Galaxy Platform상에서 Star RC-XT로부터 Milkyway Design Data에 대한 정확한 SPF (Standard Parasitic File)를 추출하고 PrimeTime에서 Delay Calculation을 한 후 SDF (Standard Delay Format)을 생성한다. 생성된 SDF를 Simulator가 읽어서 그림3에 있는 Cell Instance Name Map에서 Cell Instance를 찾아서 Instance Timing Arc에 정보를 저장하게 된다.

마. Constraint Setup 단계

Timing Closure에 사용된 SDC (Synopsys Design Constraint)들과 Clock과 Primary Port들에 대한 정보로부터 Simulation 환경을 구성한다.

바. Simulation 단계

Input Stimulus와 각 Register에서 Clock의 Synchronization Time에 따라 Simulation이 진행된다. 알고리즘은 그림4와 같다. 이때 사용되어지는 Scheduler 의 구조는 두 가지 형태를 가진다. 기존의 단순한 Array형태와 Next Event에 대한 Index를 갖는 형태이다.

```
Initialize design status;
Initialize the timers for clock, event and stimulus;
while ( event timer <= target simulation time ) {

  while ( stimulus timer = event timer ) {
    foreach stimulus at current time on the stimulus scheduler {
      if ( stimulus is different from the status of port ) {
        update pin instance status table;
        foreach pin instance connected to stimulus port {
          if ( driven pin instance is storage element pin ) {
            schedule pin instance w.r.t. interconnect delay to
            clock pin event wheel; }
          else {
            schedule pin instance w.r.t. interconnect delay to
            logic pin event wheel; }}}}
    update stimulus timer to the next scheduled stimulus;
    }

  foreach clock pin events on current time {
    if ( scheduled pin is clock ) {
      operate storage element functions;
      if ( output pin status is changed ) {
        schedule propagated events; }
      update pin status table; }
    elseif ( scheduled pin event is different from pin instance
             status ) {
             update pin instance table; }}

  foreach logic pin events on current time {
    if ( scheduled pin event is different from pin instance
         status ) {
      operate cell instance master functions;
      if ( output pin status is changed ) {
        schedule propagated events; }
      update pin instance status table; }}

  increase event timer;
}
```

그림 4. Simulation 알고리즘
Fig. 4. Simulation Algorithm

Next Event에 대한 Index를 만드는 것은 Overhead가 있기 때문에 발생 이벤트가 적을 경우에 효과가 있다. 아래 그림5는 Array형태(a)와 Array에 Next Event의 Index(b)를 갖는 경우의 수행시간을 측정한 것이다.

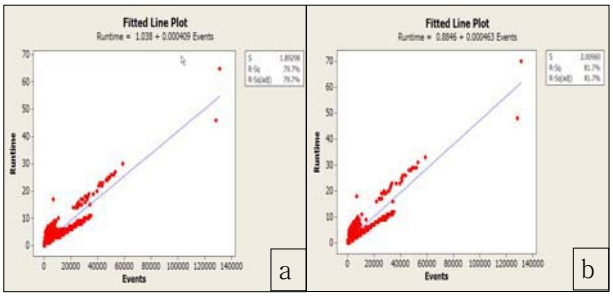

그림 5. Arrayed와 Next Event Indexed Runtime 비교
Fig. 5. Runtime Comparison (Arrayed(a) vs. Arrayed + Next Indexed(b))

$$Run\,Time(a) = 1.03787 + 0.00040935 * event\ 수 \qquad (1)$$

$$Run\,Time(b) = 0.88464 + 0.00046321 * event\ 수 \qquad (2)$$

두 형태의 수행시간을 회귀분석을 통해 얻은 식(1)과 식(2)를 비교할 때, Next Event Index 방식은 Overhead를 고려할 때 이벤트가 2800이하로 낮은 경우에 수행시간이 짧은 것을 알 수 있었다.

Ⅲ. 실 험

실험은 표1의 ISCAS 99 Benchmark 회로 중 9개 Design을 0.25um Library를 사용하여 Synopsys Design Compiler에서 합성 후, Synopsys Astro에서 P&R(Placement&Routing)후 Timing Optimization을 수행하여 최적화된 Operating Frequency를 얻었다.

표 1. ISCAS 99 Circuit의 Optimization 결과
Table1. ISCAS 99 Circuit Optimization Result

Design	Gate Count	Speed	Net Count
B19	198352	80MHz	106415
B18	95422	90MHz	49591
B22	34068	100MHz	23112
B17	26573	100MHz	12199
B14	11194	100MHz	7593
B12	1517	250MHz	596
B10	225	330MHz	109

Timing Closure가 끝난 Database를 바탕으로 표2와 같이 3가지 형태로 Simulation이 진행되었다. Type A는 기존의 Array 방식의 Scheduling 방식이고, Type N은 Next Event Indexed Array Scheduling 방식이고, Type A+N는 Primary Input Port와 Register Output Pin의 Event를 기준으로 전체 Event를 예측하여 Event가 많이 발생될 것으로 예측될 경우에는 Type A방식을 쓰고, 반대로 Event가 적을 것으로 예상될 때는 Type N방식으로 변경시켜서 사용하는 방법이다. 그리고 Event 예측은 Design의 전체 Net수를 Primary Port수와 Register Output 수로 각각 1:9로 나눈 후 각 주기별로 Primary Port와 Register Output의 변화를 근거로 Event를 예측하였다.

실험 결과는 그림6과 표2와 같이 이벤트수를 2800 기준으로 High인지 Low인지를 예측할 때 70%이상 예측이 가능했고 Simulation Time도 3~80%를 감소시킬 수 있었다.

표 2. ISCAS 99 Circuit의 Simulation 결과
Table2. ISCAS 99 Circuit Simulation Result

	Type A(sec.)	Type N (sec.)	Type A+N (sec.)	Est H/L Hit Rate	Ruduced Time Rate
B19	1874	1951	1812	81.5%	3.3%
B18	790	923	773	84.5%	2.1%
B22	1046	1135	1009	89%	3.5%
B17	331	281	281	99%	15.1%
B14	496	524	474	73%	4.5%
B12	68	33	33	100%	51.5%
B10	45	8	8	100%	82.3%

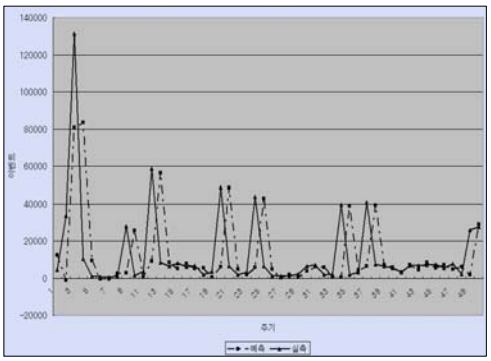

그림 6. B19의 Event 예측과 Simulation 결과 비교
Fig. 6. Number of Events Comparison between Estimation and Simulation of B19

Ⅳ. 결 론

본 논문에서는 Milkyway Database 환경에서 Simulator를 구현하여 Database를 공유함으로써 기존 상용 Tool과의 Interface를 용이하게 했으며, Event 예측을 통해 보다 효과적인 Scheduler로 성능 개선을 통해 3~80% 정도의 Simulation Time을 감소시켰다.

후 기

본 논문은 지식경제부가 지원하는 국가 반도체 연구개발사업인 “시스템 집적 반도체 기반 기술 개발 사업 (시스템 IC 2010)"을 통해 개발된 결과임을 밝힙니다.

참 고 문 헌

[1] L. Scheffer, L. Lavagno and G. Martin, "*EDA for IC System Design, Verification, and Testing*," CRC Press, 16-2, 2006.

[2] K. Riviere-Cazaux, K. Lucas and J. Fitch, "Integration Of Design For Manufacturability (DFM) Practices In Design Flows," *in Proc. of the Sixth International Symposium on Quality Electronic Design*, pp. 102-106, March 2005.

[3] "*IEEE standard for the scheme programming language,"* IEEE Std pp. 1178-1990.

[4] P. Maurer, "The Inversion Algorithm for Digital Simulation," *IEEE Transactions on Computer Aided Design, Vol. 16, No.7*, pp. 762-769, July 1997.