고성능 로직 시뮬레이터(HSIM) 구현

박 장 현[†] 이 기 준[††] 김 보 관[†††]

요 약

본 논문에서는 함수 기능에서 로직 게이트 기능까지 시뮬레이션 가능한 고성능의 로직 시뮬레이터(HSIM) 개발에 대해서 논한다. 개발된 로직 시뮬레이터는 입력부, 시뮬레이터 본체, 출력부로 구성되어 있으며, 입력부에는 네트 리스트 컴파일러, 부품 정보 컴파일러, 부품 기능 컴파일러가 포함된다. 시뮬레이터 본체에는 시뮬레이션 속도를 높이기 위한 각종 기술과 시뮬레이터의 중심 부분인 시뮬레이션 엔진 등이 소속되어 있다. 출력부에는 시뮬레이션 결과를 분석하는 파형 분석기가 있다. 개발된 시뮬레이터 본체의 주요 특징은 점진적 로더를 사용하여 컴파일된 부품 기능들을 시뮬레이션 엔진에서 직접 로드하여 시뮬레이션을 수행한다. 이렇게 한 결과 기존의 유닛 딜레이 event-driven interpretive 시뮬레이터와 비교했을 때 55% 이상 속도가 빠른 효과적인 성능 향상을 달성했다.

HSIM: Implementation of the Highly Efficient Logic SIMulator

Jang-Hyun Park,[†] Ki-Jun Lee,[††] Bo-Gwan Kim[†††]

ABSTRACT

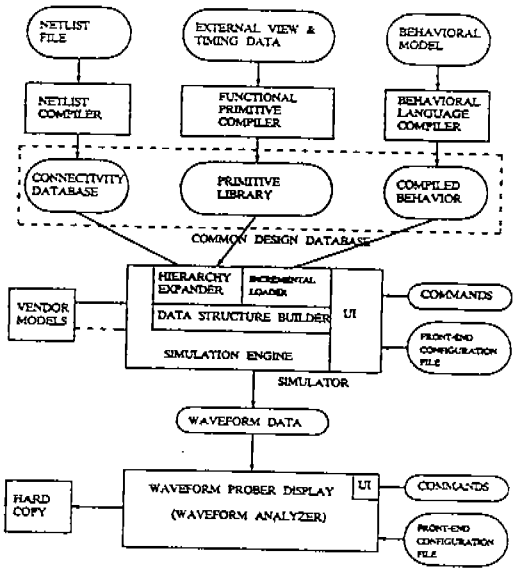
In this paper, we present a highly efficient simulation package which supports simulation from functional level to gate level. The package consists of a set of front-end tools, a logic simulator, named HSIM(Highly efficient logic SIMulator), and an waveform analyzer. The front-end tools include a netlist compiler, functional primitive compiler and behavioral compiler. Key feature of developed simulator is that the compiled behavioral models written in C language are directly executed in the simulation engine using incremental loader. By doing so, we achieved significant speed up as compared with the interpretive functional simulator. Experimental results show that HSIM runs about 55% faster than traditional unit-delay event-driven interpretive simulator.

1. 서 론

설계검증용 CAE 도구란 하드웨어 구성시 실제로 시제품을 제작하지 않고 설계된 하드웨어가 목적하는 대로 동작하는가를 검증할 수 있는 소프트웨어이다. 생산성 향상 및 경비절감이 관건인 경쟁적인 현상황에서 보다 효율적인 설계검증을 위해 이러한 소프트웨어를 이용하는 방법이 보편화된 실태이다. 본 시스템의 개발 목적은 하드웨어 설계자가 시스템 구성시 효과적으로 사용할 수 있게 최적화된 설계검증용 소프트웨어 일체를 구축하는 데 있다. Event-driven 시뮬레이션 기술[1]은 수년동안 여러형태의 시뮬레이터를 만들면서 응용되어지고 있다. 이 알고리즘의 가장 특징적인 것은 asynchronous 회로와 타이밍 분석을 쉽게 할 수 있는 selective-trace 접근방식이다. 과거 20여년동안 event-driven 시뮬레이션의 속도[2]를 증가시키기 위하여 많은 노력을 했지만 아직도 성능은 주요 문제로 남아 있다. 종래의 event-driven 알고리즘의 성능[3]을 향상시키기 위하여 여러가지 방법을 사용하여 왔

† 정 회 원 : 한국전자통신연구소 선임연구원
†† 정 회 원 : 충남대학교 전자공학과 부교수
††† 정 회 원 : 충남대학교 전자공학과 부교수
논문접수 : 1995년 3월 13일, 심사완료 : 1995년 7월 5일

다. Levelized event-driven 알고리즘을 사용한 LECSIM[4], demand-driven 알고리즘을 사용한 BACKSIM[5] 등이 그러한 것이다.



(그림 1) 시스템 구조도
(Fig. 1) System Architecture

그러나 아직도 event-driven 시뮬레이션 알고리즘의 성능 향상이 큰 문제점이다. 따라서 본 논문에서는 시뮬레이션 본체에 점진적 로더를 이용하여 event-driven 시뮬레이션의 성능을 향상시키며, in-house 시뮬레이터 특성을 최대로 살려 각 회사나 연구소의 고유 시스템 개발에 응용하고자 한다. (그림 1)에 기본적인 시스템 구조를 보여주고 있다. 본 시스템의 새로운 아이디어는 첫째 각 부품 모델에 대해서 behavioral 정보와 primitive 정보를 분리하여 모델을 구성한 것이다. 기존의 모델들은 한 장소에 모든 정보를 수용하므로 사용자가 모델 라이브러리를 관리하기가 매우 어려웠다. 모델 정보를 분리함으로써 단순 로직 개발자에게 불필요한 정보는 모델 라이브러리에서 제거할 수 있고, ASIC 개발자등이 필요한 부품의 상세한 정보를 따로 관리하므로 도구의 효용성이 증가하고 사용이 용이하다. 둘째로는 분리하여 구성된 정보를 한꺼번에 로드하는 것이 아니라 필요시 점진적 로더를 사용하여 모델을 로딩하는 것이다. 이렇게 함으로써 사전에 수행해야할 여러 단계를 한꺼번에 처리 가능하며 성능을 향상할 수 있다 . (그림 1)의 시스템 구조에서 보듯이 시스템의 주요 특징을 살리기 위해 본 시스템은 front-end 도구와 기능 수준 시뮬레이터와 파형 분석기로 구성되어 있으며, 입력 데이타는 네트 리스트 화일, PDL (Primitive Description Language) 화일, behavioral 모델 등 세 부분으로 나누어 진다. 입력부 도구들은 세 종류의 입력 데이터를 컴파일하는 도구로서, 네트 리스트 컴파일러, PDL 컴파일러, behavioral 컴파일러 및 여러개의 커맨드 번역기로 구분된다. 개발된 시뮬레이터 본체의 주요 특징은 점진적 로더를 사용하여 컴파일된 부품 기능들을 시뮬레이션 엔진에서 직접 로드하여 수행한다. 이렇게 한 결과 기존의 해석적인 기능 시뮬레이터와 비교했을 때 효과적인 성능 향상을 달성했다. 시뮬레이션 결과를 분석하는 툴로서 개발된 파형 분석기는 파형을 보여주는 것 외에 분석을 편리하게 하는 여러 가지 기능을 갖고 있다. 주요기능으로는 파형의 시간 측정 및 논리값의 측정, 다른 시뮬레이션 결과와의 비교 분석, 화면에 나타난 파형의 목록을 편집하는 기능, 그리고 하드 카피 기능 등이 있다.

본 논문은 서론에 이어 2장에서는 네트 리스트 컴파일러, behavioral 컴파일러, PDL 컴파일러 등 입력부 도구들에 대해서 자세히 설명하고, 3장에서는 시뮬레이션 본체부분의 계층적 확장기, 점진적 로더, 데이터 구조 생성기, 시뮬레이션 엔진 등에 대해서 설명한다. 4장에서는 시뮬레이션 결과를 분석하는 파형 분석기에 대해서 설명하고, 5장에서는 ISCAS '85의 예제 회로를 가지고 종래의 event-driven 시뮬레이터와 비교한 결과를 보여주며, 마지막 6장에서는 결론 및 앞으로 할 일에 대해서 설명한다.

2. 입력부 도구 개발

전자회로에서 기능 레벨의 시뮬레이션을 하는데 정보 입력이 중요한 부분의 하나이다. 각 부품들의 정보는 물론이고, 다양한 네트 리스트의 정보를 어떻게 입력하느냐가 시뮬레이터 범용성

의 요점이다. 이런 관점에서 입력부 드구 개발은 부품 정보 입력과 다양한 네트 리스트 정보 입력에 중점을 두었다. (그림 1)의 시스템 구조에서 나타나듯이 입력 데이터는 네트 리스트 화일, PDL 화일, behavioral 모델 등 세 부분으로 나누어 진다. 입력부 도구들은 세 종류의 입력 데이터를 컴파일하는 도구로서, 네트 리스트 컴파일러(EDIF Reader/Writer, SDL compiler/decompiler), PDL 컴파일러, behavioral 컴파일러 등이다. 그리고 기존 상용 CAE 툴과의 연계 사용을 위하여 상용 툴의 stimulus 화일을 개발된 시뮬레이터로 입력시키기 위한 다수의 stimulus 변환기가 구현되었다. 본 시스템의 주요 특징인 PDL 화일과 behavioral 화일 등 분리된 부품 정보를 효과적으로 처리해 주는 것이 입력부 도구들의 특성이다.

2.1 네트 리스트 컴파일러

네트 리스트 컴파일러들은 회로 편집기 또는 문서 편집기를 이용하여 생성된 ASCII 네트 리스트 정보(부품간 연결정보)를 가지고 binary 연결 데이터베이스(CDB:Connectivity Data Base) 형태로 변환하는 기능을 갖고 있다. 본 시스템에서는 산업체에서 광범위하게 사용되고 있는 SDL(Structured Design Language)[9] 및 전자회로 표현의 기본 형태인 EDIF (Electronic Design Interchange Format)[8]를 표준 네트 리스트 표현용 언어로 채택하여 이에 대한 compiler/decompiler을 구현하였다.

2.1.1 네트 리스트 데이터 구조

효과적인 네트 리스트 데이터 구조[6,7]는 로직 시뮬레이터의 성능에 매우 중요한 영향을 미친다. 계층적 네트 리스트 표현을 위한 데이터 구조는 모듈, 인스턴스, 및 네트 등을 상호연결하는 구조를 포함해야 한다. 이러한 구조에서 각종 데이터는 인스턴스나 네트를 flatten하는 데 사용된다. 시뮬레이션 수행시 flatten 인스턴스나 flatten 네트가 필요하며, flatten 인스턴스는 계층적 path로 primitive까지 traverse할 때마다 만들어진다. Flatten 네트는 한 모듈이 불려질 때마다 모든 관련된 모듈의 내부 네트를 구성할 때나 최상위 모듈의 외부 네트를 구성할 때 만들어진다.

2.1.2 EDIF Reader/Writer[12]

CAD/CAE 산업체에선 그간 상이한 CAD 툴에 의해 설계된 설계 정보를 서로간 변환이 가능하게 하는 표준화된 설계 정보 표현 방식의 제정에 많은 노력을 기울려왔다. 이러한 노력의 일환으로 제정된 것이 EDIF이다. 대부분 상용화된 CAD 툴들이 EDIF를 지원하고 있으므로, 개발된 시뮬레이터의 입력으로 EDIF의 사용은 필수적이다. EDIF reader는 yacc rule, 데이터 구조 구현 및 CDB 생성 부분으로 구성되어 있으며, 시스템의 연결정보가 EDIF로 표현된 자료를 읽어서 binary CDB를 만든다. EDIF writer는 CDB로부터 EDIF 네트 리스트 화일을 출력한다.

2.1.3 SDL Compiler/Decompiler

SDL은 전자 회로의 연결도를 기술하는 텍스트 형태의 언어이다. 필요한 주요소는 부품 정보와 연결을 위한 네트 정보이며, 회로의 여러 레벨을 계층적으로 표현도 가능하다. SDL 컴파일러를 만들어 개발된 시스템의 네트 리스트 입력으로 사용케하므로서 시스템의 범용성을 높였다. SDL compiler는 시스템의 연결정보가 SDL로 표현된 자료를 가지고 binary CDB를 만든다. SDL decompiler는 CDB로부터 SDL 네트 리스트 화일을 출력한다.

2.1.4 CDB 화일 구조

네트 리스트 컴파일러는 입력 네트 정보로부터 binary CDB 화일을 만드는 데, binary CDB 화일은 여러개의 크기가 다른 section으로 구성되어 있다. CDB 화일의 구조는 6개의 section으로 구성되어 있으며, 신호 정보를 가지는 네트 리스트 화일과 신호 정보가 없는 primitive 화일로 나누어진다. 네트 리스트 화일은 header section, port 정보, type 정보, 인스턴스 정보, 및 string table section으로 구성되어 있으며, primitive 화일은 header section, port 정보 및 string table section만으로 이루어진다. CDB 화일의 맨 앞에

위치한 header section은 이 화일의 일반적인 정보를 포함하고 있다. 특히 각 section의 offset 값도 포함하고 있어 필요시 직접적으로 각 section에 접근할 수 있다.

2.1.5 상용화된 툴 정합 기능

기존 상용 CAE 툴과 연계사용이 가능한 수종의 정합 기능도 개발하였다. 예를 들면 EDIF reader 및 stimulus 변환기 등이다. 현재 CAE 회사의 vector 화일을 WDB(Waveform Data Base)로 변환하는 4종류의 변환기(cmd2wdb, log2wdb do2wdb bang2wdb)가 있다. cmd2wdb는 ViewLogic cmdfile을 WDB 화일로, log2wdb(do2wdb)는 MentorGraphics logfile(dofile)을 WDB 화일로 변환한다. bang2wdb는 bang vector 화일을 WDB 화일로 변환한다.

2.2 Behavioral 언어 컴파일러

모든 부품의 기능은 표준 C 언어를 이용하여 표현한다. C 언어로 표현된 모든 기능들은 C 컴파일러를 사용하여 컴파일한 후 라이브러리 화일에 저장해둔다. 시뮬레이션 수행시 기능이 필요하면 라이브러리 화일에서 불러 사용한다. (그림 2)에 JK 플립플롭의 behavioral C-모델을 보여준다.

```
/* behavioral C-models for JK FF
   negative-edge triggered f/f with active low preset/clear */
int ccjkne (input, output, state)
int *input, *output, *state;
{
   register int clock, preset, clear;
   preset = input[1]; clock = input[2]; clear = input[3];
   if (preset != ONE) /** preset is zero, undef, float **/
       output[1] = xx;
   else if (clear != ONE)
       output[1] = (clear == ZERO) ? ZERO : UNDEF;
   else if (clock >= UNDEF) output[1] = UNDEF;
   else if (vfall[state[1]][clock]) { /** falling edge **/
     register int j, k;
     j = input[4]; k = input[5];
     if (j == ZERO) {
         if (k == ZERO) output[1] = state[2];
         else if (k == ONE) output[1] = ZERO;
     }
     else if (j == ONE) {
         if (k == ZERO) output[1] = ONE;
         else if (k == ONE) output[1] = vinv[state[2]];
     }
   }
   else {
     output[0] = 0; /* local disconnect: no change */
     state[1] = clock; /** last clock **/
   }
   output[2] = vinv[output[1]];
   state[1] = clock; /** last clock **/
   state[2] = output[1];
}
```

(그림 2) JK 플립플롭의 behavioral C-모델
(Fig.2) Behavioral C-Model for JK-Flipflop

2.3 PDL 컴파일러

디지탈 설계에서 부품들의 정보가 정확해야만 정확한 시뮬레이션 결과를 기대 할 수 있다. 사용자가 쓰기 편하고 많은 정보를 기술하는 PDL의 사용이 필수적이다. 본 시뮬레이터에서는 PDL을 정의하고, PDL로 기술된 입력화일을 분석하여 사용하기 편한 데이터 구조를 만들어 디스크에 저장하고, 필요시 이 데이터를 로딩하는 방법을 채택하였다. (그림 3)에서 보듯이 PDL는 부품의 핀 데이터, 타임 정보, 커패시턴스 등등을 표현하는 언어이다. PDL 컴파일러는 각 부품의 PDL 입력 화일을 읽어 binary primitive

```
PRIMITIVE p74ls375
{ GENINFOS
   { CLASS:seq,4bit_latch1;
     INPUTS:D[1-4](4*1),D[8-5](0000),E12(0),E34(2);
     OUTPUTS:Q[1-4](4*0),P5(0);
     BIPUTS:B1(0,2);
     TYPES:cc375,pwr_of_fn;
     STATES:5(0000222);
   }
  TIMEINFOS
   { PATHDELAYS
     { TIMEUNIT:ps;
       E12=> Q1: (10:18:24,10:18:24)(-,-)(10:20:30,10:20:30);
       D2 => Q1,Q3 : (10:18:30,10:18:30);
       D1,D3 => P5 : (10:18:45,10:18:35);
       D[1-4] => Q[1-4] : (18,30);
     }
     TIECHECK
     { TIMEUNIT:ps;
       SETUP:D[1-4],E12,(15,15); SETUP:D[1-4],E34,(15,15);
       HOLD:D[1-4],E12,(5,5); HOLD:D[1-4],E34,(5,5);
       MPW:E12,(20,20); MPW:E34,(20,20);
       TREC:E12,E34,15; SKEW:E12,E34,10; CYCLE:E12,10;
     }
   }
   CAPACITANCE
   { CAPUNIT:pF;
     SLEW:Q[1-4],1.152,0.936;
     D[1-4]:0.102; Q[1-4]:0.064;
   }
}
```

(그림 3) PDL 화일 구조
(Fig. 3) PDL File Structure

데이터 화일을 만든다. PDL 로더는 시뮬레이션 수행에 필요한 부품 정보를 로딩한다.

PDL 언어의 기본적인 구성은 4부분으로 되어 있다. 첫번째 부분은 부품의 일반적인 정보로 부품 종류, 입력 핀, 출력핀, 입출력 핀, 기능 함수, 상태 변이등이 포함된다. 두번째 부분은 핀과 핀 사이의 지연 시간을 표시하며, 시간에 관한 모든 정보를 입력 가능하다. 세번째 부분은 일반적인 시간 정보를 입력하여 회로 시뮬레이션시 각 시간 정보를 조사케 한다. 네번째는 각 핀별로 커패시턴스에 관한 정보를 입력시켜 ASIC 개발시 chain effect delay나 derating factor delay 등도 계산 가능케 한다.

3. 시뮬레이터 구현

시뮬레이션을 위해서는 먼저 컴파일된 네트 리스트 정보를 읽어 계층을 없애고, 이로부터 시뮬레이션에 적합한 내부 데이터 구조를 만든다. 그 다음 시뮬레션 엔진에서 필요한 부품의 behavioral 모델을 점진적 로더를 이용하여 주 메모리로 읽어드린 후 시뮬레이션을 시작한다. 시뮬레이션에 필요한 stimulus 및 시뮬레이션 결과를 표현하기 위한 범용 파형 데이터베이스(WDB: Waveform DataBase)도 함께 구현하였다. 현재 구현된 시뮬레이터는 event-driven 기능 레벨 zero/unit 지연 로직 시뮬레이터로서, C 언어로 쓰여진 부품의 behavior를 컴파일하여서 이를 시뮬레이션이 시작될 때 점진적 로더를 이용하여 다이나믹하게 로딩함으로써 기존의 interpretive 시뮬레이터에 비해 빠른 성능을 갖고 있다. 현재 구현된 시뮬레이션 엔진은 zero/unit-delay만 처리하게끔 되어 있으나, PDL 모델에서 부품의 지연시간을 고려하여 event를 스케쥴할 경우 단순 지연시간 시뮬레이션이 가능하다. 또한 setup-hold와 같은 타이밍 오류도 같은 방법으로 검증을 할 수가 있고, behavioral 모델 언어로 C 언어를 채택함으로서 복잡한 하드웨어 기술언어를 습득할 필요없이 사용자가 쉽게 부품의 특성을 모델링할 수 있을 뿐만 아니라 시뮬레이터 자체도 C 언어로 구현되었으므로 어떤 UNIX 환경에서도 사용이 가능하다. (그림 1)에서 보듯이 시뮬레이터는 계층적 확장기, 점진적 로더, 데이타 구조 생성기 및 시뮬레이션 엔진으로 구성되어 있다. 본 시스템의 주요 특성인 분리하여 컴파일된 부품 데이터를 효과적으로 처리하는 기능들이 점진적 로더와 시뮬레이터 엔진에 포함되어 있다.

3.1 계층적 확장기

주어진 회로를 시뮬레이션 하기 위해서는 먼저 시뮬레이션할 네트 리스트 정보가 입력된다. 입력된 네트 리스트의 최상위 모듈로부터 시작하여 필요한 부속 모듈을 라이브러리 화일로부터 모두 로딩시킨다. 이렇게 로딩된 모든 모듈들은 모듈 인덱스 구조에 의해 에러가 조사되며, 소속된 모듈이 primitive일때까지 이러한 과정을 계속한다. 네트 리스트 데이터를 입력한 후 확장기는 레벨 정리 구조를 사용하여 flatten 네트나 flatten 인스턴스를 만든다. 확장기는 전체의 계층적인 트리 구조를 기초로 DFS(Depth-First Search) 알고리즘을 사용하여 모든 모듈들을 traverse 한다.

3.2 점진적 로더

회로의 기능을 시뮬레이션 하기전에 C 언어로 쓰여진 각 부품의 behavioral 모델은 컴파일되어 라이브러리 화일에 저장된다. 시뮬레이션하는 동안에 점진적 로더는 필요한 부품의 컴파일된 데이터를 시뮬레이션 엔진에 직접 로딩 한다. 이것이 개발된 시뮬레이터의 주요 특성 중의 하나이다. 이렇게 함으로써 종래의 다른 event-driven 시뮬레이터보다 현저한 속도 향상을 가져올 수 있었다.

3.3 데이터 구조 생성기

시뮬레이션 엔진에서 네트 리스트 표현을 위한 데이터 구조는 게이트, 신호, 텍스트 3종류가 상호 연결되어 이루어진다. 게이트 데이터는 텍스트 정보와 여러개의 신호 데이터를 포함하고 있고, 신호 데이타는 여러개의 게이트 데이터를

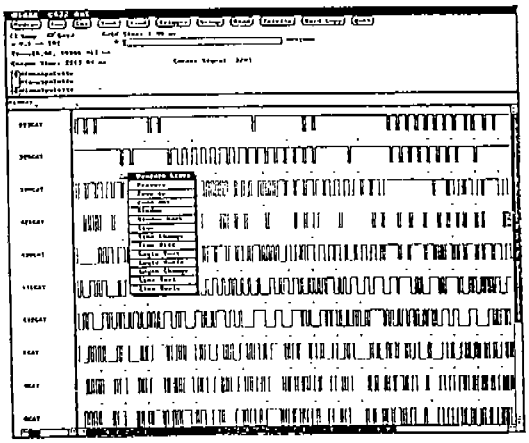
포함하고 있다. 이러한 구조의 데이타는 게이트나 신호의 event를 만들기 위하여 사용된다. 데이터 구조 생성기는 연결 데이터 베이스(CDB)로부터 기능 시뮬레이션을 위한 데이터 구조를 만든다.

3.4 시뮬레이션 엔진

시뮬레이션 엔진은 본 시스템의 핵심 부분이다. 이 부분에 여러가지 기능이 있지만 event scheduler와 selective tracer가 중요한 부분이다. Event scheduler는 linear linked list를 이용하여 신호와 게이트 event를 schedule 한다. Selective tracer는 한 event를 선택하여 주어진 함수를 계산한 후 결과를 이웃 게이트나 신호로 전달한다. 이 두가지 기능이 event-driven 시뮬레이터의 주요 특성이다.

4. 파형 분석기

시뮬레이션 결과를 분석하는 툴로서 파형을 보여주는 것 외에 파형 분석을 위한 편리한 여러 기능을 갖고 있다. 주요 기능으로는 파형의 시간 측정 및 논리 값의 측정, 다른 시뮬레이션 결과와의 비교분석, 그리고 화면에 나타난 파형의 목록을 편집하는 것 등이 있다. 파형 분석기의 기능 수행을 위한 명령을 사용자가 키보드 입력, 마우스를 이용한 메뉴 선택, 그리고 버튼 등을 통해 편리하게 입력할 수 있다. GUI를 위한 윈도우 구성은 (그림 4)와 같으며, 윈도우의 사용 목적은 다음과 같다.



(그림 4) 파형 분석기
(Fig. 4) Waveform Analyzer

- 버튼 명령 판넬: 명령 입력을 위한 버튼과 임의의 시간에 대한 파형을 분석하기 위한 슬라이더 바를 제공한다.
- 시간 캔버스: 파형의 시간 측정을 위해 눈금으로 된 자를 나타낸다.
- 시간 판넬: 시간 캔버스의 큰 눈금 사이의 시간을 표시한다.
- 파형 이름 캔버스: 파형의 이름을 표시하며 파형의 목록을 편집하는 데 이용된다.
- 파형 캔버스: 파형을 그래픽으로 나타내며 파형의 시간 측정 및 논리 값의 측정을 위해 이용된다.
- 메시지 윈도우: 명령의 수행 과정과 수행에 필요한 정보를 표시한다.
- 키보드 명령 판넬: 키보드를 통한 명령입력을 위해 사용된다.

5. 실험 결과

앞장에서 설명된 내용을 주축으로 구현된 시뮬레이터(HSIM)의 성능을 평가하기 위하여 ISCAS '85의 중간 크기의 벤치마크 회로를 사용하였다. 이 회로들은 ATPG 패키지나 시뮬레이터의 성능을 테스트하는 데 자주 사용된다. 사용된 모든 회로는 조합회로이며, HSIM은 두개의 다른 시뮬레이터와 성능을 비교했다. 〈표 1〉에 LECSIM[4], EUSIM[4] 및 HSIM의 성능 평가 결과를 보여주고 있다. LECSIM은 LEvelized event-driven Compiled 로직 시뮬레이터이며, EUSIM은 전통적인 단위 지연, two-pass event-driven interpretive 로직 시뮬레이터이다. 〈표 1〉에 표시된 결과는 시뮬레이션 수행 시간이며, LECSIM이나 EUSIM과 달리 HSIM에는 회로 정보와 입력 벡터를 입력하는 시간과 시뮬레이션 결과를 출력하는 시간도 합산된 것이다. HSIM에서는 모든 기능이 한꺼번에 수행되므로 네트 리스트 입력이나 시뮬레이션 벡터 입력, WDB로 출력하는 각각의 시간을 분리하여 조사

할 수 없었다. 그래서 이 시간은 전체 시뮬레이션 수행 시간에 큰 영향이 없을 것으로 생각하고 성능을 비교했다. 모든 수행 속도 테스트는 SUN 3/260에서 수행되었으며, 각 회로마다 임의로 선택된 5000개의 입력 벡터를 사용하여 시뮬레이션 한 결과이다. 〈표 1〉에서 보듯이 HSIM은 같은 event-driven 시뮬레이터인 EUSIM보다 55% 빨았으며, levelized event-driven 시뮬레이터인 LECSIM보다는 느렸다.

〈표 1〉 벤치마크 테스트 결과
〈Table 1〉 Results of Benchmark Test

Circuit	LECSIM	EUSIM	HSIM
C432	95	190	124
C499	128	213	144
C880	219	356	232
C1355	309	763	398
C1098	500	1466	860
C2670	707	1550	1040
C3540	875	2318	1405
C5315	1347	3815	2636
C7552	2133	6257	4058
Total	6313	16946	10900

***Run Time in Seconds(sun 3/260)

6. 결론 및 앞으로 할 일

Event-driven 시뮬레이션 알고리즘의 성능향상을 위해 많은 연구[10,11]를 했지만 아직도 문제점이 많다. 이 문제를 해결하고자 event driven 로직 시뮬레이터인 HSIM을 개발했다. 본 시스템의 새로운 아이디어는 첫째 각 부품 모델에 대해서 behavioral 정보와 primitive 정보를 분리하여 모델을 구성한 것이다. 이렇게 함으로써 개발자들이 필요한 부품의 상세한 정보를 따로 관리하므로 도구의 효용성이 증가하고 사용이 용이하다. 둘째로는 분리하여 구성된 정보를 한꺼번에 로드하는 것이 아니라 필요시 점진적 로더를 사용하여 모델을 로딩하는 것이다. 이 기능은 사전에 수행해야 할 여러 단계를 한꺼번에 처리 가능하며 성능을 향상할 수 있다. 현재의 시뮬레이터는 기능 수준 zero/unit 지연시간 모드만 시뮬레이션 가능하다. 그러나, PDL 컴파일러의 정보를 사용하면 단순 딜레이 모드는 쉽게 응용 할 수 있다. HSIM이 종래의 event driven interpretive 시뮬레이터와 다른 점은 점진적 로더 기술을 사용하여 성능을 향상시킨 것이다. 실험 결과를 보면 이 기술이 매우 우수하다는 것을 보여주고 있다. HSIM는 전통적인 단위 지연 event-driven interpretive 시뮬레이터인 EUSIM보다 약 55%가 빨았다. LECSIM (LEvelized event-driven Compiled SIMulator)이 HSIM보다 72%정도 빨았지만 궤환회로가 많이 있는 회로나 event가 작은 회로에서는 event-driven 접근방식을 사용한 HSIM이 매우 유용할 것으로 사료된다.

현재 HSIM은 zero/unit 지연시간만 가능하지만 앞으로 multi-delay로 확장중에 있으며, mixed 모드 시뮬레이션도 가능하게 circuit level의 모델도 첨가할 예정이다. 또한 in-house 툴의 특성을 살리기 위해 개발자들이 필요로 하는 특수한 부품들도 (monitor, driver, bus elements 등) 수용할 것이며, Synopsys사의 SmartModel과 같은 상용 회사의 모델도 사용 가능하게 할 예정이다. HSIM의 최종 목표는 C- 모델을 수용하는 VHDL 시뮬레이터와 병행 사용 가능하게 하는 것이다.

참 고 문 헌

[1] M. A. Breuer, A. D. Friedman, 'Diagnosis and Reliable Design of Digital Systems', Computer Science Press, Woodland Hills, CA, 1976

[2] E. G. Ulrich, D. Herbert, "Speed and Accuracy in Digital Network Simulation based on Structural Modeling", Proc. 19th DAC, pp. 587-593, 1982

[3] S. Gai, F. Somenzi, M. Spalla, "Fast and Coherent Simulation with Zero Delay Elements", IEEE Trans. on CAD, Vol. 6, No. 1, pp. 85-91, Jan. 1987.

[4] Z. Wang, P. M. Maurer, "LECSIM: A Levelized Event Driven Compiled Logic Simulator", Proc. 27th DAC, pp. 491-496, June 1990

[5] S. P. Smith, M. R. Mercer, B. Brock, "Demand Driven Simulation: BACKSIM", Proc. 24th DAC, pp. 181-187, 1987

[6] S.Meyer, "A Data Structure For Circuit Net Lists", Proc. 25th DAC, pp. 613-616, 1988

[7] W.C. Diss, "Circuit Compilers Don't Have To Be Slow" Proc. 25th DAC, pp. 622-627, 1988

[8] 'Electronic Design Interchange Format Version 200', Electronic Industries Association, EDIF Steering Committee, 1987

[9] 'Structured Design Language(SDL)', Silvar-Lisco/SDL Design Capture Command Reference Manual, Vol.1[10] S. Gai, F. Somenzi, M. Spalla, "Fast and Coherent Simulation with Zero Delay Elements", IEEE Trans on CAD, Vol. CAD-6, No.1, pp. 85-92, Jan 1987

[11] Z. Barzilai, J.L. Carter, B.K. Rosen, J.D. Rutledge "HSS - A High-Speed Simulator", IEEE Trans on CAD, Vol. CAD-6, No.4, pp. 601-617, July 1987

[12] 박장현, 김보관, 이기준 "EDIF Reader/Writer의 구현", 전자계산, 반도체. 재료 및 부품, 씨에이디 및 VLSI 설계 학술발표회 논문집, pp. 135-138, 1992

박 장 현

1983년 서강대학교 전자공 학과(공학사)
1985년 AIT 컴퓨터공학과(공학석사)
1994년 충남대학교 전자공 학과 박사과정 수료
1985년~현재 한국전자통신연구소 선임연구원
관심분야 : 설계자동화, 개인통신(PCS)

이 기 준

1978년 서울대학교 전자공 학과(공학사)
1981년 한국과학기술원 전기및전자공학과(공학석사)
1986년 한국과학기술원 전기 및 전자공학과(공학박사)
1983년 ~86년 한국과학기술연구소 연구원
1986년~90년 충남대학교 전자공학과 조교수
1990년~현재 충남대학교 전자공학과 부교수
관심분야 : Mixed Mode Sumulator, VLSI-CAD, Simulator.

김 보 관

1976년 서울대학교 전자공 학과(공학사)
1978년 국과학기술원 전기 및 전자공학과(공학석사)
1989년 Univ. of Wiscon sin-Madison 전기및컴퓨터공학과(공학박사)
1978년~80년 한국과학기술연구소 연구원
1980년~91년 금오공대 전자공학과 조교수
1991년~현재 충남대학교 전자공학과 부교수
관심분야 : Logic Synthesis, Hardware/Software Codesign