게이트 및 기능 레벨 논리 시뮬레이터

°박홍준, 김종성, 조순복, 신용철, 임인칠
한 양 대 학 교

A Gate and Functional Level Logic Simulator

H.J.Park, J.S.Kim, S.B.Cho, Y.C.Shin, I.C.Lim
Hanyang University

ABSTRACT

This paper proposes a gate and functional level logic simulator which can be run on XENIX O.S. The simulator has hierarchical structure including Hardware Description Language compiler, Waveform Description Language compiler, and Simulation Command Language compiler. The Hardware Description Language compiler generates data structure composed of gate structure, wire structure, conditon structure, and event structure.

Simulation algorithm is composed of selective trace and event-driven methods. To improve simulation speed, Cross Referenced Linked List Structure is defined in building the data structure of circuits.

Ⅰ. 서 론

현재 LSI/VLSI 설계시 회로를 검증하기 위한 방법으로 시뮬레이션이 널리 사용되고 있으며, 시뮬레이션은 회로 레벨, 타이밍 레벨, 논리 레벨, 레지스터 레벨등으로 분류할 수 있다.[1]

논리 레벨 시뮬레이션은 입출력 값들의 관계를 모델링함으로써 논리적인 정확성, race 및 hazard의 검출을 가능하게 하며, 게이트 레벨과 기능 레벨 시뮬레이션으로 구분할 수 있다. 게이트 레벨 시뮬레이션은 회로 분석이 정확한 반면, 회로가 대형화되면 기술하는 게이트의 수가 급격히 증가하게 되어 과다한 기억 용량이 요구되며 복잡한 게이트들을 모델링하기 위해 많은 시간이 소모된다. 기능 레벨 시뮬레이션은 대규모 회로 설계에 용이할 뿐만 아니라 시뮬레이션 시간을 크게 감소시킬 수 있다. 따라서 회로 분석이 정확한 게이트 레벨 뿐만 아니라 대규모 회로 설계에 유용한 기능 레벨에서도 수행이 가능한 시뮬레이터가 절실히 요구되고 있다.

현재 논리 레벨 시뮬레이터는 HILO-3, TEGAS, FUNSIM과 PC용으로 PCAD등 여러 가지가 사용되고 있으며, 최근에 HILO-3가 대형의 범용 컴퓨터를 이용한 VLSI 설계에 널리 사용되고 있다.

본 논문에서는 HILO-3와 호환성이 있는 논리 시뮬레이터 SWEET(Soft Ware Electronic Engineering Tool)를 개발한다. 개발한 시뮬레이터는 게이트 레벨 뿐만 아니라 기능 레벨에서도 수행이 가능한 논리 시뮬레이터로 PC용으로 일반 사용자가 손쉽게 사용할 수 있도록 한다.

회로에 대한 데이타 구조는 게이트 구조, wire 구조와 기능 레벨에서만 사용되는 조건 구조와 event 구조로 구성하며, 이 데이타 구조는 시뮬레이션 속도를 향상시키기 위해 Cross Referenced Linked List 구조로 만든다.

제안한 논리 시뮬레이터에 대한 확장성을 고려하여 회로 기술 언어(Hardware Description Language, HDL) 컴파일러, 입력 파형 기술 언어(Waveform Description Language, WDL) 컴파일러, 시뮬레이션 명령 언어(Simulation Command Language, SIM) 컴파일러를 XENIX O.S상에서 yacc[5]를 이용하여 구성한다.

Ⅱ. SWEET 구조

SWEET는 그림1과 같이 5부분들로 구성된다. SWEET 명령 프로세서는 상위 레벨 명령을 처리한다. HDL 컴파일러는 HDL subfile을 읽고 회로에 대한 데이타 구조를 설정하며, WDL 컴파일러는 WDL

subfile을 읽고 입력 파형 데이타 구조를 설정한다. SIM 컴파일러는 SIM subfile을 읽어 입력 파형을 갖고 회로를 시뮬레이션할 환경을 만든다. 시뮬레이션 프로세서는 WDL에 의해 준비된 입력 파형을 갖고 HDL에 의해 기술된 회로를 평가한다.

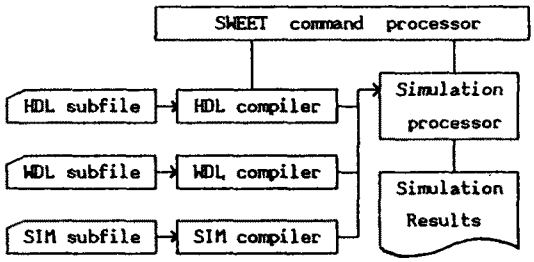


그림 1. SWEET의 계층 구조

HDL 컴파일러에 의해 구성되는 데이타 구조는 그림 2와 같이 게이트 구조, wire 구조, 조건 구조와 event 구조로 이루어진다.

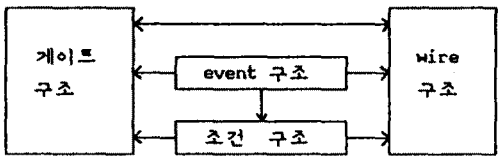


그림 2. SWEET의 데이타 구조

Ⅲ. 기술 언어

논리 시뮬레이션을 수행하기 위해서는 회로에 대한 기술과 입력 파형에 대한 기술, 그리고 이 회로와 입력 파형을 지정하여 시뮬레이션을 실행시키는 시뮬레이션 명령어들이 필요하다. SWEET에서는 회로를 HDL로 기술하며, 표 1과 같은 게이트 모델, wire 모델과 기능 모델의 3가지 모델을 사용한다. 입력 파형은 WDL을 이용하여 기술하며 시뮬레이션 명령어로서는 SIM을 이용한다.

HDL에 대한 예로서, 그림 3과 같은 RAM 기능 모델을 HDL로 기술한 것이 그림 4이다.

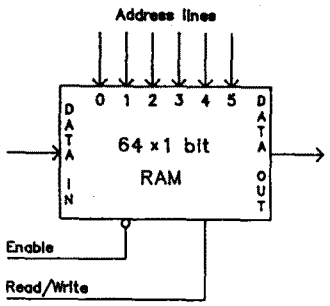


그림 3. RAM 기능 모델

```
cct rammodel(dataout, enable, rw, address[0:5],
                                  datain)
register(1, 1)     dataout;
ram(0:63)          r;
when rw(0 to 1) do
     if enable = 0 then r[address] = datain
     endif;
when address(0 to x) or address(1 to x) do
     display("address line went unknown at time",
             time);
when address(0 to 1) or address(1 to 0) do
     if enable = 0 then dataout = r[address]
     endif;
input    datain enable rw address[0:5].
```

그림 4. 그림 3의 회로에 대한 HDL 기술

WDL은 입력 파형의 기술 뿐만 아니라 회로의 각 wire를 초기화할 수 있다. wire의 초기값을 알 수 없을 경우에는 X로 초기화할 수 있다. 그림 5과 같은 입력 파형을 WDL로 나타낸 것이 그림 6이다.

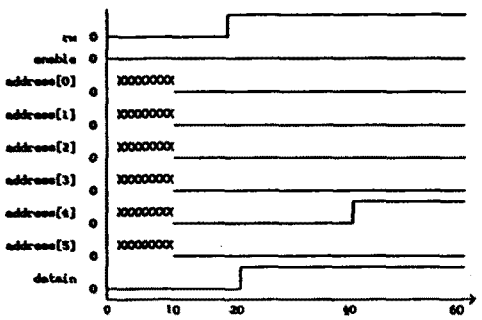


그림 5. 입력 파형

```
waveform ramw
stimulus rw, datain, enable = 0 address[0:5] = x;
initialize  dataout = x;
10       address[0:5] = 0;
20       rw = 1       datain = 1;
40       address[0:5] = bin 000010;
60       finish.
```

그림 6. 그림 5에 대한 WDL 기술

그림 4의 HDL과 그림 6의 WDL을 이용하여 시뮬레이션 하기 위해서는 회로의 이름과 입력 파형의 이름, 지연 모드와 시뮬레이션 결과로 보고자 하는 wire들을 지정해야 한다. 예를 들어 지연 모드를 minimum으로 하고 그림 4의 각 wire들을 보고자 할 경우는 그림 7과 같이 SIM을 기술한다.

```
sim ramsim
source rammodel ramw
min
dch(time...rw..enable..datain..address...dataout).
```

그림 7. SIM 기술

Ⅳ. 시뮬레이션 알고리즘과 적용 예

시뮬레이션 속도는 알고리즘뿐만 아니라 회로에 대한 데이타 구조에도 그 영향을 받는다. 시뮬레이션을 수행하는 동안, 게이트의 형, 게이트 이름, 지연,

입출력 wire들이 계속 참조되며 또한 wire의 형, wire 이름, wire를 구동하는 게이트와 wire에 의해 구동되는 게이트들도 계속 참조된다. 이러한 이유로 회로에 대한 데이타 구조는 Cross Referenced Linked List 구조로 만들어진다. 그림9는 그림8의 회로에 대한 데이타 구조를 표현한다.

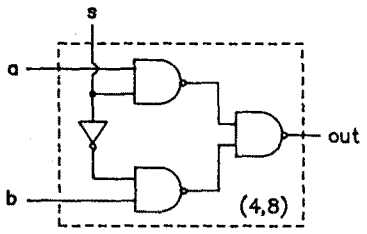


그림8. 2입력 멀티 플렉서

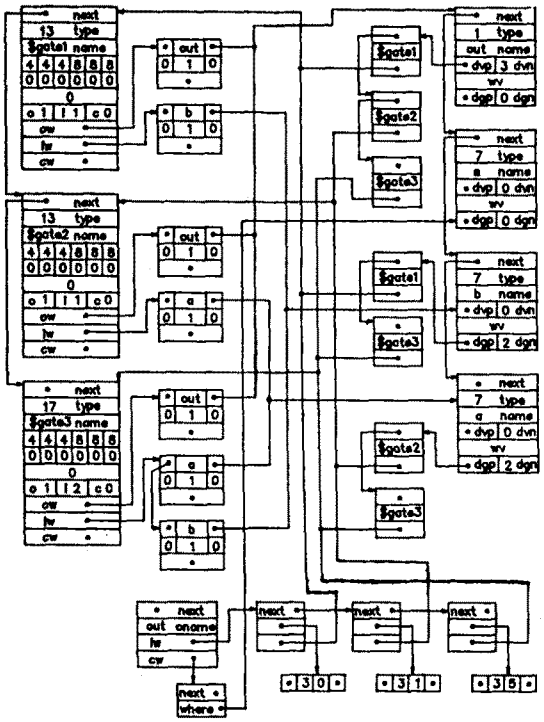


그림9. 그림8에 대한 데이타 구조

시뮬레이션 알고리즘은 selective trace와 event-driven 방법을 이용한다.

단계1] 스케쥴된 wire를 하나 선택하여 wire 구조로부터 그 wire를 찾는다.

단계2] wire 구조에서의 wire 값이 스케쥴된 wire의 값과 다르다면, 스케쥴된 wire의 값으로 바꾼다. 이 wire에 의해 구동되는 게이트 집합에 삽입한다.

단계3] 단계1부터 단계2까지를 한 시간 단위에 스케쥴되는 모든 wire들에 대해 반복 수행한다.

단계4] 게이트 집합으로부터 한 게이트를 선택하여 입력 값들과 control 값들로부터 새로운 출력 값을 계산한다.

단계5] 새로운 출력 값이 이전 출력값과 다르다면,

1) 지연 시간을 계산한다.

2) 출력 wire가 tri-state나 wired wire가 아닐 경우, 다중 스케쥴 모드이면, 이전 스케쥴을 새로운 스케쥴로 바꾼다. 다중 스케쥴 모드가 아니라면, time wheel에 이 event를 삽입한다.

3) 출력 wire가 tri-state wire이거나 wired wire라면, tri wire 집합에 이 wire를 삽입한다.

단계6] 게이트 집합에서 그 게이트를 제거한다.

단계7] 게이트 집합이 null이 될 때까지 단계4부터 단계6까지 반복 수행한다.

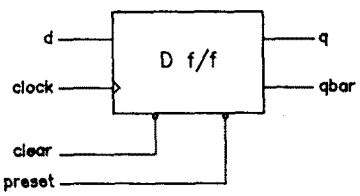
단계8] tri wire 집합으로부터 하나의 wire를 선택하여 drive 값들로부터 tri-state wire를 평가한다.

단계9] 새로운 wire 값이 이전 wire 값과 다를 때, 다중 스케쥴 모드가 아니면, time wheel에 이 event를 삽입한다. 다중 스케쥴 모드이면, 이전 스케쥴을 새로운 스케쥴로 바꾼다.

단계10] tri wire 집합으로부터 그 wire를 제거한다.

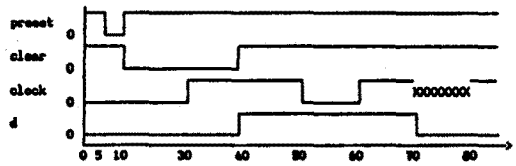
단계11] tri wire 집합이 null이 될 때까지 단계8부터 단계10까지 반복 수행한다.

본 논문에서 제안한 알고리즘을 이용하여 그림10과 같은 기능 레벨에서의 D 플립플럽에 그림11과 같은 입력 파형을 인가하여 그림12와 같은 결과를 얻을수 있었다.




```
cct dff(q, qbar, clock, d, preset, clear)
wire(2,3)       q = regq;
wire            qbar = not q;
register(1,1)   regq = case [ preset, clear ],
                          01 = 1,
                          10 = 0,
                          00 = x
                        endcase;
when clock(0 to 1) do regq = d;
when clock(0 to x) or clock(x to 1) do
        regq = d same regq;
when clock(0 to x) or clock(1 to x) do
        display("clock went unknown at time",
                time);
input clock preset clear d.
```

그림10. D 플립플럽의 기능 레벨 모델링




```
waveform dffw
stimulus d, clock = 0  preset, clear = 1;
initialize q, qbar = x;
5       preset = 0;
10      preset = 1      clear = 0;
30      clock = 1;
40      d = 1           clear = 1;
50      clock = 0;
60      clock = 1;
70      d = 0           clock = x;
80      clock = 1;
90      finish.
```

그림 11. 입력 파형

```
sim dffsim
source dff dffw
dch(time...d..clock..preset..clear...q..qbar).

                d c p c  q q
                  l r l    b
                  o e e    a
                  c s a    r
                  k e r
                    t
  time----
  0             0 0 1 1  X X

  5             0 0 0 1  X X
  8             0 0 0 1  1 0

  10            0 0 1 0  1 0
  14            0 0 1 0  0 1

  30            0 1 1 0  0 1

  40            1 1 1 1  0 1

  50            1 0 1 1  0 1

  60            1 1 1 1  0 1
  63            1 1 1 1  1 0

  dff : clock went unknown at time 70

  70            0 X 1 1  0 1

  80            0 1 1 1  0 1
  83            0 1 1 1  X X
```

그림 12. 시뮬레이션 결과

V. 결 론

본 논문에서는 HDL 컴파일러, WDL 컴파일러, SIM 컴파일러를 포함하는 게이트 레벨 뿐만 아니라 기능 레벨에서도 실행이 가능한 PC용 논리 시뮬레이터를 개발하였다. HDL 컴파일러에 의한 데이타 구조는 게이트 구조, wire 구조와 기능 레벨에서만 요구되는 조건 구조, event 구조로 구성하여 대규모 회로도 시뮬레이트할 수 있도록 하였다.

시뮬레이션 알고리즘은 Selective Trace와 event-driven 방법을 이용하여 구성하며, 시뮬레이션 속도를 향상 시키기 위하여 회로에 대한 데이타 구조를 형성하는데 Cross Referenced Linked List 구조를 사용하였다.

논리 시뮬레이터 SWEET는 C언어로 작성하여 MS-DOS와 XENIX O.S상에서 실현하였다.

참 고 문 헌

1. P.L. Flake, P.R. Moorby, G. Musgrave, "An Algebra Logic Strength Simulation," IEEE 20th Design Automation Conf., pp. 615-618, 1983.
2. Manuel d'Abreu, "Gate-Level Simulation," IEEE Design and Test, pp. 63-71, 1985.
3. M. Breuer, A.D. Friedamn, Diagnosis & Reliable Computer Science Press, 1976.
4. HILO-3 Reference Manual, GenRad, 1985.
5. Axel T. Schreiner, H. George Friedman, Jr., Introduction to Compiler Construction with UNIX, Englewood Cliffs, N.J., Prentice-Hall, Inc., 1985.
6. P. Wilcox, "Digital Logic Simulation at the Gate and Functional Level," IEEE 16th Design Design of Digital Systems, Potomac, Maryland, Automation Conf., pp. 242-248, 1979.
7. 신용철, 조상복, 임인칠, "논리 레벨 시뮬레이션," 대한전자공학회 하계종합학술대회 논문집 Vol.9, pp. 586-590, 1986.
8. Dan Nash, Keith Russell, Paul Silverman, Mary Thiel, "Functional Level Simulation at Raytheon," IEEE 20th Design Automation Conf. pp. 634-639, 1980.

표 1. 게이트, wire와 기능 모델의 지정어 및 의미

모 델	지정어	의 미
게이트 모델	and	and gate
	or	or gate
	nand	nand gate
	nor	nor gate
	balr	balanced line receiver
	not	not gate
	buf	buffer gate
	xor	exclusive or gate
	xnor	exclusive nor gate
	same	same gate
	bufif0	buffer if control 0 gate
	bufif1	buffer if control 1 gate
	notif0	not if control 0 gate
	notif1	not if control 1 gate
	moveif0	unidirectional if control 0 gate
	moveif1	unidirectional if control 1 gate
	capacitor	capacitor connected at wire
wire 모 델	wire	untyped normal wire
	tri	tri-state wire(floats to Z, if not driven)
	tri0	tri-state wire(floats to 0, if not driven)
	tri1	tri-state wire(floats to 1, if not driven)
	wand	wired and wire
	wor	wired or wire
	input	input wire
	supply0	ground wire(logical 0)
	supply1	power supply wire(logical 1)
기 능 모 델	register	register
	rom	read only memory
	ram	random access memory