論文 89-26-5-15

LOSIM : VLSI의 설계검증을 위한 논리 시뮬레이션 프로그램

(LOSIM : Logic Simulation Program for VLSI)

康 敏 燮*, 李 哲 東*, 柳 瑛 昱*

(Min Sup Kang, Chul Dong Lee and Young Uk Yu)

要 約

본 논문은 mixed level에서 VLSI 회로의 논리설계를 검증하기 위한 논리 레벨 시뮬레이터인 LOSM (LOgic SIMulatos)에 대해서 논의한다.

본논문에서는 8개의 신호값과 2개의 신호강도를 이용하여 일반소자, 기능소자, transmission 게이트 그리고 tri-state 게이트의 경우 종래의 시뮬레이터[5~6,9] 보다 정확한 결과를 얻을 수 있는 모델링 방법을 제안한다. LOSIM은 rise delay와 fall delay를 사용하여 주어진 회로에 대한 타이밍 분석과 hazard분석이 가능하다. Hazard 분석 및 검출은 5상태의 신호값과 time queue를 이용한 scheduled time을 이용한다.

개발된 알고리듬은 SUN-3/160 워크스테이션상에서 C-언어를 사용하여 구현되었으며, 정적 RAM 셀과 비동기 회로에 대해서 프로그램의 동작 예제로 하였다.

Abstract

The simulator described here-LOSIM (LOgic SIMulator)-was developed to verify the logic design for VLSI (Very Large Scale Integrated) circuits at mixed level.

In this paper, we present a modeling approach to obtain more accurate results than conventional logic simulators [5-6, 9] for general elements, functional elements, transmission gates and tri-state gates using eight signal values and two gignal strengths.

LOSIM has the capability which can perform timing and hazard analysis by using assignable rise and fall delays. We also prosent an efficient algorithm to accurately detect dynamic and static hazards which may be caused by the circuit delays. Our approach is based on five logic values and the scheduled time. LOSIM has been implemented on a SUN-3/160 workstation running Berkeley 4.2 UNIX, and the program is written in C language. Static RAM cell and asynchronous circuit are illustrated as an example.

*正會員, 韓國電子通信研究所
(Electronics and Telecommunications Research Institute)

接受日字 : 1988年 10月 4日

I. 서 론

LSI/VLSI 기술의 급속한 발달에 따라 집적회로의 설계를 검증하기 위해서 많은 시뮬레이션 프로그램들이 이용되고 있다. 논리 시뮬레이터를 사용하는 주된 목적은 칩이 완성되기 전에 설계 오류를 미리

검출, 교정하여 설계시간 및 설계비용을 줄이는 데 있다. 논리 시뮬레이터는 게이트 레벨 시뮬레이터와 트랜지스터 레벨 시뮬레이터로 구분된다. 게이트 레벨 논리 시뮬레이터[1~3]는 AND, OR, NAND 등과 같은 논리기능을 검증하는데 사용되며, MOSSIM[4]과 같은 트랜지스터 레벨 시뮬레이터[5,6]들은 MOS 트랜지스터 회로를 검증하는데 사용되는 CAD(computer aided design) 툴이다. 또한 이와같은 논리회로와 MOS 회로를 동시에 취급할 수 있는 혼합형 논리시뮬레이터[7,8]도 유용하게 사용되고 있다.

본 논문은 VLSI(very large scale integrated circuit)의 논리설계를 검증하기 위한 논리 시뮬레이션 프로그램인 LOSIM(LOgic SIMulator)에 관한 것이다. LOSIM은 일반소자, 기능소자, transmission 게이트 그리고 tri-state 게이트를 효과적으로 취급하기 위해서 8상태의 신호값과 2상태의 신호 강도를 이용한다.

본 논문에서는 일반소자, 기능소자, transmission 게이트 그리고 tri-state의 경우 종래의 시뮬레이터[5~6,9]보다 정확한 결과를 얻을 수 있는 모델을 제안한다. 일반소자, 기능소자 그리고 tri-state 게이트의 신호 모델은 5상태의 신호값을 이용하였으며, transmission 게이트는 6상태의 신호값과 2상태의 신호강도를 이용하였다. 본문에서는 또한, 회로지연에 의해서 발생될 수 있는 static hazard와 dynamic hazard를 정확히 검출할 수 있는 알고리듬을 제안한다. Hazard 검출은 5상태의 신호값과 scheduled time을 이용하며, scheduled time을 얻기 위한 timing 메카니즘은 event queue를 이용한다. Delay model은 assignable delay를 사용하였고, 시뮬레이션 알고리듬은 selective trace에 의한 event-driven 방식을 사용하였다. 제안된 hazard 검출알고리듬은 timing 분석에 의한 static hazard와 dynamic hazard 검출이 가능하므로 문헌[11, 13]에서 해결할 수 없었던 timing 분석 및 dynamic hazard 검출이 가능하다. Multi-value를 사용하고 있는 time-based, event-driven 방식에서는 제안된 알고리듬을 컴퓨터상에서 쉽게 실현할 수 있다.

다음 장에서는 신호값 및 신호 강도에 대해서 설명하며, Ⅲ장에서는 회로망 모델링에 대해서 설명한다. Ⅳ장에서는 시뮬레이션 알고리듬 및 데이타 구조에 대해서 설명하고, Ⅴ장에서는 hazard 검출 알고리듬에 대해서 설명한다. Ⅳ장에서는 시뮬레이션 결과에 대해서 고찰한 다음 마지막으로 결론에 대해서 설명한다.

Ⅱ. 신호값 및 신호강도

시뮬레이터의 정확성은 신호 또는 전압 레벨을 표시하는데 사용되는 신호값의 수에 따라 좌우된다. 종래의 논리 시뮬레이션에서 사용된 신호 모델은 3상태[9], 4상태,[6] 8상태[1] 그리고 9상태[10]가 주로 사용되었다.

LOSIM에 사용된 신호모델은 일반소자, 기능소자, tri-state 게이트 그리고 transmission 게이트를 효과적으로 시뮬레이션 하기 위해서 표 1과 같은 8상태의 신호값과 2상태의 신호강도를 이용한다.

표 1. 8상태의 신호값과 2상태의 신호강도
Table 1. 8-value and 2-strength.

신 호 값	사 용 기 호
0	0
1	1
0에서 1변화	U(up)
1에서 0변화	D(down)
unknown	X
고 임피던스 low	Z0
고 임피던스 gigh	Z1
고 임피던스 unknown	ZX

신 호 강 도	사 용 기 호
Driving 신호	0, 1, X, U, D
고임피던스 신호	Z0, Z1, ZX

이 신호값들 중에서 일반소자나 기능소자에는 5상태(0, U, 1, D, X)의 신호값이 사용되며, tri-state게이트에는 6상태(0, 1, X, Z0, Z1, ZX)의 신호값과 2상태의 신호강도가 사용된다. 여기에서 0, U, 1, D, X는 driving 신호가 되며, Z0, Z1, ZX는 고임피던스 신호가 된다.

LOSIM에서는 driving 신호는 fanout 단자를 구동시킬 수 있으나, 고 임피던스 신호는 fanout을 구동시킬 수 없다는 가정을 채택하여 시뮬레이션을 행한다. Z0, Z1, ZX의 신호가 일반소자나 MOS 회로의 콘트롤 게이트의 입력에 인가되면, 이들은 각각 논리 0, 1, X로 계산되며, U와 D는 X로 취급된다.

Ⅲ. 회로망 모델링

논리 시뮬레이션을 위한 회로의 모델링은 회로를 구성하는 각 소자들간의 상호 연결관계를 말하며, 모델링을 행하는 이유는 시뮬레이션 결과와 실제 회로의 결과가 일치되도록 하기 위함이다.[12]

일반적으로 논리 시뮬레이션에 있어서 소자들는 게이트 레벨, 기능 레벨 그리고 트랜지스터 레벨로 나누어서 모델되며, 모델링은 다음의 세가지 요소를 고려하여 행하게 된다.

1) 실제회로에 있어서 소자의 형태
2) processing 시스템에 있어서 소자의 형태
3) 기본적인 processing 알고리듬

본장에서는 일반소자 및 기능소자, transmission게이트 그리고 tri-state 게이트의 회로망 모델링 및 회로망 기술에 대해서 설명한다.

1. 일반소자 및 기능소자

일반소자 및 기능소자에서는 5 상태가 사용되며, 5 상태 신호모델을 이용하면 U와 D의 변화를 허용하므로 3 상태 모델을 이용하는것 보다 정확한 결과를 얻을 수 있다.

그림 1(a)는 AND 게이트에 대한 심볼을 나타내며, 1(b)는 이 회로에 대한 회로망 기술을 나타낸다. 그림 1(b)의 C는 게이트의 출력단자, AND는 게이트의 종류, 3과 4는 각각 상승시간, 하강시간 그리고 A와 B는 입력단자를 나타낸다. OR 게이트와 Exclusive-OR 게이트와 같은 회로의 회로망 기술도 위의 경우와 마찬가지이다. 그러나 그림 2과 같은 기능소자인 flip-flop의 회로망 기술은 다음과 같이 표시해야 하며, 이들은 flattening을 시키지 않고 black box로서 시뮬레이션이 가능하다.

그림 2(b)의 회로망 기술에서 Q는 flip flop의 출력단자, JK-FF은 소자명, r f은 각각 상승시간 및 하강시간을 나타낸다. CK는 클락 펄스, J와 K는 입력단자 그리고 SB와 RB는 제어클락을 나타낸다. 그리고 출력Q와 QBAR는 출력으로서 '/'에 의해서 분리되어야 한다.

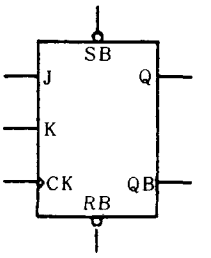
(a)

```
C AND 3 4 A B
```

(b)

그림 1. 일반소자
(a) AND 게이트의 심볼
(b) 회로망 기술

Fig. 1. General element.
(a) symbol of ANDgate.
(b) circuit description.



(a)

```
Q JK-FF r f CK J K SB RB/Q QB
```

(b)

그림 2. 기능소자
(a) JK-FF의 심볼
(b) 회로망 기술

Fig. 2. Functional element.
(a) symbol of JK-FF.
(b) circuit description.

2. Transmission 게이트

MOS transmission 게이트들은 Boolean 연산자로 표시할 수 없는 기능들을 취급하기 위하여 사용된다. 3 상태(0, 1, X)에 임피던스 상태를 추가한 4 상태 모델링 방법[11]과 2 논리값(0, 1)에 Z0, Z1, ZX를 사용한 transmission 게이트 모델 방법이[5]에 의해 제안되었다. 그러나 이 방법들은 양방향 transmission 게이트인 경우 고 임피던스 상태에서 0, 1, X를 효과적으로 취급할 수 없다.

여기에서는 6 상태의 신호값과 2 상태의 신호강도를 이용하여 단방향 transmission 게이트와 양방향 transmission 게이트를 효과적으로 시뮬레이션할 수 있는 모델링에 대해서 설명한다.

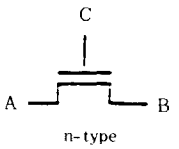
1) 단방향 Transmission 게이트

Transmission 게이트는 기본적으로 콘트롤 스위치가 on될 때 회로가 연결되고 off 될 때 회로가 차단되는 스위치 회로이다.

만약 콘트롤 스위치가 on이면 회로의 출력은 입력 신호값으로 전달되며, 다음 시간에서 스위치가 off로 전환되면 출력측은 고 임피던스 상태를 유지하게 된다. 최근의 시뮬레이터들은 MOS의 출력에서 charge leakage를 시뮬레이션하는 것과, 출력상태가 decay되지 않도록한 상태에서 시뮬레이션하는 두가지 형태가 있는데,[5] 본 연구에서는 후자의 경우를 택하여 모델링을 행한다. 단방향 MOS transmission게이트에 대한 회로는 그림 3과 같으며, 이에 대한 진리표는 표 2과 같다.

표 2. 단방향 transmission 게이트의 진리표
Table 2. Truth table for unidirectional transmission gate.

A \ C	0	1	X
0	*Z	0	X
1	*Z	1	X
X	*Z	X	X
Z0	*Z	Z0	X
Z1	*Z	Z1	X
ZX	*Z	ZX	X



B NTRG r f C A

(a)　　(b)

그림 3. 단방향 Transmision gate
(a) 단방향 transmission gate의 심볼
(b) 회로망 기술

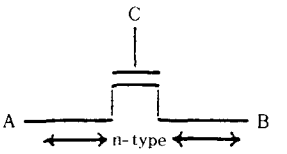
Fig. 3. Uni-directional transmission gate.
(a) Symbol for uni-directional transmission gate.
(b) Circuit description.

진리표에서 A는 게이트의 입력신호를, C는 콘트롤 게이트를 나타낸다. 이 표에서 만약 콘트롤 게이트가 0인 경우의 출력값(여기서는 *Z로 표시됨)은 입력값의 변화와 관계없이 이전 상태에 따라서 결정된다.[6] 만약 콘트롤 스위치가 0이면 출력값은 입력값과 관계없이 이전에 charge 되었던 값에 의해 좌우되고, 콘트롤 스위치가 1이면 출력값은 입력값에 의해서 결정된다.

예를 들어서, 이전의 한 시점에 있어서 임의의 출력값이 1인 경우, 콘트롤 게이트가 0으로 변화 했다면, 이때의 출력값은 Z1이 된다. 여기서 NTRG는 NMOS형 transmission 게이트를 나타낸다. 그리고 PMOS인 경우에는 PTRG를, CMOS인 경우에는 CTRG를 사용하면 된다.

2) 양방향 Transmission 게이트

콘트롤 스위치의 양단자(drain, source)값이 동일한 경우에는 문제가 없으며 이런 경우 양단자는 동일한 값으로 구동된다. 그러나 게이트의 양단자값이 서로 다를 경우에는 양단자의 전압상태는 불분명하게 된다. 예를 들어서 그림 4와 같은 양방향 transmission 게이트에 대해서 생각해 보자. 만약 콘트롤 스위치 C가 1인 상태에서 A와 B의 값이 서로 다르다고 가정하면, 일반적인 경우에 있어서 양단자의 값A, B는 X로 처리된다.



A BINTRG r f C B

(a)　　(b)

그림 4. 양방향 Transmission gate
(a) 양방향 transmission gate의 심볼
(b) 회로망 기술

Fig. 4. Bi-directional transmission gate.
(a) Symbol for bi-directional transmission gate.
(b) Circuit description.

그러나 이 경우 6상태의 신호값과 2상태의 신호강도를 이용하면 표3에 나타난 바와 같이 X값이 아닌 출력값을 얻을 수 있으므로 transmission 게이트의 상태를 효과적으로 시뮬레이션 할 수 있다. 표3에서 A, B는 게이트의 양단자를, C는 콘트롤 게이트를 나타낸다. 이 경우 출력값은 논리신호의 세기에 의해서 결정되며, 논리신호 세기의 순서는 driving 신호(입력신호)>고 임피던스 신호이다.

만약, 콘트롤 게이트가 1이고, 양단자 값이 driving이거나 고 임피던스이면 가장 최근에 발생한 신호값이 이전의 값을 overriding하게 된다.[6] 예를 들어서 콘트롤 게이트가 1이고, 이전의 한 시점에서 한쪽 단자 A가 1인 상태에서 다른 단자B가 0으로 event가 발생했다고 가정하자. 이 경우에 있어서단자 B의 값1은 단자B의 값1을 overriding하게 되므로 출력값은 0이 된다. 그림4(b)에서 BICTRG는 CMOS형 transmission 게이트를 말하며, PMOS형 및 NMOS형 transmission 게이트인 경우에는 각각 BIPTRG 및 BINTRG를 사용한다.

3. Tri-state 게이트

Tri-state의 출력은 Z상태에서도 0, 1, X와 같은 값들을 가질 수 있기 때문에 4상태(1, 0, X, Z) 값만을 이용하여 시뮬레이션할 경우, 완전한 시뮬레이션 결과를 기대할 수 없다.[15] 예를 들어서 그림5와 같

표 3. 양방향 transmission 게이트의 진리표
Table 3. Truth table for bi-directional transmission gate.

A B \ C	0	1	X
0 0	0 0	0 0	0 0
0 1	0 1	0 0	0 X
0 X	0 X	0 0	0 X
0 Z0	0 Z0	0 0	0 X
0 Z1	0 Z1	0 0	0 X
0 ZX	0 ZX	0 0	0 X
1 0	1 0	1 1	1 X
1 1	1 1	1 1	1 1
1 X	1 X	1 1	1 X
1 Z0	1 Z0	1 1	1 X
1 Z1	1 Z1	1 1	1 X
1 ZX	1 ZX	1 1	X X
X 0	X 0	X X	X X
X 1	X 1	X X	X X
X X	X X	X X	X X
X Z0	X Z0	X X	X X
X Z1	X Z1	X X	X X
X ZX	X ZX	X X	X X

A B \ C	0	1	X
Z0 0	Z0 0	0 0	X 0
Z0 1	Z0 1	1 1	X 1
Z0 X	Z0 X	X X	X X
Z0 Z0	Z0 Z0	Z0 Z0	X X
Z0 Z1	Z0 Z1	ZX ZX	X X
Z0 ZX	Z0 ZX	ZX ZX	X X
Z1 0	Z1 0	0 0	X 0
Z1 1	Z1 1	1 1	X 1
Z1 X	Z1 X	X X	X X
Z1 Z0	Z1 Z0	ZX ZX	X X
Z1 Z1	Z1 Z1	Z1 Z1	X X
Z1 ZX	Z1 ZX	ZX ZX	X X
ZX 0	ZX 0	0 0	X 0
ZX 1	ZX 1	1 1	X 1
ZX X	ZX X	X X	X X
ZX Z0	ZX Z0	ZX ZX	X X
ZX Z1	ZX Z1	ZX ZX	X X
ZX ZX	ZX ZX	ZX ZX	X X

(이 표는 A의 신호값이 B의 신호값 보다 더 강할 경우를 나타낸다.)

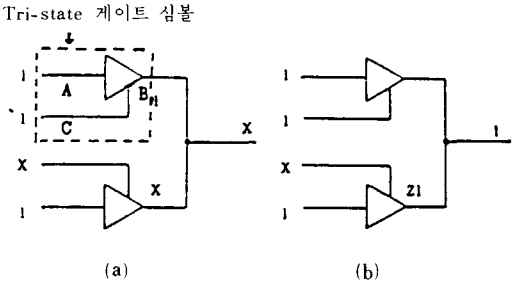


그림 5. Wired-or tri-state 게이트
(a) 4 상태 시뮬레이션
(b) 6 상태 시뮬레이션
Fig. 5. Wired-or tri-state gate.
(a) 4-value simulation.
(b) 6-value simulation.

은 wired-or tri-state 게이트에 대해서 생각해 보자.

그림 5에서 콘트롤 게이트가 X인 경우 (a)의 경우 출력은 X가 되면 (b) 경우의 출력은 1이 된다. 즉 4 상태(0, 1, X, Z)의 경우보다 6 상태(0, 1, X, Z0, Z1, ZX)를 사용하여 시뮬레이션 하는것이 보다 정확하다는 것을 알 수 있다. 그림 5 중의 tri-state 게이트 심볼에서 A, C, B는 각 입력단자, 콘트롤 게이트 그리고 출력단자를 표시하며, Tri-state 게이트의 소자 평가는 표 4와 같은 진리표를 이용한다. 회로망 기술방법은 단방향 transmission 게이트의 경우와 같다.

표 4. Tri-state 게이트의 진리표
Table 4. Truth table for tri-state gate.

A \ C	0	1	X
0	ZX	0	Z0
1	ZX	1	Z1
X	ZX	X	ZX
Z0	ZX	Z0	Z0
Z1	ZX	Z1	Z1
ZX	ZX	ZX	ZX

Ⅳ. 시뮬레이션 알고리듬 및 데이타 구조

신호값의 변화로서 표현되는 event는 time-based 시뮬레이터에서 사용되는 기본적인 개념으로서, 전체 회로에 대한 신호선의 event 비율은 약 2~10 % 정도이다. 만약 어떤 회로의 한 node에서 event가 발생했다면, 전체회로중에서 event에 의해서 구동되는 fanout node들만을 선택하여(selective trace) 소자를 평가하면, 아주 빠른 시간내에 각 node에 대한 신호값을 얻을 수 있는데, 이와같은 방식을 selective trace, event-driven 시뮬레이션방식이라 한다.

Event-driven 시뮬레이션 방식에 있어서, time queue 및 table-driven 표현에 대한 데이타 구조는 그림 6과 같다. Time queue는 EVENT-LIST와 결합하여 event를 scheduling 하는데 사용된다. 그리고 table-driven 표현은 NETLIST와 FANLIST로 구성되며, NETLIST는 회로망의 각소자에 대한 중요한 속성으로 구성되고, FANLIST는 소자의 fanin 및 fanout에 대한 속성을 갖는다. Time queue는 pointer array로서 실현되며, EVENT-LIST 대한 header를 포함한다. EVENT-LIST는 출력 핀의 식별번호, net 값, event 시간 그리고 다음의 EVENTLIST에 대한 link를 가지고 있다. NETLIST는 FANLIST에 대한 pointer 및 net의 식별번호, 현재 값, 이전 값, rise delay, fall delay 그리고 출력을 위한 node flag 등을 포함하고 있다.

LOSIM에서는 hazard 분석을 효과적으로 수행하기 위해서 그림 7과 같은 event-driven 시뮬레이션 알고리듬을 사용한다.

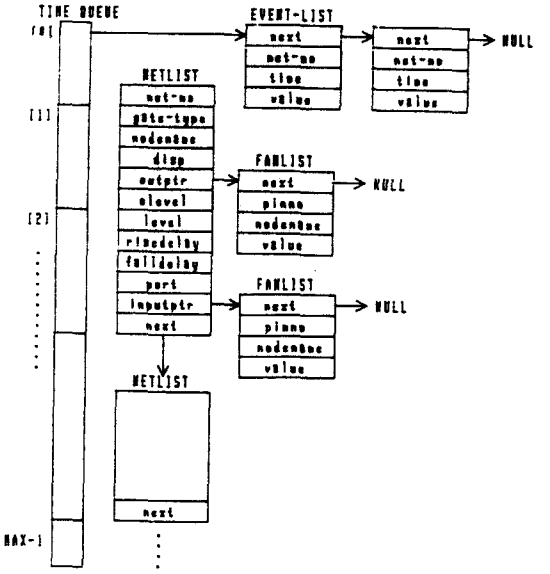


그림 6. Time queue와 table-driven 표현
Fig. 6. Time queue and table-driven representation.

```
Procedure Event_Driven
begin
(1)  initialize SARRAY to 1 for the hazard analysis;
(2)  set the current time equal to the initial time;
(3)  while the current time does not exceeds the final time do
       begin
(4)      for all current event do
           begin
(5)          take the event from event list;
(6)          update the value of the net;
(7)          if SARRAY is equal to current time then
                set SARRAY equal to 0;
(8)          for all fan-out do
               begin
(9)              get the output value and state
                 by the element evaluation;
(10)             if output value was changed for the first time
                     then  set SARRAY equal to 0;
(11)             if output value was changed then
                   begin
(12)                 if SARRAY is equal to 0 then
                       begin
(13)                     calculate the scheduling time
                         of the future time;
(14)                     insert this output value into the
                         time queue at scheduling time;
(15)                     scheduling time is placed in SARRAY;
                       end
(16)                 else perform hazard analysis;
                   end
               end
           end
(17)       increase the current time by one
       end
   end
```

그림 7. 시뮬레이션 알고리듬
Fig. 7. Simulation algorithm.

그림 7의 알고리듬에 있어서, 현재 시간을 시작 시간으로 하여 현재 시간이 마지막 시간을 초과하지 않는 범위내에서 시뮬레이션을 행한다. 현재시간에 등록된 각각의 event에 대하여 event list로 부터 event를 꺼내고, net 값을 갱신한다.

3번째 loop에 있어서, 모든 fanout 단자에 대하여, 소자 평가를 행한 다음 출력값 및 출력상태를 얻는다. 이때 출력이 변화한 net에 한하여 scheduling time을 계산하고 scheduling을 행한다. 현재시간을 1로 증가시키면서 마지막 time step까지 위의 과정을 반복한다.

그림 7의 알고리듬에서 (1), (7), (12), (15) 그리고 (16)은 hazard 검출을 위한 routine이며, hazard 검출에 대한 알고리듬은 V장에서 설명한다.

V. Hazard 검출 알고리듬

회로의 지연에 의해서 조합회로는 순간적으로 비정상적인 값(error 또는 spike)을 발생하는 경우가 있는데, 이것을 hazard라 한다. hazard에는 static hazard와 dynamic hazard로 구분할 수 있는데, 자세한 사항은 문헌[15]에서 기술하였다. Hazard검출은 신호값중의 U상태 및 D상태 그리고 time queue를 이용하며, hazard가 검출되는 조건은 그림 7에 나타난 알고리듬에 의해서 수행된다. 그림 7의 알고리듬에서는 (1), (7), (9), (10)을 수행하여 hazard 검출 준비를 한다. (12)에서, SARRAY가 0이면 정상적으로 scheduling을 행하고, SARRAY가 0이 아니면 hazard 분석을 수행한다. Hazard분석 알고리듬에 대해서 논하기 전에 hazard분석이 수행되는 조건에 대해서 알아보자. 그림 8에서 (a)는 NAND 게이트를 나타내며, (b)는 이 게이트에 대한 타이밍 차아트를 나타낸다. 이 게이트의 상승지연 및 하강지연은 공히 4 *time unit*를 가진다고 가정한다. 타이밍 차아트에서 입력 패턴 A가 충분한 시간을 가지기전에 time 10에서 event가 발생했기 때문에, 출력Z의 time 12에서 14까지 짧은 신호값 0을 가지게 되며 이것이 static hazard된다. 이와같은 hazard의 검출은 다음과 같은 절차에 의해서 수행된다.

(1) Time2에서, 단자A의 event에 의해서 출력값1과 출력상태U를 얻는다. 최초에는 SARRAY[Z] 가 0이므로 hazard분석을 행하지 않고 time 6으로 event scheduling을 행한다. SARRAY[Z]에도 6을 set 한다.

(2) Time 4에서는 출력값이 이전값과 동일하므로 scheduling을 행하지 않는다.

(3) Time 6에서는 SARRAY[Z]의 값과 현재 시간이 동일하므로 SARRAY[Z]의 값을 0으로 reset한다.[4] Time 7에서도 scheduling을 행하지 않는다.

(5) Time 8에서는 단자B의 event에 의해서 출력값 0

와 출력상태 D를 얻는다. 이때는 SARRAY[Z]의 값이 0이므로 hazard 분석을 행하지 않고 time12로 출력값 0을 scheduling한다. SARRAY[Z]에도 12를 set한다.

(6) Time10에서 단자A의 event에 의해서 출력값 1과 출력상태U를 얻는다. 이때는 SARRAY[Z]의 값이 0이 아니므로 시뮬레이션 알고리듬의 (16)에서 그림 9의 hazard의 분석 알고리듬을 적용한다. 이 알고리듬의 (3)에 의해서 (time6에서의 출력값(1)과 현재의 출력값(1)이 같으므로) time 12에서의 Spike는 Static hazard가 된다.

(7) Hazard 검출이 수행된 후 time10에서의 출력값 1은 (5), (6), (7)에 의해서 scheduling되고, (8)에 의해서 SARRAY[Z]는 0으로 reset 된다.

그림 9의 알고리듬에 있어서 이미 scheduling 되었던 동일 net(여기서는 C단자임)의 출력상태가 U이고, 이때의 신호값 A, B가 서로 다르므로, 이미 scheduling 되었던 time(여기서는 25가 됨)에서 dynamic hazard가 생성된다. 만약 신호값 A, B가 같을 경우에는 static hazard가 생성된다.

Time22에서 future scheduling time(current time + delay of the gate)을 계산하고, time27에서의 event list로 새로운 출력값 0를 등록한다.

Time25의 입력변화에 의해서 time27로 이미 scheduling 되었던 출력값 1을 unknown으로 대치시킨다.

다시 SARRAY를 0로 하여 다음 단계에서 hazard 검출준비를 한다.

Time27 이후에서도 위와 같은 방법을 이용하면 time55에서 hazard가 검출된다. Hazard 분석을 위한 전 과정이 끝나면 그림 7의 첫단계 부터 최대시간까지 위의 과정을 반복한다.

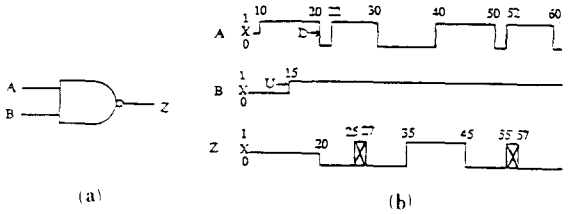


그림 8. NAND 게이트에 대한 hazard 분석
(a) NAND 게이트의 심볼
(b) 타이밍 차아트

Fig. 8. Hazard analysis for NAND gate.
(a) Symbol for NAND gate.
(b) Timing chart.

```
Procedure Hazard_Analysis
begin
(1)  if SARRAY is not equal to 0 then
       begin
(2)      if scheduled output state is equal to "U" .OR.
            scheduled output state is equal to "D" then
            begin
(3)           if first stable value is not equal to second stable value then
                 static hazard exists at scheduled time already;
(4)           else dynamic hazard exists at scheduled time
                 already;
            end
(5)      calculate the scheduling time of the future;
(6)      insert a new output value into event list of time queue;
(7)      replace the scheduled old value with unknown value;
(8)      SARRAY is set to 0;
       end
end
```

그림 9. Hazard 검출 알고리듬

Fig. 9. Hazard detection algorithm.

VI. 시뮬레이션 결과

본 논문에서 제안한 알고리듬을 프로그램으로 실현하여 그림10과 같은 statie RAM cell 회로를 시뮬레이션한 결과 그림11과 같은 결과를 얻을 수 있었다.

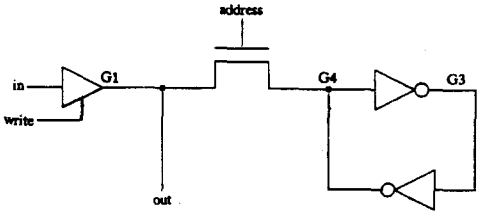


그림10. 예제회로 1 (static RAM cell)

Fig. 10. Example circuit 1(Static RAM cell).

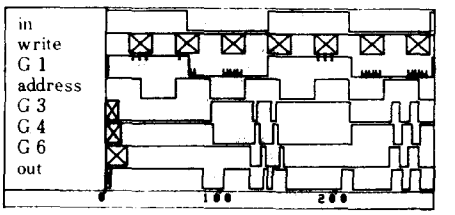


그림11. Static RAM cell의 시뮬레이션 결과

Fig. 11. Simulation result for static RAM cell.

여기에 사용된 모든 소자의 delay는 rise time과 fall time이 1인 unit delay를 이용하였다. 결과의 time20에서 write 신호가 X인 경우, 출력값 G1은 I(고 임피던스 1)가 되고, 이 신호값은 G4의 출력값(1) 보다 약하므로 out에서는 결과가 1인 출력값을

얻을 수 있다. 시뮬레이션 결과에서 H, I는 각각 Z0, Z1과 대응된다. 그림10에서 G1과 out node의 경우, 실제의 회로에 있어서는 동일한 node로서 취급되나 (프로그램 내부에서는 서로 다른 node(점선 부분)로서 취급됨), 여기에서는 논리 strength에 대한 이해를 돕기 위해서 G1 node를 삽입하여 시뮬레이션을 행하였다.

다른 예로서, 그림12(a)와 같은 비동기 순서회로를 이용하여 실험을 행하였으며, 실험결과는 (c)와 같다. 여기서 NAND 게이트의 rise time과 fall time은 공히 5로 하며, NOT 게이트는 공히 3으로 한다. 지금, 그림12(c)의 T를 입력으로 하여 시뮬레이션을 행하면 S2에서는 Spike를 포함한 (time 50에서 53) 출력값을 얻게 된다. 이것은 S1과 S2의 지연에 의해서, 게이트E (time 45)와 게이트 D (time 48)의 신호값에 의해서 발생된 것이다. 또한 단자D (time 55 에서 58) 및 S1(time 60에서 63)의 spike는 S2에 의한 것이다. 이때, 시뮬레이터는 hazard가 발생한 시각과 Net명을 사용자에 통보하게 된다. 이 통보에 의해서 IC설계자는 그림12(b)와 같이 hzard를 제거한 회로설계가 가능하게 된다.

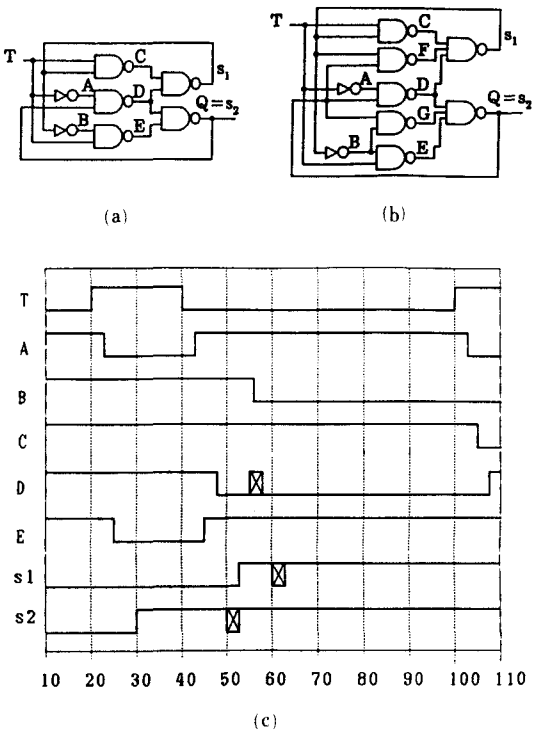


그림12. 예제회로 2
(a) hazard를 가진 비동기 순서회로
(b) hazard를 제거한 회로
(c) (a) 회로에 대한 타이밍 차아트

Fig. 12. Example circuit 2.
(a) Asynchronous sequential circuit with hazard
(b) Hazard-free implementation.
(c) Timing chart of (a).

Ⅶ. 결　　론

Assignable delay 모델을 이용하여 게이트 레벨 및 트랜지스터 레벨의 회로를 효과적으로 시뮬레이션 할 수 있는 프로그램인 LOSIM을 개발하였다.

본 논문에서는 8상태(0, U, 1, D, X, Z0, Z1, ZX)와 2상태의 신호강도(driving, high-impedance)를 이용하여, 게이트 레벨, 기능 레벨, tri-state 게이트 그리고 transmission 게이트를 효과적으로 시뮬레이션 할 수 있는 모델을 제안하였다. 특히, 양방향 transmission 게이트는 6상태와 2상태의 신호값를 이용하여 모델링을 행하므로써[5,6] 보다 정확한 결과를 얻을 수 있었다. 또한, 회로 지연에 의해서 발생될 수 있는 static hazard와 dynamic hazard를 정확히 검출할 수 있는 알고리듬을 제안하여 시뮬레이션 결과를 고찰하였다. Hazard 검출은 시뮬레이션 command "hazard에 의해서 수행되며, default는 "hazard off"이다.

시뮬레이션 알고리듬은 selective trace에 의한 event-driven 시뮬레이션 방식을 사용하였다. 제안된 알고리듬들은 C언어로 SUN-3/160 workstation 상에서 구현하였으며, 사용자의 편의를 위하여 자유입력 형식을 사용할 수 있도록 하였다.

參 考 文 獻

[1] Glenn R. CASE and Jerry D. Stauffer, "SALOGS-IV: A program to perform logic simulation and fault diagnosis," *Proc. 19th Design Automation Conference*, pp. 392-397, 1978.

[2] M.A. d'Abreu, "Gate-level simulation," *IEEE Design & Test*, pp. 63-71, Dec. 1985.

[3] I. Shirakawa, H. Mizuno, M.S. Kang, "A pipeline scheme for dogic simulation," *Proc. European Conference on Circuit Theory and Design*, pp. 687-692, 1987.

[4] Randal E. Bryant, "MOSSIM: A switch level simulator for MOS LST", *Proc. 18th Design Automation Conference*, pp. 786-790, 1981.

[5] D. Holt, D. Hutchings, "A MOS/LSIT oriented logic simulator," *Proc. 18th Design Automation Conference*, pp. 280-287, 1981.

[6] R.M. McDermott, "Transmission gate modeling in an existing three-valued simulator," *Proc. 19th Design Automation Conference*, pp. 678-691, 1982.

[7] M.A. d'Abreu et al, "ORACLE- A simulator for bipolar and MOS IC design," *Proc. 21th Design Automation Conference*, pp. 343-349, 1984.

[8] T. Takahashi, H. Fukuda, "A logic simulation technique for gate transistor level circuits with precise delay estimation," *IEEE International Conference on Computer Aided Design*, pp. 200-202, 1984.

[9] 강민섭, 김욱현, 이철동, 유영욱 "EDAS_ P 시스템에서의 Gate Level Logic Simulator 개발," 대한전자공학회 추계종합학술대회 논문집, vol. 9, no. 2, pp. 935-938, 1986.

[10] P.L Flake, P.R. Morby, G. Musgrave, "An algebra for logic strength simulation," *Proc. 20th Design Automation Conference*, pp. 615-618, 1983.

[11] A.K. Bose, S.A. Szygenda, "detection of static and dynamic hazard in logic nets," *Proc. 14th Design Automation Conference*, pp. 220-224, 1977.

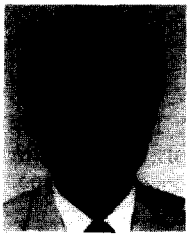
[12] M.A. Breuer, A. Friedman, Diagnosis and Reliable Design of Digital System, Computer Science Press, Woodland Hills, CA, 1976.

[13] E.W. Thompson, S.A. Szygenda et al., "Timing analysis for digital fault simulator using assignable delays," *Proc. 11th design Automation Confernece*, pp. 266-272, 1974.

[14] HILO-3 Reference Manual, GenRad, 1985.

[15] 강민섭, 시라가와 이사오, 이철동, 유영욱, "Assignable Delay를 이용한 논리 시뮬레이션의 hazar검출," 대한전자공학회 추계학술대회, nol. 10, no. 1, pp. 683-686, 1987. *

著 者 紹 介



康 敏 燮(正會員)

1979年 광운대학교 무선통신공학과 졸업. 1984年 한양대학교 전자공학과 석사학위 취득. 1988年 ~ 현재 일본 오오사카대학 전자공학과 박사과정 재학중. 1984年~1988年 한국전자통신연구소 연구원. 1986年~1987年 오오사카대학 연구원. 주관심분야는 Logic and timing simulation, VLSI Testing, Design for testability, Fault modeling and Simulation 등임.

◆

李 哲 東 (正會員) 第26卷 第4號 參照
현재 전자통신연구소 자동설계기기연구실 실장

柳 瑛 昱(正會員)

1946年 6月 2日生. 1973年 한양대학교 전자공학과 졸업 학사학위 취득. 1975年 8月 한국과학기술원 전기 및 전자공학과 졸업 석사학위 취득. 1975年 1月~1977年 1月 한국과학기술연구소 반도체 기술센터 연구원. 1977年 1月~1981年 10月 한국전자기술연구소 선임연구원 설계실장. 1981年 10月~1986年 5月 한국전자기술연구소 미국사무소장. 1985年 5月~1986年 12月 한국전자통신연구소. 현재 벨리드 로직 시스템즈 코리아 지사장.