Prolog를 이용한 논리회로의 기능적 시뮬레이션

○김 종 성, 조 순 복, 박 홍 준, 임 인 칠
한 양 대 학 교

Functional Simulation of Logic Circuits by Prolog

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

ABSTRACT

This paper proposes a functional simulation algorithm which decrease the internal memory space and run time in simulation of VLSI.

Flip-flop, register, ram, rom, ic and fun are described as functional elements in the simulator. Especially icf is made as new functional element by combining the gate and the functional element, therefore icf is used efficiently in simulation of VLSI.

The proposed algorithm is implemented on PC-AT(MS-DOS) in by Prolog-1.

I. 서 론

VLSI기술이 발전함에 따라 논리회로 설계시 정확한 검증이 요구되고 있으며, 검증 방법으로 시뮬레이션이 널리 사용되고 있다. 시뮬레이션은 회로 레벨, 타이밍 레벨, 논리 레벨, 레지스터 전달 레벨 등으로 구분한다[7].

논리 레벨 시뮬레이션은 입출력값의 관계와 시간지연을 모델링하여 회로가 정상동작 하는가를 검증한다. Facts, rules, questions로 구성된 PROLOG(PROgramming LOGic)[8]언어는 강력한 패턴조합 능력과 자동 백트래킹(backtracking)기능을 갖고 있으며, nonprocedural한 프로그램을 구성하므로 시뮬레이션에서 사용하기에 편리하고 알고리즘을 간단히 기술 할 수 있다. 또한 논리 문제를 쉽게 해결하는 장점을 가지고 있으므로, 논리회로 시뮬레이션에 Prolog의 사용이 증가되고 있다. Prolog를 사용하여 논리회로를 시뮬레이트 하기 위해서는 게이트를 동작 시키기 위하여 필요한 논리값 선언 rule, 지연을 고려한 게이트 모델링 rule, 신호선 모델링 rule, 시뮬레이션을 수행하기 위한 rule로 구성한다. Rule들로 구성된 게이트 레벨 시뮬레이션은 정확한 결과는 얻을 수 있으나 대규모 회로의 기술이 용이하지 않고, 논리회로 특성에 알맞은 기능적 표현이 어렵다. 또한 각각의 게이트 단위로 수행되므로 내부 기억 용량과 수행시간이 증가한다. Nam Sung. W[1]은 기능 모델을 사용하였으나 게이트 레벨로 동작하므로 회로 기술이 어렵고 수행시간이 길며, M.Malek[4]은 기능 모델에서 지연을 고려하고 있으나 한개의 출력을 갖는 회로로 분할하여야 하므로 대규모 회로에서는 적용이 어렵다.

본 논문에서는 기능레벨을 갖는 flip-flop, register, ram, rom, icf, fun등 기능소자를 기본단위로 기술하며, 특히 ich는 게이트와 기능소자를 하나의 상태로 묶어 새로운 기능소자로 만들므로 대규모 회로 기술이 용이하고, 시뮬레이션 수행시 내부 기억 용량과 수행시간이 빠른 기능적 시뮬레이션 알고리즘을 제안한다. 기능소자의 논리값과 시간지연은 게이트 레벨 시뮬레이션과 물리적 고찰을 이용하여 다입력 상태에 대한 다출력의 정확한 논리값과 시간지연 결과를 얻어 고려하고, 이것을 기능소자 기본단위로하여 시뮬레이션 한다.

II. 계층구조와 논리상태 논리값

기능적 시뮬레이션을 위한 계층구조는 6가지 모드로 구성되며 그림 1에 나타내었다. 화일 조작 모드는 시뮬레이션할 화일의 수정 또는 편집 기능이 있다. 화일 선택 모드에는 필요한 데이타 화일을 시뮬레이터 내로 읽어들인다. 시뮬레이션 모드는 시뮬레이션 결과를 화면에 표시한다. PROLOG호출 모드는 Prolog술어를 실행시킬 수 있다. 시스템 호출 모드는 시스템(MS-DOS)명령어를 실행한다.

논리상태와 논리값은 데이타 베이스에 기술하며 표1에 나타내었다.

표1. 논리 상태와 논리값

논리상태	논 리 값
1, 0	1,0, x = {0,1}
1, z, 0	1,z,0, t = {1,z}, b = {z,0}, x = {1,z,0}
1,h,z,1,0	1,h,z,1,0, x = {1,h,z,1,0}, p = {1,h}, r = {h,z}, f = {z,1}, n = {1,0}, t = {1,h,0}, w = {h,z,1}, b = {z,1,0}, u = {1,h,z,1}, d = {h,z,1,0}

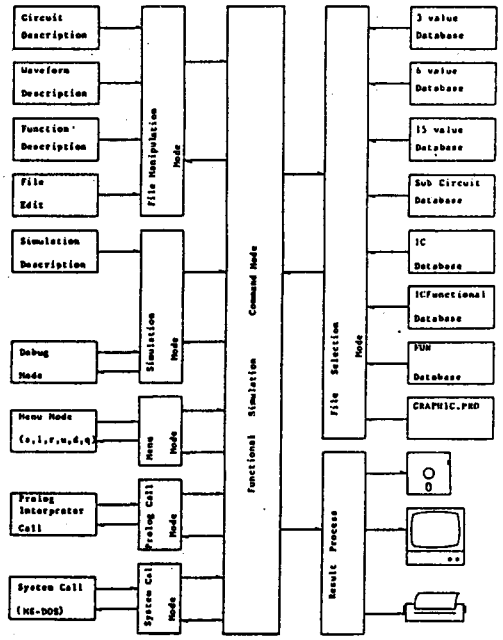


그림 1. 기능적 시뮬레이션의 계층구조

Ⅲ. 기능적 모델링

Prolog를 이용한 기능적 모델링은 모델링의 확장을 용이하게 하고, 게이트 레벨 시뮬레이션에 어려운점(edge-sensitive devices)에 대하여 쉽게 시뮬레이션을 가능하게 한다. 또한 시뮬레이션 효율을 증가시키며, 회로를 상세한 수준으로 표현하고, 기능 단위로 쉽게 기술하게 한다.

시뮬레이터는 4부류(wire exprssion, functional primitives, event statement, functional statement)의 기능적 모델링을 가지고 있으며 각 기능적 모델링은 표2와 같다.

표2. 기능적 모델링

wire	wire expression
register, ram, rom	functional primitive
when, next, case, if loadif 0, loadif 1, ifnot	event statement
icf fun	functional statement

기능적 모델링은 적용에 있어서 각각 서로 상호 호환관계를 가지고 있다.

〔 wire, terms, = , wire-expression 〕,

〔 wire, delay, terms, = , wire-expression 〕

wire-expression에서 terms은 한개 또는 복수개의 vector 형식으로 되어있으며 wire-expression은 부울 함수식으로 구성되어 있다. 또한 wire-expression의 기능적 구성이 지연을 가지게 될때 wire-expression에 지연을 적용할 수 있다.

예) 〔 wire, sum, = , 〔a,xor,b,xor,c〕〕,

〔'wire,〔sum,1,4〕, = ,〔〔a,1,4〕,xor,〔b,1,4〕,xor〔c,1,4〕〕〕

wire-expression에서 사용할 수 있는 연산자는 표3에 있는 것과 같다.

표3. wire-expression 연산자

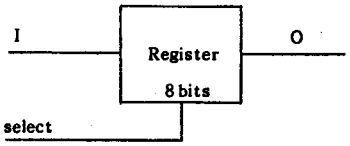
and, or, nand, nor, xor, xnor, same

register, ram, rom, 은 함수적 기본소자로써 내부적 상태를 하나의 모듈화 한 기본소자로 사용하고 있다.

〔 register,delay,c-type,set-wirelist,cond-wirelist,act-wirelist 〕

로 표시되며 c-type에는 case, loadif 0, loadif 1 등이 사용될 수 있고, 조건 cond-wirelist를 만족하면 act-wirelist를 set-wirelist의 값으로 인가한다.

예) register의 표현



〔 register,〔9,5〕, loadif 0,〔o,1,8〕,select,〔i,1,8〕〕.

그림 2. Register의 논리 표현

when, next, case, loadif 0, loadif 1, ifnot은 조건발생을 만족할때 특성 소자값을 변화시키는 것으로 event statement라하고, 시뮬레이션 수행중에 조건이 만족되면 action list를 실행한다.

〔 when,condition,do,actionlist 〕,

〔 next,condition,do,actionlist 〕,

〔 case,set-wirelist,cond-wirelist,act-wirewlist 〕,

〔 loadif 0,set-wirelist,cond-wirelist,act-wirelist 〕,

〔 loadif 1,set-wirelist,cond-wirelist,act-wirelist 〕,

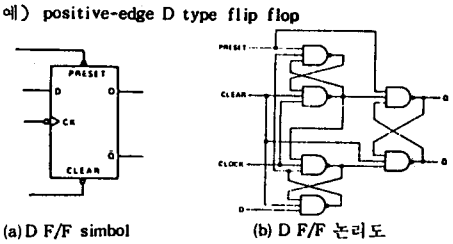
〔 if,ifcond,hen,actional,else,action 2 〕,

〔 ifnot,ifcond,then,actional,else,action 2 〕.

fun는 사용자 정의에 의하여 사용되는 함수적 표현으로 모델링 방법은 다음과 같다.

1) 상태변화에 의하여 결정되는 요인이 있는 입력단자는 이전 상태 값과 현재상태 값을 동시에 나타내고 있어야 한다.
2) 출력단자가 메모리 요소를 지니고 있는 경우는 이전상태 값과 현재상태 값을 동시에 나타내고 있어
3) 이외의 단자는 현재상태 값만 나타낸다.
4) 출력의 결정에 관여하지 않은 입력단자는 anominous〔_〕로 나타낸다.

예) positive-edge D type flip flop



(a) D F/F simbol (b) D F/F 논리도

그림 3. D type Flip-Flop

〈데이타 베이스에 기술〉

```
d-pf(-,[N,N],[Q,Q],[B,B]).
d-pf(0,[0,1],[-,0],[-,1]).
d-pf(1,[0,1],[-,1],[-,0]).
d-pf(-,[N,P],[Q,Q],[B,B]).
```

〈시뮬레이션 회로기술〉

```
[ dff1,[fun,d-pf],[10,20,30]],
        [d,[oidcp,precp],[oidg,preg],[oidgb, pregb]]
```

또한 fun에 의한 기술 방법은 사용자에 의하여 fun을 데이타 베이스에 기술하여 사용자 특성에 맞는 소자를 시뮬레이션할 수 있도록 확장할 수 있다.

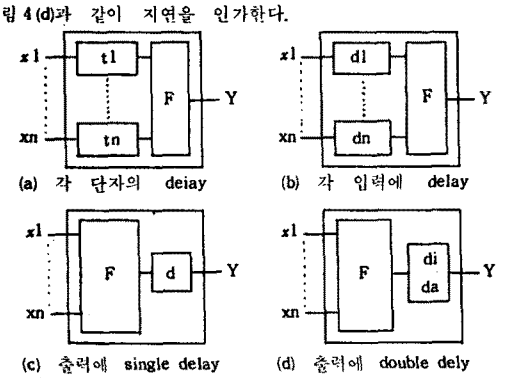
시뮬레이터는 fun으로 다음 표4와 같은 것이 정의되어 있다.

표4. fun list

d-nf	d type flip, flop
t-pf,t-nf	t type flip, flop
sr-1,sr-pf,sr-nf	sr type flip, flop
jk-1,jk-pf,jk-nf	jk type flip, flop

기능적 모델링에서 사용하고 있는 지연은 각 기능적 모델에 따라 적용된다. zero지연은 지연을 고려하지 않고 입력에 따라 출력이 그대로 나타나는 지연의 종류를 의미하고 물리적 고찰에 의한 지연은 설계시에 설계소자에 따라 나타나는 전기적 흐름의 지연이다. icf,fun의 지연은 다음 두가지로 나눌 수 있다. 첫째는 모듈의 출력측에 지연을 고려하는 것이며, 둘째는 모듈의 각각 입력측에 지연을 고려하는 것이다.

각각 입력에 의한 출력발생 지연시간 △tij그림4(a)을 구한 후 입력단자에 지연을 인가할 경우에 di=△ti 로 하여 그림4(b)와 같이 나타내고, 출력단자에 지연을 인가할 경우에 single 지연이면 그림4(c) d=Max△ti로 하고, doule 지연이면 d min=Min△ti, d max=Max△ti 로 그림4(d)과 같이 지연을 인가한다.



(a) 각 단자의 deiay (b) 각 입력에 delay

(c) 출력에 single delay (d) 출력에 double dely

그림4. 기능소자의 시간지연

기능적 모델링 icf(functional statement IC)를 적용한 TTL IC SN74163(SYNCHRONOUS 4-BIT COUNTERS)의 논리도를 그림5에 나타내었으며, 표현은 표5에 기술하였다.

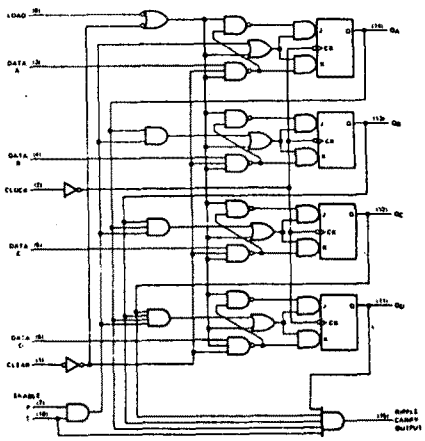


그림5. SN74163 SYNCHRONOUS 4-BIT COUNTERS

표5. SN74163의 icf에 의한 기술

```
sn74163([0,[0,1],-,-,-,-,0,0,
        1,0,[-,0],[-,0],[-,0],[-,0],0,1],
        [[0,0,0],[26,26,38]]).
sn74163([1,[0,1],A,B,C,D,O,O,
        O,O,[-,D],[-,C],[-,B],[-,A],0,1],
        [[17,17,25],[19,19,29]]).
sn74163([1,[0,1],-,-,-,-,1,0,
        1,1,[A,E],[B,F],[C,G],[D,H],P,1],
        [[13,13,20],[15,15,23]])
        :- state([A,E],[B,F],[C,G],[D,H],P).
state([0,1],[B,B],[C,C],[D,D],0).
state([1,0],[0,1],[C,C],[D,D],0).
state([1,0],[1,0],[0,1],[D,D],0).
state([1,0],[1,0],[1,0],[0,1],0).
state([1,0],[1,0],[1,0],[1,0],1).
```

Ⅳ. 시뮬레이션 알고리즘

논리 시뮬레이션 알고리즘은 다음과 같다.

단계1. 회로기술의 모든 요소에 대하여 입력단자, 신호선 출력단자의 관계를 데이타 베이스에 넣고 초기화 한다.

단계2. 파형기술의 초기조건과 event를 묶어 event-예정표를 만들고 빈 sched-예정표를 만든다.

단계3. 스케줄러에서 event-예정표와 sched-예정표 중 최근에 발생할 시간을 선택한다.

단계4. 현재시간표에 발생시간을 넣고 발생 논리값을 각 단자에 인가한 후 현재상태의 논리값을 현재논리표에 넣는다.

단계5. 스케줄러에서 발생한 시간에 활성화될 요소를 찾아 논리값과 시간을 묶어 논리실현으로 보낸다.

단계6. 요소에 논리값을 넣어 발생 논리값을 구하고 예정표에 의하여 sched-예정표에 지연이 고려된

발생논리값과 예정시간 요소를 기록한다.

단계 7. 단계 3에서 선택시간이 없을 때까지 단계 3에서 단계 6을 반복한다.

단계 8. 현재 시간표에 의하여 결과를 도형으로 나타낸다.

단계 9. Menu-Mode로 들어가서 사용자 입력을 기다린다.

단계 10. 입력조건을 실행한다. 만약 halt 또는 q이면 시뮬레이션을 끝낸다.

그렇지 않으면 단계 9로 간다.

시뮬레이션 알고리즘 수행에 대한 프로그램은 표6과 같다.

표6. 시뮬레이션 수행 rule

```
asim : -head, read-simfile(L), set-sim(N,L,C,W),
       check-sim, hdl(N,C,CL), wdl(N,W,WL),
       run(N,CL,WL,SL,T,D),
       disp(T,D), m-disp(T,D), end,/.
```

V. 시뮬레이션 예제

설계자에 의하여 설계된 논리 회로의 기능 시뮬레이션은 다음과 같다. 그림6은 POP VENDING MACHING CONTROL SYSTEM으로 회로기술에서 사용된 기능적 모듈들을 시뮬레이션 이전에 데이타 베이스에 기술되어 있어야 한다. 시뮬레이션의 실행 결과는 표7의 파형기술을 받아 그림7과 같이 화면에 도형으로 나타난다.

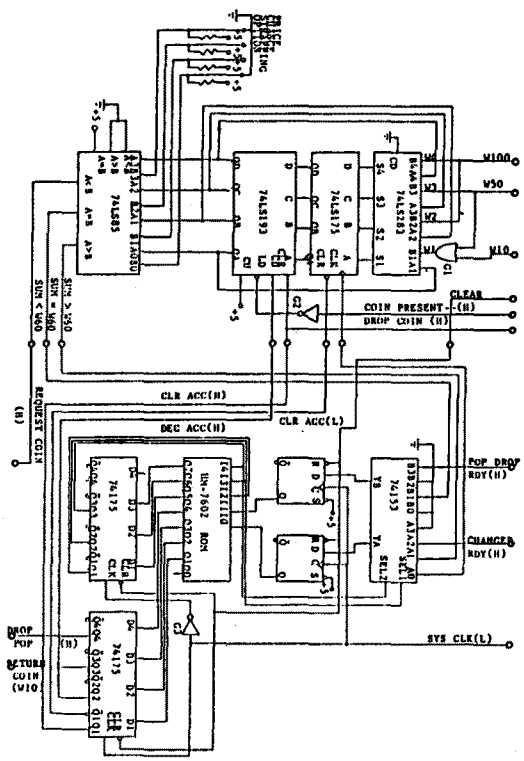


그림6. POP VENDING MACHINE CONTROL SYSTEM

표7. 파형기술

```
pop-wave( initial ([[w-100,0],[w-50,0],[w-10,0],[clear,0]]),
event([[w-100,[[clear,1]]],500,[[clear,0]]],[100,[[w-50,1]]],
      [1500,[[w-50,0],[w-10]]]],[2000,[[w-10,0]]]])).
```

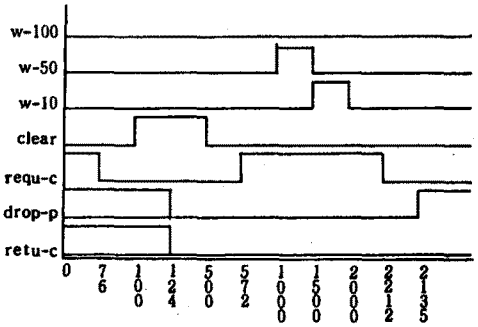


그림7. 결 과

VI. 결 론

본 논문에서는 게이트와 기능소자를 하나의 상태로 묶어 새로운 기능소자로 사용할 수 있는 icf를 제안하였으며 기본단위의 기능소자로 flip-flop, register, ram, rom, fun 등을 기술하여 시뮬레이션을 위한 대규모 회로 기술이 용이하게 하고, 내부 기억 용량과 수행시간이 감소하는 기능적 시뮬레이션 알고리즘을 제안하였다. 기능소자를 기본단위로 하여 시뮬레이션 하므로써 게이트 레벨로 시뮬레이션한 것과 같은 정확성을 유지하며 시간지연이 고려된 논리값을 얻을 수 있었으며, 시뮬레이션 수행시간을 감소할 수 있었다.

본 논문에서는 제안한 기능적 시뮬레이션 알고리즘을 PC-AT MS-DOS 상에서 PROLOG-1을 이용하여 실현하였다.

참 고 문 헌

1. Nam Sung Woo, "A Prolog Based Verifier for the Functional Correctness of Logic Citcuits," IEEE International Conference on Computer Design : VLSI in Computers, pp. 203-207, 1985.
2. F. Maruyama, M. Fujita, "Hardware Verification, "Computer, pp 22-32, Feb, 1985.
3. P.L. Flaka, P.R. Moorby, G.Musgrave, "An Algebra for Logic Strength Simulation, " 20 th Design Automation Conference, pp. 615-618, 1983.
4. Miroslaw Malek, Ajoy K. Bose, "Functional Simulation and Fault Diagnosis," IEEE 15 th Design Automation Conf, pp. 340-346, 1978.
5. Takao Uehara, Nobaki Kawato, "Logic Circuit Synthesis Using Prolog, "New Generation Computing, pp. 187-193, 1983.
6. 조순복, 김종성, 박홍준, 임인칠, "Prolog를 이용한 논리회로의 기능적인 검증," 한국정보과학회, 87봄 학술발표 논문집, pp. 291-295. 1987.