論 文

상태합성기 설계를 위한 상태축소 알고리듬 제안 및 그래픽 에디터 개발에 관한 연구

正會員 李 近 萬* 正會員 林 寅 七*

A Proposal of State Reduction Algorithm and the Development of a Graphic Editor for State Machine Synthesizer

Keun Man Yi* In Chil Lim* *Regular Members*

要 約

본 논문에서는 상태합성기 설계를 위하여 상태그래프를 자동으로 CHDL로 변환하는 그래픽 에디터를 개발하였다. 또한 FSM 합성과정 중에 상태테이블의 중복상태를 제거할 수 있는, 구성이 간단하고 빠른시간 내에 최소해를 구할 수 있는 상태축소 알고리듬을 제안하였다.

ABSTRACT

In this paper, we developed a Graphic Editor which automatically translated the state transition graph into state CHDL. Also, an algorithm for efficient state minimization is presented to reduce the redundancy state of state transition table.

Ⅰ. 서 론

최근 VLSI 기술의 발전과 더불어 회로의 복잡도가 증가함에 따라, CAD 툴(tool)을 이용한 설계자동화 연구가 활발히 진행되고 있다.

특히, 디지틀 시스템의 제어부분에 사용되는 큰 규모의 순차회로인 경우는 상위단계에서의 기술분으로부터 FSM(Finite Synthesis Machine)을 자동으로 설계할 수 있는 툴이 요구된다.

디지틀 시스템의 동작특성을 기술하는 HDL (Hardware Description Language)은, 제어회로를 기술하는 언어와 조합논리회로를 기술하는 언어로 분리되는바, 제어회로의 기술언어는 상태그래프를 기술할 수 있는 FSM(Finite State Machine) 형태로 기술되어야만 한다.[1]

FSM(Finite State Machine)의 설계는 FSM 합성용 툴 즉, 상태합성기(State Machine Synthesizer)를 필요로 하며, FSM 합성절차는 FSM용 HDL로 부터 기술된 순차회로를 구문분석(parsing)하여 기호상태표(Symbolic state table)을 추출한 후, 추출된 상태표를 이용하여 상태최소화와 상태할당을 수행하는 순서로 진행된다.

그러나, 대부분의 FSM 합성용 툴은 HDL 기술문으로 부터 구해지는 기호테이블에서 중복상태(redundant state)를 제거하는데 많은 양의 비교와 반복절차를 수행하여야만 하는 단점을 갖고 있다.[3][7]

*漢陽大學校 電子工學科
Dept. of Elec. Eng., Hanyang Univ.
論文番號 : 92-42(接受1992. 3. 12)

따라서, 본 논문에서는 상태합성기의 구현을 위하여, 상태그래프로 부터 상태 CHDL(C-based Hard ware Description Language)을 생성할 수 있는 그래픽 에디터를 개발하였다.

또한, 본 논문에서는 상태테이블의 중복상태를 제거하기 위해, 방향성 트리(directed tree)를 기저로 이에 몇가지 휴리스틱 룰을 적용하여, 기존의 알고리듬에 비해 구성이 간단하면서도 빠른시간내에 최소해를 구할 수 있는 상태축소 알고리듬을 제안하였다.

Ⅱ. PLD를 이용한 FSM 회로설계

PAL(Programmable Array Logic)은 AND-OR 2단의 규칙적인 구조를 갖고 있기 때문에, 설계자동화가 용이하고 논리최소화 툴을 이용한 최적설계가 가능하며, 특히, FSM회로 설계 시, 설계자가 요구하는 설계사양(specification)에 매우 적합한 회로를 설계할 수 있다는 장점을 갖고 있다.

FSM 순차회로를 PAL Device로 구현하는 전체 흐름도는 그림 1과 같다.

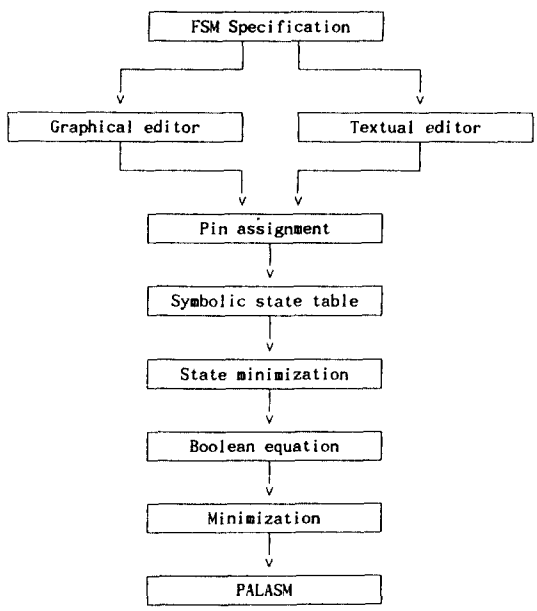


그림 1. 전체 흐름도
Fig. 1. flowchart

1. 그래픽 에디터(Graphic editor)

FSM 회로를 설계하기 위해서는 동작특성을 상태 CHDL로 기술하여야 만 한다. 이를 위하여, 본 연구에서는 상태그래프로부터 상태 CHDL를 자동으로 기술할 수 있는 그래픽 에디터를 개발하였다. 개발된 그래픽 에디터는 그림 2와 같이, 아이콘 메뉴방식으로 동작되도록 설계하였다.

그림 2의 그래픽 에디터에서 상태비트, 조건, 출력변수 명은 우측에 정의되며, 이 중, 상태와 출력은 이진(binary)형태로, 조건은 논리식의 형태로써 상태그래프를 입력된다.

완성된 상태그래프는 상태CHDL 기술형태인 IF, THEN, ELSE, SWITCH 구문으로 자동 변환되어 화일에 저장된다. 상태그래프를 상태CHDL로 변환하는 알고리듬은 그림 3과 같다.

일반적으로 기존의 HDL은 조합논리회로의 기술에는 적합하나, FSM 기술의 경우는 그 기술방법이 까다롭고 복잡하기 때문에, 본 논문에서는 FSM 회로와 조합논리회로를 쉽게 기술할 수 있는 상태 CHDL을 개발하였다.[11]

개발된 상태CHDL의 문장구조는 IF, THEN, ELSE, SWITCH, CASE문으로 구성되어, 다입력조건과 다출력조건을 모두 기술할 수 있어, 규모가 큰 FSM 회로의 설계에 보다 유리하다는 장점을 갖고 있다.

그림 4는 그림 2의 상태그래프를 그림 3의 변환 알고리듬에 의해 변환시킨 상태CHDL 기술문이다.

2. FSM 합성기의 상태최소화 알고리듬

FSM 합성과정에서는 상태그래프에서 추출된 상태테이블로 부터 중복상태(redundant state)를 제거하는 상태축소(state reduction) 절차가 필요하다.

이러한 상태축소 절차는 상태할당에 요구되는 비트의 수와 순차회로의 상태천이함수를 감소시키기 위한 것이다.

불완전하게 기술된 순차시스템(incompletely specified sequential machine)에서의 상태축소 방법은 Paull과 Unger에 의해 일반적인 이론이 정립되었다.[13]

이후, 프로그램이 가능한 체계적인 방법(systematic method)이 연구되어 왔으나, 이들의 대부분은 최소의 닫혀진 커버링(minimal closed covering)를

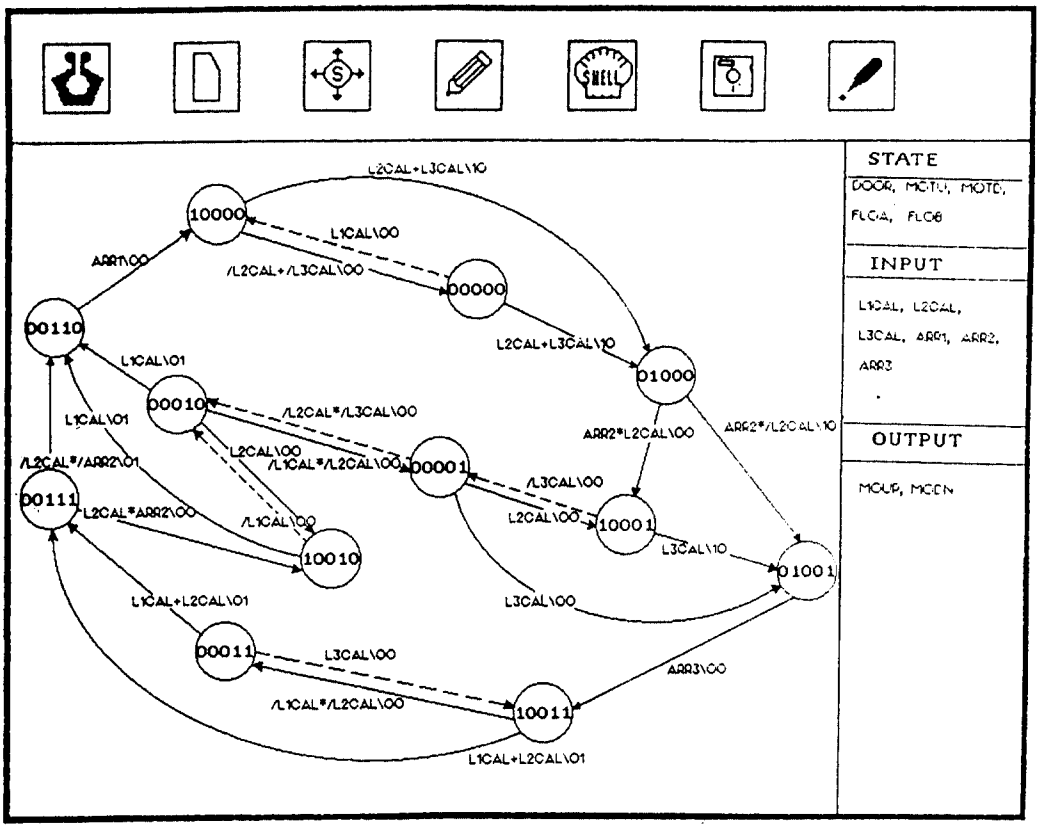


그림 2. 그래픽 에디터
Fig. 2. Graphic editor

```
if ( exist EGA bord ) {   /*  EGA 모니터인가를 확인한다. */
    load_font(state, curror) ;
                          /*  필요한 그래픽도형을 메모리로 불러온다 */
    variable_define ;
                  /* 상태비트변수, 입력변수, 출력변수를 정의한다. */
    making_state_diagram_with_reversed_key ;
                /* 예약키를 사용하여 상태할당, 천이조건, 출력비트값 등을
                                  입력함으로써 상태그래프를 작성한다. */
    state = 1 ;
    write_file (basic_infomation) ;
          /* 상태비트변수 및 입.출력변수 등을 CHDL로 기술한다. */
    while  ( state = STATE_MAX ) {
                        /*  각 상태에 대한 내용을 CHDL로 변환  */
      write ( PSTATE = state) ;
      write ( EQUATION = state_boolean_equation )
```

```
/* 현재상태를 상태비트변수와 비교하여 boolean 방정식으로 변환한다. */
      if ( branch_of_state > 2 or
        variable_of_branch_conditon1 != variable_of_branch_conditon2)
   /* 천이조건을 고려하여 IF문과 SWITCH문으로 구분한다. */
           write ( SWITCH = variable_of_branch_condition ) ;
                        /* 다입력 조건일 경우 SWITCH문으로 기술한다. */
           for ( i = 1 ; i > number_of_branch ; i++)  {
              write ( CASE = valuve_of_branch_conditon ) ;
              write ( NSTATE = next_state_of_state_in_condition ) ;
              write ( OUTPUT = output_variable_of_state ) ;
      }
      else {                                        /* IF문으로 기술한다. */
           write ( IF = variable_and_valuve_of_branch_conditon ) ;
           write ( NSTATE = next_state_of_state_in_condition ) ;
           write ( OUTPUT = output_variable_of_state ) ;
           if ( branch_of_state != 1 )
           write ( ELSE ) ;                    /* ELSE문으로 기술한다. */
           write ( NSTATE = next_state_of_state_in_condition ) ;
           write ( OUTPUT = output_variable_of_state ) ;
      }
    state++ ;
    }
    write ( END )
}
```

그림 3. 상태 CHDL 변환 알고리듬
Fig.2. Conversion algorithm of state CHDL.

```
 MODEL (THREE_ELEVATOR)

ACTIVE high ;
EQUATION
  #  OPEN1  =  DOOR*/MOTU*/MOTD*/FLDA*/FLDB;
  # CLOSE1  = /DOOR*/MOTU*/MOTD*/FLDA*/FLDB;
  #    UP1  = /DOOR* MOTU*/MOTD*/FLDA*/FLDB;
  # OPEN2U  =  DOOR*/MOTU*/MOTD*/FLDA* FLDB;
  # OPEN2D  =  DOOR*/MOTU*/MOTD* FLDA*/FLDB;
  # CLOSE2U = /DOOR*/MOTU*/MOTD*/FLDA* FLDB;
  # CLOSE2D = /DOOR*/MOTU*/MOTD* FLDA*/FLDB;
  #    UP2  = /DOOR* MOTU*/MOTD*/FLDA* FLDB;
  #    DN2  = /DOOR*/MOTU* MOTD* FLDA*/FLDB;
  #    OPEN3 =  DOOR*/MOTU*/MOTD* FLDA* FLDB;
  #   CLOSE3 = /DOOR*/MOTU*/MOTD* FLDA* FLDB;
  #    DN3  = /DOOR*/MOTU* MOTD* FLDA* FLDB;
END

CONTROL

PSTATE = OPEN1 ;                          | PSTATE = OPEN2U ;
 IF (L2CAL+L3CAL) THEN NSTATE = UP1 ;     |  IF (L3CAL) THEN NSTATE = UP2;
```

```
    OUT = MOUP, /MODN ;
  ELSE NSTATE=CLOSE1 ;
    OUT = /MOUP, /MODN ;
 PSTATE = CLOSE1 ;
  IF (L2CAL+L3CAL) THEN NSTATE=UP1 ;
    OUT = MOUP, /MODN ;
 PSTATE = CLOSE1 ;
  IF (L1CAL)THEN NSTATE=OPEN1 ;
    OUT = /MOUP, /MODN ;
 PSTATE = UP1 ;
  IF (ARR2 */L2CAL) THEN NSTATE=UP2 ;
    OUT = MOUP, /MODN ;
   ELSE NSTATE = OPEN2U ;
    OUT = /MOUP, /MODN ;
 PSTATE = CLOSE2D ;
 SWITCH (L1CAL,L2CAL) {
  CASE 1,- :  NSTATE = DN2 ;
    OUT = /MOUP, MODN ;
  CASE -,1 :  NSTATE = OPEN2D ;
    OUT = /MOUP, /MODN ;
  CASE 0,0 :  NSTATE = CLOSE2D ;
    OUT = /MOUP, /MODN ;
   }
 PSTATE = DN2 ;
  IF (ARR1) THEN NSTATE = OPEN1 ;
    OUT = /MOUP, /MODN ;
 PSTATE = UP2 ;
  IF (ARR3) THEN NSTATE=OPEN3 ;
    OUT = /MOUP, /MODN ;
 PSTATE = OPEN2D ;
  IF (L1CAL) THEN NSTATE=DN2 ;
    OUT = /MOUP, MODN ;
   ELSE NSTATE = CLOSE2D ;
    OUT = /MOUP, /MODN
    OUT = MOUP, /MODN ;
  ELSE NSTATE = CLOSE2U ;
    OUT = /MOUP, /MODN ;
 PSTATE = CLOSE2U ;
 SWITCH (L2CAL,L3CAL) {
  CASE 0,0 : NSTATE = CLOSE2D ;
    OUT = /MOUP, /MODN ;
  CASE 1,- : NSTATE = OPEN2U ;
    OUT = /MOUP, /MODN ;
  CASE -,1 : NSTATE = UP2 ;
    OUT = /MOUP, /MODN ;
 PSTATE = OPEN3 ;
  IF (L1CAL+L2CAL) THEN NSTATE = DN3 ;
    OUT = /MOUP, MODN ;
  ELSE NSTATE = CLOSE3 ;
    OUT = /MOUP, /MODN ;
 PSTATE = CLOSE3 ;
  IF (L1CAL+L2CAL) THEN NSTATE = DN3 ;
    OUT = /MOUP, MODN ;
 PSTATE = CLOSE3 ;
  IF (L3CAL) THEN NSTATE = OPEN3 ;
    OUT = /MOUP, /MODN ;
 PSTATE = DN3 ;
 SWITCH (ARR2, L2CAL) {
  CASE 0,0 : NSTATE = DN2 ;
    OUT = /MOUP, MODN ;
  CASE 1,1 : NSTATE = OPEN2D ;
    OUT = /MOUP, /MODN ;
  }
 END
```

그림 4. 상태 CHDL 기술
Fig. 4. State CHDL description

구하는 데 있어 많은 양의 비교와 반복이 요구되므로, 매우 비효율적이다.

따라서, 본 논문에서는 최소의 상태수를 찾는 부분에 있어, 트리의 생성을 하나로 압축하기 위하여 커버링테이블(covering table)과 휴리스틱 룰을 사용하였으며, 이를 바탕으로 Luccio와 Graselli[5]의 "prime class"와 Meise[6]의 "방향성 트리(directed tree)"의 개념을 조합시켜, Miesel 알고리듬에 비해

구성이 간단하면서도 빠른시간 내에 최소해를 구할 수 있는 알고리듬을 제안하였다.

2-1. 최소의 닫혀진 커버링을 구하기 위한 상태축소 알고리듬

FSM의 상태테이블로 부터 중복상태(redundant state)를 제거하는 상태축소 절차에 의해, F/F의 수와 순차회로의 상태천이함수가 감소된다. 표 1의 상태표에 대해 일반적인 상태최소화 알고리듬[11]을 적용시켜, 상태수를 최소화하는 절차는 아래와 같다.

표 1. 상태표
Table 1. State table

Present State	Next state, Output			
	00	01	11	10
A	A, 0	-, -	E, -	B, 1
B	E, -	C, 1	B, -	D, 0
C	-, -	B, 0	-, 1	D, 0
D	A, 0	-, -	F, 1	B, -
E	B, 0	-, -	B, 0	-, -
F	-, -	C, 1	-, 0	G, 1
G	D, 1	D, -	-, -	-, 0

단계 1) : 행 매칭(row matching) 절차에 의해 호환쌍을 추출한다.

단계 2) : 최대호환체(maximal compatible)를 구한다.

{(CD) (CG) (BEF) (ABD) (ABE)}

단계 3) : prime class를 추출한다.

표 1의 상태표로 부터 구해진 prime class는 표 2와 같다.

표 2. prime class의 집합
Table 2. Final list of prime classes

	Prime Class	Class set
C1	ABD	AE, BEF
C2	ABE	0
C3	BEF	0
C4	CD	BD
C5	CG	BD
C6	AD	EF
C7	BD	AE, BF
C8	D	0
C9	C	0
C10	G	0

단계 4) : 최소의 닫혀진 커버링(Minimal closed covering)을 구한다.

기존의 상태축소 알고리듬은 트리의 생성요소로 가장 적은 수의 상태수를 포함하는 prime class의 최소수를 트리의 시작점으로 삼았다.[5] 이 경우, 선택된 prime class의 최소수가 복수개일 때에는 각각의 생성요소 마다 트리를 구성하여야 하며, 최소해를 찾기 위해서는 여러개의 패스를 조사하여야 하므로, 비교적 규모가 작은 상태기에 적용시키기 조차도 적합치 못한 단점이 있다.

따라서, 본 논문에서는 이러한 단점을 개선하고자, 트리의 생성을 하나로 압축시킴으로서, 기존의 방법에 비해 보다 적은수의 패스를 조사하면서도, 가능한 한 많은 수의 최소해를 구할 수 있는 상태축소 알고리듬을 제안하였다.

2-2. 제안한 상태축소 알고리듬

본 논문에서 제안한 상태축소 알고리듬의 단계별 절차는 다음과 같다.

단계 1) 표3와 같은 커버링테이블(covering table)을 작성한다.

표 3. 커버링 테이블
Table 3. Covering table

	Prime Class	A	B	C	D	E	F	G
C1	ABD	×	×		×			
C2	ABE	×	×			×		
C3	BEF		×			×	×	
C4	CD			×	×			
C5	CG			×				×
C6	AD	×			×			
C7	BD		×		×			
C8	D				×			
C9	C			×				
C10	G							×
		3	4	3	5	2	1	3

단계 2) 트리의 생성요소를 결정하기 위한 선택조건을 커버링테이블과 prime class의 집합에 적용시킨다.

(1) 트리의 생성요소를 결정하기 위한 선택 조건

조건1 : 필수 prime class(Essential Prime Class)를 선택한다.
조건2 : 가장 큰 상태수를 커버하는 prime class를 선택한다.
조건3 : 함축된 호환체의 수가 최소인 prime class를 선택한다.

선택조건에 대한 적용순서는 다음과 같다. 먼저, 조건1을 커버링테이블에 적용시켜 트리의 생성요소를 결정한다. 만일 조건1을 통해 트리의 생성요소가 결정되지 않을 경우는 조건2를 커버링테이블에 적용시킴으로서, 트리의 생성요소를 결정한다.

또한, 조건1과 조건2에 의해서도 트리의 생성요소가 결정되지 않을 경우는, 조건3을 prime class 테이블에 적용시켜, 가장 큰 상태수의 호환체들 중, 함축된 호환체의 갯수가 최소인 prime class를 선택하여, 트리의 시작점으로 삼는다.

이러한 절차를 표2에 적용시키면, 조건1에 의해 표3에서(BEF)가 선택되며, 선택된(BEF)는 필수 prime class가 되어, 최소의 닫혀진 커버링 집합 속에 당연히 포함되야 하는 prime class를 의미한다. 따라서, 이러한 필수 prime class(BEF)가 제일 먼저 트리의 생성요소로 결정되게 된다.

(2) 트리의 생성

트리의 생성은 Meisel의 방법에 따라 그림 5와 같이 생성된다. 트리의 노드들은 prime class들로 구성되며, 왼쪽에서 오른쪽으로 생성되는 트리구조 상의 생성요소의 닫힘조건에 따라, prime class들을 확장시킨다.

만약, 부분적으로 구성된 트리의 패스상에서, 닫힘조건에 만족되지 않는 경우는 더 이상 닫힘조건을 따른 확장이 이루어질 수 없으며, 이때는 커버링 조건을 만족시키기 위해 현재 커버링되지 않은 상태를 포함하는 prime class의 최소수를 커버링테이블로 부터 추출하여, 트리의 다음 요소로 지정하게 된다(만일, 상태에 대한 prime class의 최소수가 같을 경우, 임의로 하나의 상태를 선택한다.)

예를 들어, 그림5의 생성요소(BEF)는 함축쌍이 존재하지 않기 때문에, a,c,d,g를 포함한 prime class의 최소수를 생성요소(BEF) 다음으로 지정하게 된다.

커버링테이블 상에서 볼 때, 상태g는 2개의 prime class에 나타나고 있다. 또한, 상태a와 상태c는 3개의 prime class에 나타나며, 상태d는 5개의 prime class에 존재하고 있다.

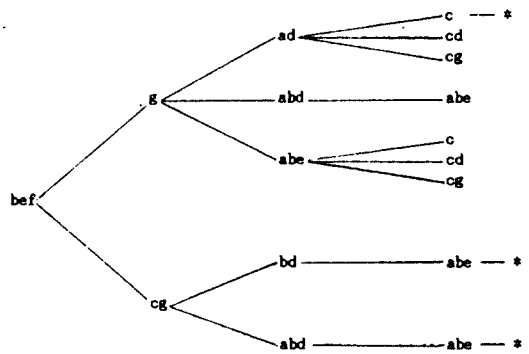


그림 5. 방향성 트리
Fig. 5. Directed tree

상태g를 포함한 prime class 즉, (G)와 (CG)를 생성요소(BEF) 다음에 확장시키면 다음과 같은 2개의 서브체인(BEF, G)와 (BEF, CG)가 구성된다. 이중, 서브체인(BEF, CG)의 경우는, 서브체인(BEF, CG)의 CG에 함축쌍 BD가 존재한다.

또한 BD를 포함하는 prime class (ABD)와 (BD)는 CG의 다음요소로 지정되므로, 서브체인(BEF, CG)는 2개의 서브체인(BEF, CG, BD), (BEF, CG, ABD)로 확장된다.

따라서, 구성된 트리내에서 닫혀진 커버링 조건을 만족하면서, 길이가 가장 짧은 패스로 구성된 체인이 최소의 닫혀진 커버링이 된다.

그림5로부터의 최소해는 아래와 같으며, 줄여진 상태표는 표4와 같다.

1) (BEF, G, AD, C)
2) (BEF, CG, AD, ABE)

표 4. 축소화된 상태표
Table 4. Reduced state table

Present State	Next state, Output			
	00	01	11	10
A	A, 0	D, 1	A, 0	D, 1
B	A, 0	D, 1	A, 1	A, 1
C	D, 0	D, 1	A, 0	A, 0
D	B, 1	B, 0	−, 1	B, 0

3) (BEF, CG, ABD, ABE)

Ⅲ. 실험 및 검증

표5는 Meisel과 Biswas의 알고리듬을 본 알고리듬과 비교한 결과표이다. 성능비교는 최소의 닫혀진 커버링을 추출하는 단계에서 생성된 트리의 노드수로서 비교하였다.

표5에서 FSM1은 문헌[2]에서 제시한 상태표를 Meisel의 상태축소 알고리듬으로 최소화한 결과로서, 20개의 node와 3개의 상태수로 최소화된 반면, 본 논문에서 제안한 상태 축소알고리듬을 적용시킬 경우, 6개의 node와 3개의 상태수를 얻을 수 있었다.

본 논문에서 제안한 상태축소 알고리듬의 복잡도는 $O(n\log_2 n/2)$ 이다.

표 5. 실험 결과

Table 5. Experimental results

FSM	parameters				Meisel [6]		Proposed algorithm		
	ns	ni	#MC	#PC	search node	ns_1	search node	ns_2	t_2
FSM1	6	2	4	14	20	3	6	3	2.4
FSM2	9	4	6	33	N/A	4	12	4	2.8
FSM3	7	4	5	10	36	4	36	4	2.7
FSM4	8	7	5	12	46	4	33	4	2.7

Ⅳ. 결 론

본 논문에서는 FSM 상태합성기를 설계하기 위해, FSM 회로의 상태그래프를 자동으로 상태 CHDL로 변환시켜 주는 그래픽 에디터를 개발하였다.

또한 FSM 설계과정에서 추출된 상태테이블로 부터, 중복상태를 빠른시간 내에 제거하는 상태축소 알고리듬을 제안하였다.

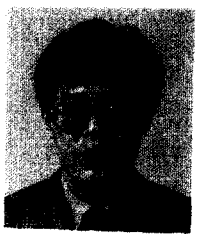
본 논문에서 제안한 상태축소 알고리즘의 성능을 비교하기 위하여, 4개의 benchmark 모델에 대한 실험을 행하였다. 그 결과, 기존 알고리즘에 비해 탐색 노드의 수가 감소됨을 알 수 있었다.

제안된 상태축소 알고리듬과 그래픽 에디터는 PC(MS-DOS) 상에서 C언어로 구현하였다.

참 고 문 헌

1. K.A.Bartlett, D.G.Bostick, G.D.Hachtel, R.M. Jacoby, M.R.Lightner, P.H.Moceyunas, C.R. Morrison and D.Ravens-croft, "BOLD : A multi-level logic optimization system," ICCAD 87, 1987.
2. Z.Kohavi, "Switching and Finite Automata Theory," New York : McGraw-Hill, 1970.
3. M.C Paull and S.H.Unger, "Minimizing the number of state in incompletely specified sequential switching functions," IEEE Trans. Electron. comput. vol EC-8, pp.356-366, sept. 1959.
4. H.S.Kim, K.M.Lee, S.Z.Byun, "The State CHDL descriptions and symbolic cover minimization algorithms for state machine synthesizer using BOLD logic synthesis tool," ICVC '89, pp.81-84, 1989.
5. A.Grasselli and F.Luccio, "A method for minimizing the number of internal states in incompletely specified sequential networks," IEEE Trans. Electron. comput, vol.EC-14, pp. 350-359, june 1965.
6. W.S.Meisel, "A note on internal state minimization in incompletely specified sequential networks," IEEE Trans. Electron. Comput. (Short Note), vol.C-18, pp.508-509, Aug. 1967.
7. N.N.Biswas, "State minimization of incompletely specified sequential machines," IEEE

Trans. Comput, vol. C-19, pp.80-84, Jan. 1974.
8. 류 정호, 김 희석, "PAL을 이용한 상태 CHDL 컴파일러 개발에 관한 연구," 청주대학교 논문집, 1992.
9. 류 정호, 변 상준, 박 광현, 김 희석, "PAL SYNTHESIS를 위한 상태 CHDL(C-Based Hardware Description Language) 기술에 관한 연구," CAD, 전자계산반도체 재료 및 부품 합동 학술 발표회 논문집, pp.34-37, 1991.
10. 조 석, 이 근만, 김 희석, "FSM 합성기에서의 상태 최소화 알고리듬 개발," 씨에이디, 전자계산 반도체 재료 및 부품 합동 학술 발표회 논문집, pp.78-81, 1991.
11. R.G.Bennetts, "An improved method of prime C-class derivation in the state reduction of sequential network," IEEE Trans. on Computers.



李 近 萬(Keun Man YI) 正會員

1949年 9月 18日生

1973년 2월 : 한양대학교 전자공학과 졸업

1980년 : 한양대학교 대학원 전자공학과 졸업 공학석사학위 취득

1992년 4월～현재 : 한양대학교 대학원 전자공학과 박사과정 재학중

1992년～현재 : 청주대학교 전자공학과 전자공학과 부교수

※주관심분야는 VLSI 설계 등임.

林 寅 七(In Chil LIM) 正會員

1962년 : 한양대학교 공과대학 졸업

1970년 : 와세다대학 전자공학과 박사학위 취득, 일본 후지쯔(주) 정보처리시스템연구소 연구원, 한국과학기술원 대우 교수 University of Illinois at Urbana-Champaign의 Computer Science학과 Visiting Professor역임.

1964년 : 한양대학교 강사. 조교수, 부교수.

1977년～현재 : 한양대학교 교수 재직, 현재 동대학교 전자계산소 소장.
IEEE Computer Society Korea Chapter Chairman.

※주연구분야는 Computer Architecture, RISC Processor 및 Compiler설계, VLSI Testing / Testable Design, Simulation, Layout, AI기법을 이용한 VLSI 설계 및 Machine-Translation등임.