論文 92-29A-4-12

상태합성기(State Machine Synthesizer) 설계를 위한 상태 CHDL 개발 및 Two-level minimizer 개발에 관한 연구

(The Developments of State CHDL and Two-Level Minimizer for State Machine Synthesizer)

金 熙 碩* 李 近 萬* 林 寅 七**

(Hi Seck Kim, Keun Man Yi, and In Chil Lim)

要 約

상태합성기는 FSM 합성에 매우 중요하게 사용된다. 본 논문에서는 상태합성기 설계를 위하여 문장구조가 IF, THEN, ELSE, SWITCH, CASE 문으로 구성된 상태 CHDL을 개발하였다. 또한 FSM 합성에 필요한 상태축소 알고리듬, Two-level minimizer 알고리듬과 입출력에 필요한 Pin map window Graphic editor를 개발하였다.

Abstract

The state machine synthesizer is widely used to FSM synthesis. In this paper, we developed the state machine description language "State CHDL" such as IF, THEN, ELSE, SWITCH,CASE statements. Also, an algorithm for efficient state minimization and two level minimizer of FSM and graphical user interface - pin map window, supporting the designer with input-output effency, are presented.

I. 서 론

최근 LSI/VLSI의 복잡도가 증가함에 따라 설계시간의 단축 및 설계의 정확성을 위하여 논리합성 tool을 이용한 설계자동화에 대한 연구가 활발히 진행되고 있다. Digital system을 자동설계하기 위해서는 시스템의 동작특성을 기술하는 기술 언어인 HDL (hardware description language)로 기술한 다음 제어용 부분을 기술하는 언어와 논리조합용 언어로 분리하는데 제어용 부분의 기술언어는 상태 그래프를 기술할 수 있는 FSM(finite state machine) 형태로 기술되어야 한다.

FSM(finite state machine)의 설계는 FSM용 HDL로 부터 기술된 순차회로를 구문분석(parsing)하여 기호 상태표(Symbolic state table)을 추출하고 추출된 상태표를 이용하여 상태 최소화, 상태할당을 수행한다. 그런 다음 논리최소화 과정을 거쳐 최적

*正會員, 淸州大學校 電子工學科
(Dept. of Elec. Eng., Cheongju Univ.)
**正會員, 漢陽大學校 電子工學科
(Dept. of Elec. Eng., Hanyang Univ.)
接受日字: 1991年 8月 12日
(※ 이 논문은 1990년도 교육부 학술연구조성비에 의하여 연구되었음.)

화된 FSM 회로를 device로 구현하는 FSM합성 tool 즉 상태합성기(State Machine Synthesizer)를 필요로 한다.[1]

그러나 대부분의 FSM 합성 시스템은 설계하고자 하는 시스템의 동작특성이 복잡할 경우 HDL로 변환이 어렵고 PC상에서 효과적인 논리최소화가 불가능한 단점을 가지고 있다.

따라서, 본 논문에서는 상태 합성기(FSM) 시스템을 설계하기 위하여 FSM 동작 특성이 다입력, 다출력인 경우에도 기술이 가능한 boolean equation language인 상태 CHDL(c-based hardware description language)과 상태 테이블에서 추출한 boolean equation을 최소화 하는 ESSPRESO-Ⅱ 알고리듬을 기저로한 2단 논리 최소화 알고리듬 C-PLAMIN를 개발하였다. 또한 상태수가 제한적일 때 상태그래프를 자동으로 CHDL로 변환해주는 Graphic Editor와 CHDL에 의해 생성된 netlist에 적합한 PLD Device을 pin 할당할 수 있는 디바이스 pin map editor를 개발하였다. Two-level minimizer(C-PLAMIN)에 의해 최소화한 boolean equation은 PLA 설계 tool인 PALASM의 입력화일 형태로 변환하여 PLD소자로 구현하는 PAL synthesis tool의 설계환경을 구축하였다.[9][10]

Ⅱ. PLD를 이용한 FSM 회로 설계

PAL(programmable array logic)는 AND-OR 2단의 규칙적인 구조를 가지고 있기 때문에 설계 자동화가 용이하며 논리 최소화 tool을 이용한 최적 설계가 가능하고 특히 PLA을 이용한 FSM 회로는 설계자가 요구하는 설계사양(specification)에 매우 적합하게 설계를 할 수가 있다. 이러한 FSM 회로는 조합회로 부분과 제어회로 부분으로 나누며 Primary 입력과 현재상태에 의해 출력과 다음 상태가 결정되는데 그림1에 FSM 회로를 나타냈다.

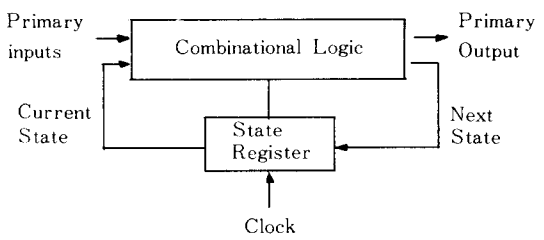

그림 1. FSM 회로
Fig. 1. FSM circuit.

FSM 순차회로는 상태그래프(state graph)로 전개할 수 있으므로 상태그래프를 상태 CHDL로 기술한 뒤 PLA Device로 구현하는 전체 흐름도는 그림 2와 같다.

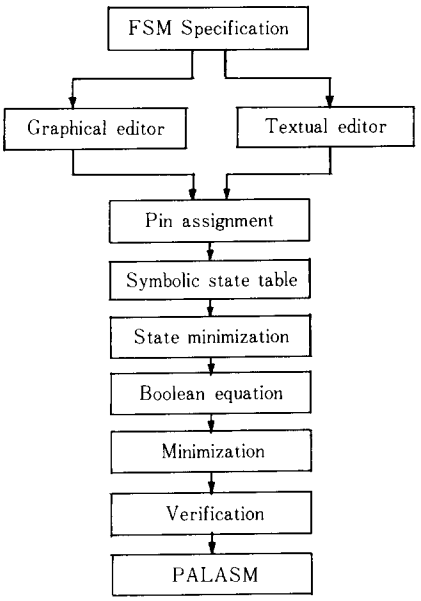

그림 2. 전체 흐름도.
Fig. 2. Total flowchart.

1. 상태 CHDL 기술(State CHDL description)

PAL을 이용하여 FSM회로를 설계하기 위해서는 동작특성을 상태 그래프에서 상태 CHDL로 기술해야 하는데 설계하고자 하는 FSM 회로의 상태수가 제한적일 경우(약 20개 이하) 본 연구실에서 개발한 Graphic editor을 이용하여 직접 상태그래프로부터 상태 CHDL를 자동으로 기술하는 방법과 상태 그래프로부터 수작업을 통하여 상태 CHDL로 기술하는 방법으로 나눌수가 있다. Graphic edtior을 이용하여 상태 CHDL를 기술하는 방법은 상태 그래프로 직접 입력하여 상태 CHDL의 문법에 맞추어 자동으로 상태 CHDL를 기술할 수 있다. Graphic editor는 EGA graphic card에서 개발 하였고, icon 메뉴방식으로 그림3에 Graphic editor의 화면과 엘리베이터 제어회로의 상태도를 예를들어 나타내었다.

그림3에서 Graphic와 editor의 입력기술은 우측의 상태비트, 조건, 출력 변수명을 정의한 후 각각의 상태와 출력은 이진(binary)으로 조건은 논리식의 형

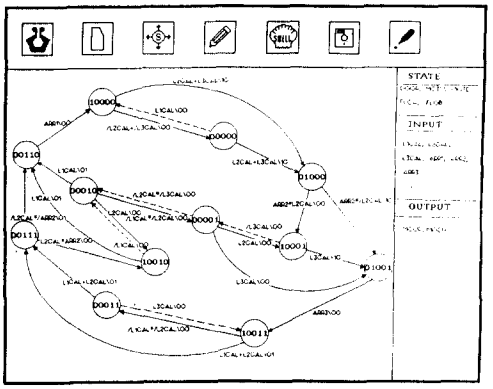

그림 3. 그래픽 에디터
Fig. 3. Graphic editor.

태로써 상태그래프를 입력한다. 완성된 상태그래프를 상태 CHDL 기술형태인 IF, THEN, ELSE, SWITCH 구문으로 자동 변환되어 화일에 저장 되는데, 상태그래프를 상태 CHDL로 변환하는 알고리듬은 그림4와 같다.

```
if ( exist EGA bord ) {   /* EGA 모니터인가를 확인한다. */
    load font(state, curror) :
                          /* 필요한 그래픽 도형을 메모리로 불러온다 */
    variable_define :
                  /* 상태 비트 변수, 입력 변수, 출력 변수를 정의한다. */
    making_state diagram_with_reversed key :
                  /* 예약키워 사용하여 상태할당, 천이조건, 출력비트값등을
                                        입력함으로써 상태그래프를 작성한다. */
    state = 1 :
    write_file (basic_infomation) :
            /* 상태 비트 변수 및 입 . 출력 변수 등을 CHDL에 기술한다. */
    while  ( state = STATE_MAX ) {
                     /*  각 상태에 대한 내용을 CHDL로 변환  */
       write ( PSTATE = state) :
       write ( EQUATION = state boolean equation )
/* 현재 상태를 상태 비트 변수와 비교하여 boolean 방정식으로 변환한다. */
       if ( branch_of_state > 2 or
         variable_of branch_conditon1 != variable of_branch_conditon2) {

/* 천이조건을 고려하여 IF문인가 SWITCH문으로 할 것인가를 결정한다. */
        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 4. 상태 CHDL 변환 알고리듬
Fig. 4. Translation algorithm of state CHDL.

일반적으로 기존 HDL은 조합논리회로의 기술은 용이하나 FSM 형태의 기술이 까다롭고 복잡하기 때문에 본 논문에서는 FSM 회로와 조합논리회로를 쉽게 기술할 수 있는 PAL 설계용 상태 CHDL을 개발하였다[11] 개발된 PAL 설계용 상태 CHDL의 문장구조는 IF, THEN, ELSE, SWITCH, CASE문으로 구성되어 다 입력 조건과 다 출력인 경우까지 기술할 수 있어 복잡한 FSM 회로의 설계가 가능하다.

예를들어 Graphic editor에 입력된 3층 엘리베이터의 상태그래프를 자동으로 상태 CHDL로 변환하면 그림5와 같다.

```
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;
```

```
CONTROL
PSTATE = OPEN1 :
 IF (L2CAL*L3CAL) THEN NSTATE=UP1:
    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 :
 ELSE NSTATE= CLOSE1:
    OUT = /MOUP, /MODN :
PSTATE = UP1:
 IF (ARR2 *L2CAL) THEN NSTATE=UP2:
    OUT = MOUP, /MODN :
  ELSE NSTATE = CLOSE2D:
    OUT = /MOUP, /MODN :
PSTATE = CLOSE2D:
SWITCH (L1CAL,L2CAL){
 CASE 1, - :   NSTATE = DN2:
    OUT = /MOUP, MODN :
 CASE  1 :   NSTATE = OPEN2D:
    OUT = /MOUP, /MODN :
 CASE 0,  :   NSTATE = CLOSE2D:
    OUT = /MOUP, /MODN :
 CASE  ,0    NSTATE = CLOSE2D:
    OUT = /MOUP, /MODN :
  }
PSTATE = DN2:
 IF (ARR1) THEN NSTATE = OPEN1:
    OUT = /MOUP, /MODN :
 ELSE NSTATE = CLOSE2U:
    OUT = /MOUP, /MODN :
PSTATE = UP2:
 IF (ARR3) THEN NSTATE=OPEN3:
    OUT = /MOUP, /MODN :
 ELSE NSTATE = UP2:
    OUT = MOUP, /MODN :
PSTATE = OPEN2D:
 IF (L1CAL) THEN NSTATE=DN2:
    OUT = /MOUP, MODN :
PSTATE = UP1:
 IF (ARR2*L2CAL) THEN NSTATE=OPEN2
    OUT = MOUP, /MODN :
 ELSE NSTATE=UP1:
    OUT = MOUP, /MODN :
PSTATE = OPEN2U:
 IF (L3CAL) THEN NSTATE=UP2:
    OUT = MOUP, /MODN :
 ELSE NSTATE= CLOSE2U:
    OUT = /MOUP, /MODN :
PSTATE = CLOSE2U:
 IF (L3CAL) THEN NSTATE=UP2:
    OUT = /MOUP, /MODN :
PSTATE = CLOSE2U:
 IF (L2CAL)THEN NSTATE=OPEN2U:
   OUT = /MOUP, /MODN :
 ELSE NSTATE = DN2:
   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 :
 ELSE NSTATE = CLOSE3:
    OUT = /MOUP, /MODN :
PSTATE = DN3:
  SWITCH  (ARR2, L2CAL) {
    CASE 1, 0 :  NSTATE = DN2:
     OUT = /MOUP, MODN :
    CASE 1, 1 :  NSTATE = OPEN2D:
     OUT = /MOUP, /MODN :
    CASE 0, - :  NSTATE = DN3:
     OUT = /MOUP, MODN :
   }
  END
```

그림 5. 상태 CHDL 기술
Fig. 5. State CHDL description.

2. Device 선정 및 Pin 할당

디지탈 시스템을 설계하기 위해서는 동작특성을 상태 CHDL로 기술한 뒤 설계 사양에 맞는 device을 선정하여 pin을 할당하여야 하는데, 기존의 HDL은 device 선정 및 핀 할당시 device을 선정하기 위해 data book을 참고해야 하고 각 핀의 기능을 일일이 확인해야 하는 어려움이 있다.

본 논문에서는 상태 CHDL을 기술하고 소자 선정시 고려되어야 하는 조건을 입력하여 만족하는 소자들을 알려주며, 각 핀의 기본적인 기능이 출력되어 핀 지정이 용이한 pin map editor를 개발 하였다. 개발한 pin map editor에서 핀 할당은 방향 key를 사용하여 핀을 변경하고, 지정된 key로 핀 할당된 정보를 저장하게 되는데 핀 할당 방법을 그림6에 기술하였고 개발한 pin map editor는 그림7에 나타내었다.

```
open_library_file ;
select_device:          /* 조건을 입력하고, 조건을 만족하는 소자를 선택한다. */
open window ( PIN MAP )
control pin assignment <- device.control.function ;
        /* VCC, GND등과 같은 특수한 핀은 library로 부터 할당을 한다. */
input pin_and_output pin_and in-output_pin_display ;
         /* 입력 핀, 출력 핀, 입출력 겸용핀인가를 window에 표시한다. */
MAX_PIN <- total_pin_size ;
pin <- 1 ;
b <- initial_location_of_horizon ;
c <- initial_location_of_vertical ;
while ( "TAB" or "ESC" key input ) {
                /* TAB이나 ESC 키를 누르면 pin_map 화면이 사라진다.*/
  gotoxy( b, c ) ;
  switch ( key input ) {
                    /* 방향키를 사용하여 핀을 지정한후 할당한다. */
  case UP_arrow_key :
       if (  c != initial_location_of_vertical  )
          ( a <= MAX_PIN ) ?  pin-- :    p++ ;
  case Down_arrow_key :
       if (  c != MAX_PIN/2  )
          ( a <= MAX_PIN ) ?  pin-- :    p++ ;
  case Left_key :
       if (  b != initial_location_of_horiaon  )
          b = initial_location_of_horiaon , pin = MAX_PIN - pin + 1 ;
  case Right_key :
       if (  b != finial_location_of_horiaon  )
          b = finial_location_of_horiaon , pin = MAX_PIN - pin + 1 ;
  case TAB_key :
       close_window ( PIN MAP ) ;
       pin assignment save :    /* 핀 할당된 정보를 화일로 저장한다. */
       break ;
  case ESC_key :

       clear_window ( PIN MAP ) ;
       break ;
  default :
       if ( pin == control_pin ) break ;
   /* 특수한 핀(VCC, GND등)들은 library에 의해 이미 할당된다. */
       if ( pin == input_output_pin )
          select pin with input or output pin ;
  /* 입출력겸용핀은 입력 또는 출력 핀으로의 사용여부를 먼저 결정 */

       pin_assignment : /* 첫글자는 영문자나 "/"로 기술한다음 핀 할당 */
```

그림 6. 핀 할당 알고리듬
Fig. 6. Pin assignment algorithm.

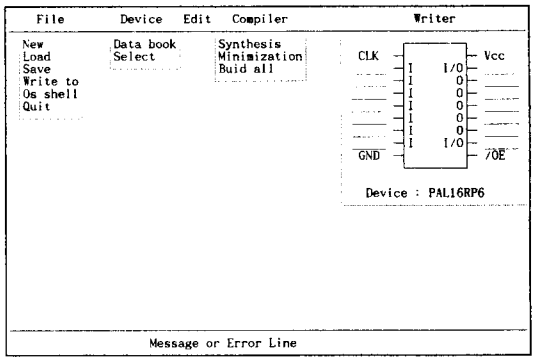

그림 7. 핀 맵 에디터 화면
Fig. 7. Screen layout of a pin map editor.

그림8에 pin map editor를 위한 library의 구조를 나타내었으며, VCC나 GND 등과 같은 기본적인 핀은 library에 의해 할당되므로 user가 할당할 수 없도록 하였다. 할당된 pin 정보는 상태 CHDL의 입력과 출력변수로 저장되고, 동작기술과 연관되어 boolean equation을 추출할때 사용된다.

```
struct device {
   int total ;                  /* 전체 pin의 갯수 */
   int input[MAX_PIN] ;         /* 입력 pin의 갯수및 위치 */
   int output[MAX_PIN];         /* 출력 pin의 갯수및 위치 */
   int inout[MAX_PIN] ;         /* 입출력 겸용 pin의 갯수및 위치 */
   struct {
      int pin ;                 /* 기본적인 기능 pin의 위치 */
      char *function ;          /* 기능 */
   } control[MAX_PIN] ;
}
```

그림 8. Libary의 자료구조
Fig. 8. Data structure of library.

3. FSM 합성기에서의 상태 최소화 알고리듬 개발

FSM synthesis 과정에서 상태 그래프로 부터 추출된 상태 테이블에서 중복된 상태(redundant state)를 제거하는 상태 축소(state reduction) 절차가 존재한다. 이러한 상태 축소는 상태 할당 하는데 있어 비트의 수를 줄일 뿐 아니라 순차 회로의 상태 천이 함수를 감소시키게 된다. Don't care를 고려한 불완전하게 기술된 순차 시스템(incompletely specified sequential machine)에서의 상태 축소 절차는 Paull과 Unger에 의해 일반적인 이론이 정립되었다.

이 후 프로그램이 가능한 체계적인 방법(systematic method)이 연구되어 왔으나, 대부분이 최소의 닫

혀진 커버링(minimal closed covering)를 구하는데 있어서 막대한 양의 비교(comparison)와 반복적인 절차로 구성되어 있다. Meisel[6]은 prime compatibles로 부터 방향성 트리를 구성하여, 그 중 가장 짧은 패스(path)를 최소의 해로 선택하였다. 이 방법은 비교적 작은 상태기에서 조차도 많은 패스를 조사해야 하는 단점을 가지고 있기 때문에 본 논문에서는 최소의 상태수를 찾는 부분에 있어 트리의 생성을 하나로 압축하기 위한 커버링 테이블(covering table)과 몇가지 heuristic rule을 제안하며, 이를 바탕으로 Luccio and Graselli[5]의 "prime class"와 Meisel[6]의 "방향성 트리(directed tree)"의 개념을 조합하여 기존의 알고리듬[6]에 비해 구성이 간단하면서도 빠른 시간내에 최소의 해를 구할 수 있는 알고리듬을 사용하였다.

본 논문에서 사용한 상태 축소 알고리듬의 timing complexity는 $n\log_2 n/2$ 이다.[10]

4. Boolean equation 추출

상태 CHDL로 부터 조합논리는 Boolean equation으로 기술하지만 순서논리는 FSM 형태로 기술되어 있으므로, symbolic state table로 부터 Boolean equation을 추출해낼 수가 있다. 그림5의 상태 CHDL로 부터 구한 Simbolic state table은 표1에 나타내었으며 Simbolic state table로 부터 Boolean equation을 추출하는 알고리듬은 그림9와 같다.

표 1. 상태 테이블의 예
Table 1. Example of state table.

L1CAL	L2CAL	L3CAL	ARR1	ARR2	ARR3	PS	NS	MOUP	MODN
-	1	-	-	-	-	OPEN1	UP1	1	0
-	-	1	-	-	-	OPEN1	UP1	1	0
				⋮					
-	0	-	-	1	-	DN3	DN2	0	1
-	1	-	-	1	-	DN3	OPEN2D	0	0
-	-	-	-	0	-	DN3	DN3	0	1

표1에서 PS, NS의 상태변수들의 상태할당은 매우 중요하게 취급하고 있으나 본 논문에서는 임의로 할당하였으며 PC상에서 상태할당할 수 있는 상태할당 알고리듬을 개발할 예정이다. 상태테이블에서 출력변수가 1인가 0인가를 판단한 뒤 입력변수를 추출하여 논리합을 구하면 각 출력에 대한 boolean equation을 구할수가 있다.

표2는 boolean equation 추출 알고리듬을 이용하여 boolean equation을 추출한 예이다.

```
E = { e1,1, e1,2, ..... , e_{ℓ,m} }   /* 추출할 변수의 집합 */
I = { i1,1, i1,2, ..... , i_{ℓ,n} }   /* primary 입력 집합 */
P = { p1,1, p1,2, ..... , p_{ℓ,a} }   /* 현재상태 집합 */
B = { b1, b2, ..... , b_m }           /* 추출될 boolean equation 집합 */
    ℓ : 상태천이표의 행(raw)의 수        m : 추출할 변수의 수
    n : primary 입력 변수의 수           a : 현재상태 변수의 수

for ( i = 1 ~ m ) {
  Oi = temp = ϕ
  for ( j = 1 ~ ℓ ) {
    if ( [ei,j] == ACTIVE ) {
      Bi = Bi + temp , temp = ϕ
      for ( k = 1 ~ n )
        if ( [ij,k] == 1 )
          temp = temp * ij,k
        else if ( [ij,k] == 0 )
          temp = temp * /ij,k
      for ( k = 1 ~ a )
          temp = temp * pj,k
    }
  }
}
```

그림 9. Boolean equation 추출 알고리듬
Fig. 9. Boolean equation extraction algorithm.

표 2. 부울 방정식을 추출한 예
Table 2. Example of boolean equation extraction.

```
DOOR=L1CAL*/DOOR*/MOTU*/MOTD*/FLDA*/FLDB+L2CAL*ARR2*
     /DOOR*MOTU*/MOTD*/ FLDA*/FLDB
                          :
MODN=L1CAL*DOOR*/MOTU*/MOTD*FLDA*/FLDB+L1CAL*/DOOR*
     /MOTU*/MOTD*FLDA*/FLDB
    +/ARR1*/DOOR*/MOTU*MOTD*FLDA*/FLDB+L1CAL*DOOR*
     /MOTU*/MOTD*FLDA*FLDB
```

추출된 boolean equation은 최적화된 상태가 아니므로 논리최소화를 수행하기 위해 본 논문에서 개발한 논리최소화 tool인 C-PLAMIN의 입력형태인 binary값으로 변환하면 표3과 같이 나타낼 수 있다.

표 3. 부울 방정식의 이진 변환
Table 3. Binary translation of boolean equation.

```
door=1 2 2 2 2 2 0 0 0 0 0     fldb=2 0 2 2 1 2 0 1 0 0 0
     2 1 2 2 1 2 0 1 0 0 0          2 1 2 2 1 2 0 1 0 0 0
                          :
     2 1 2 2 1 2 0 0 1 1 1
     2 2 2 2 0 2 0 0 1 1 1
```

5. 논리 최소화(Logic minimization)

최근의 논리합성 tool로는 ESPRESSO-Ⅱ, ESPRESSO-MV, ESPRESSO-MLT, MIS 등과 같은 큐브를 기저로한 논리합성 tool이 사용되고 있는데, 이러한 논리합성 tool들은 논리함수를 최소화 하기 위해

일반적으로 두가지 방법을 사용하고 있다. 첫번째 방법은 논리함수의 offset을 이용하여 논리최소화를 수행하는 방법이고, 두번째 방법은 Tautology checking 알고리듬을 이용하여 논리최소화를 수행하는 방법이다. 논리함수의 offset을 이용한 논리최소화 방법은 주어진 큐브가 implicant인지 아닌지를 결정하는데 사용되지만 offset의 큐브의 수가 입력변수의 수를 지수함수적으로 증가시킬 수 있어 offset를 생성하는데 시간이 오래 걸린다는 단점도 가지고 있다.

Tautology checking 알고리듬을 이용한 논리최소화 방법은 Redundant cube을 찾는데 시간은 오래 걸리지만 onset 큐브갯수가 작고 offset 큐브갯수가 크면 최소커버(minimal cover)를 구하는 시간이 빠르다는 장점을 가지고 있다.

따라서, 본 논문에서는 추출된 boolean equation의 binary 값을 Tautology checking 알고리듬을 기저(based)로 한 본 논문에서 개발한 논리최소화 tool인 C-PLAMIN을 이용하여 최소화를 수행하였다. 논리최소화 tool C-PLAMIN은 2단 레벨 최소화 알고리듬으로 PC상에서 최소화 과정을 수행하며 수행시간을 줄이기 위해 병합(merge) 알고리듬을 이용; prime 수를 최소로 줄인 상태에서 Quick expand, Reduce, Irredundant_cover을 수행하므로써 최소화 시간을 대폭 감소시켰다.

다음은 C-PLAMIN의 최소화 과정의 단계별 절차이다.

[단계 1] 병합(Merge) 알고리듬을 수행한다.[8]

병합(Merge) 알고리듬은 prime를 구하기 위해 각 큐브(cube)들의 don't care('2'의 갯수) 갯수를 갯수를 계산하여 don't care의 갯수가 가장 많은 큐브부터 차례로 sorting한 다음 각 큐브들간에 거리(distance)가 1인 큐브들을 하나의 큐브로 병합하는 distance_one_merging 알고리듬과 각 큐브들의 포함관계를 조사하여 병합하는 single_cube_containment 알고리듬 그리고 두 큐브의 거리(distance)가 1인 갯수가 maxdistance(전체 리터럴의 1/3에 해당되는 literal) 보다 작거나 같으며 하나의 큐브로 병합하는 supercube 알고리듬을 이용하여 최소화를 수행한다.[8]

[단계 2] QUICK_EXPAND 알고리듬을 수행한다.

Quick_expand는 병합(merge)된 각 큐브들로 부터 많은 literal를 줄이기 위해 사용되는 논리 최소화 알고리듬으로 각 큐브들의 literal를 제거하기 위해 Onset인 큐브 $(c_1, c_2, \cdots c_n)$의 집합과 don't care 큐브 $(d_1, d_2, \cdots, d_m)$의 집합에 있는 큐브들의 거리(distance)를 신속히 비교하기 위해 큐브들의 2의 갯수가 가장 많은 큐브를 첫번째에 큐브가 되도록 Sorting 한 다음 onset인 큐브와 거리(distance)가 1이 되는 don't care 큐브를 찾아서 병합한다.

[단계 3] 축소(Reduce) 알고리듬을 수행한다.

축소(Reduce) 알고리듬은 Quick-expand 알고리듬에 의해 생성된 큐브의 갯수가 가장 많은 큐브를 기준최소항으로 선정한 뒤 기준최소항을 제외한 나머지 큐브들을 축소하게 된다. 축소된 큐브들은 서로 다른 방향으로 확대(expand)하여 최소화가 가능한 새로운 큐브(cube)로 병합된다.

[단계 4] 비 장황 커버(Irredundant_cover) 알고리듬을 행한다.

비장황 커버(Irredundant_cover) 알고리듬은 축소(reduce) 알고리듬에 의해 생성된 큐브들중에 Redundant한 큐브를 제거하기위해 Cofactor 개념과 Tautology 알고리듬을 이용하여 필수주항(Essential prime)을 구하게 된다. 기준최소항은 필수주항(Essential pirme)이지만 나머지 큐브들은 필수주항임을 보장할수가 없기 때문에 각 큐브들의 Cofactor을 구하여 $T \equiv 1$(tautology)이면 Redundant 큐브이므로 제거하고, $T \neq 1$이면 필수주항(Essential prime)이므로 제거할 수가 없다. 따라서, 위의 과정을 반복적으로 수행하므로써 필수주항을 구할 수 있었다.

표4는 본 논문에서 개발한 논리최소화 tool인 C-PLAMIN과 BOLD의 two-level minimizer와 최소화 결과를 비교한 것이고 표5는 binary로 변환된 boolean equation을 C-PLAMIN을 이용하여 최소화한 결과이다.

표3을 본 연구실에서 개발한 C-PLAMIN으로 최소화한 결과 총 490개의 literal을 176개의 literal로 최소화하여 총 literal의 65%가 최소화 되었다.

또한 PALASM의 논리최소화 tool의 최소화 결과인 186개의 literal에 비해 10개의 literal이 감소되었다. 따라서 표5의 최소화 결과를 PALASM의 입력

표 4. C-PLAMIN와 BOLD 및 PALASM2의 최소화 결과

Table 4. Minimization result of C-PLAMIN & BOLD & PALASM2.

	I/O	literal	BOLD	PALASM2	two-level minimizer
FSM 1(traffic controller)	5/7	122	37	···	37
FSM 2(Z Z Z)	7/4	59	59	···	59
FSM 3(ROM)	9/10	183	183	186	166
FSM 4(three elevator)	6/7	490	···	45	176
FSM 5(dummy)	7/6	47	···	45	39

표 5. 논리최소화 결과
Table 5. Logic minimization result.

door=	1 2 2 2 2 2 0 0 0 0 0	flda=	2 2 2 2 2 2 1 0 2 1 1
	2 1 2 2 1 2 0 1 0 0 0		2 2 2 2 2 2 2 0 0 0 1 2
	2 1 2 2 2 2 0 0 0 0 1		1 2 2 2 2 2 2 2 0 0 1 2
	2 1 2 2 2 2 0 0 0 1 0		2 2 2 0 2 2 2 1 2 1 1 0
	2 2 2 2 2 1 0 1 0 0 1		2 0 0 2 2 2 2 1 0 0 2 1
	2 2 2 1 2 2 0 0 1 1 0	fldb=	2 2 2 2 2 2 2 0 0 0 2 1
	2 2 1 2 2 2 0 0 0 1 1		2 2 2 2 2 2 2 2 9 0 1 1
	2 2 2 2 1 2 0 0 1 1 1		2 1 2 2 2 2 2 2 0 0 2 1
motu=	2 1 2 2 2 2 2 0 0 0 0		2 2 1 2 2 2 2 2 0 0 2 1
	2 2 2 2 2 0 1 0 1 0 1		2 2 2 2 1 2 1 1 1 0 0 2
	2 2 1 2 2 2 2 0 0 0 2		2 2 2 2 1 2 1 1 1 0 0 2
	2 0 2 2 2 2 1 0 0 1 0		2 2 2 2 2 2 2 1 0 1 1
	2 2 2 2 0 2 1 1 0 0 0		2 2 2 2 0 2 1 0 2 1 1
motd=	2 2 2 0 2 2 2 1 1 1 1		0 0 2 2 2 2 2 1 0 0 1 2
	2 0 2 2 2 2 2 1 1 1 1	moup=	2 2 2 0 2 2 2 2 2 1 1 2
	2 2 2 0 2 2 1 0 1 1 0		2 2 2 2 0 2 2 2 1 2 1 2
	1 2 2 2 2 2 2 1 0 1 2		2 0 2 2 1 2 2 2 2 1 1 2
	2 1 2 2 2 2 2 0 0 1 2	modn=	2 2 2 0 2 2 2 2 1 2 1 2
			2 2 2 2 0 2 2 2 1 2 1 2
			2 0 2 2 1 2 2 2 1 2 1 2

화일로 기술하기 위해 다시 boolean equation으로 변환하면 그림10과 같다.

그림10의 boolean equation을 PALASM의 입력화일로 변환하여 JEDEC화일을 생성한 뒤 PC상의 RS-232C 시리얼 포트를 통해 PAL WRITER에 전송하여 PAL 디바이스로 구현하였다.

```
DOOR := ARR2*L2CAL*DOOR*/MOTU*MOTD*FLDA*FLDB
      + LICAL*DOOR*/MOTU*/MOTD*/FLDA*/FLDB
      + ARR2*L2CAL*DOOR*/MOTU*/MOTD*/FLDA*/FLDB
                    .
                    .
                    .
MODN = ARR1*MOTD*FLDA
      + ARR2*MOTD*FLDA
      + /L2CAL*ARR2*MOTD*FLDA
```

그림10. Boolean equation의 최소화 예
Fig. 10. Example of boolean equation minimization.

Ⅲ. 결 론

본 논문에서는 PAL을 이용한 FSM 상태 합성기를 설계하기 위해 PAL 설계용 boolean equation 언어인 상태 CHDL과 PAL 설계사양에 맞는 PAL device의 pin map을 지정하고 지정된 출력을 boolean equation으로 추출할 수 있는 pin map editor와 제한된 상태수를 가진 FSM 회로의 상태그래프를 자동으로 상태 CHDL로 변환할 수 있는 Graphic editor를 개발하였다. FSM 설계과정에서 추출된 boolean equation을 최소화 하기 위해 논리최소화 tool 인 two-level minimizer(C-PLAMIN)을 개발하였다. 개발된 상태 CHDL과 C-PLAMIN에 의해 추출된 boolean equation을 PALASM의 입력화일 형태로 변환하여 최종적으로 PAL device로 구현할 수 있는 PAL synthesis tool의 설계환경을 구축하였다.

본 논문에서 개발한 상태 축소 프로그램과 상태 CHDL 그리고 two-level minimizer는 IBM-AT(MS-DOS) 상에서 C언어로 구현하였다.

앞으로 연구과제는 보다 효과적인 pin map editor의 개발 및 PAL Synthesis tool상에서 JEDEC file을 생성하는 algorithm 개발과 상태할당을 PC상에서 할 수 있는 상태할당 알고리듬 개발이 요구된다.

參 考 文 獻

[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-craft, "BOLD: A multi-level logic optimization system," ICCAD 87, 1987.

[2] K. Bartlett, R. Brayton, G. Hachtel, R. Rudell, A. Sangiovanni-Vincentelli, and A. Wang, "Multi-level logic minimization using implicant Don't cares," ICCAD 1986.

[3] H. Savoj, A. Malik, R.K. Rrayton, "Fast-two-level logic minimizes for multi-level logic synthesis," ICCAD 89, pp. 544-547, 1989.

[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] 김 희석, 이 근만, 변 상준, "상태합성기 설계를 위한 PC상에서의 다출력 PLA minimizer개발에 관한 연구," 반도체 · 재료 및 부품 CAD 연구회 합동학술 발표회 논문집, pp. 166-168, 1990.
[9] 류 정호, 변 상준, 박 광현, 김 희석, "PAL SYNTNESIS를 위한 상태 CHDL(C-Based Hardware Description Language)기술에 관한 연구," CAD, 전자계산 반도체 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] PAL DATABOOK

著 者 紹 介

金 熙 碩(正會員)
1954年 12月 23日生. 1977年 2月 한양대학교 전자공학과 졸업. 1980年 2月 한양대학교 대학원 전자공학과 졸업 공학석사 학위 취득. 1985年 8月 한양대학교 대학원 전자공학과 박사학위 취득(CAD 전공). 1987年 8月~1988年 9月 미 콜로라도 대학 VLSI 설계실 객원교수. 현재 청주대학교 전자공학과 부교수. 주관심분야는 PLD 설계 및 Logic Synthesis 등임.

林 寅 七 (正會員) 第28卷 A編 第2號 參照
현재 한양대학교 전자공학과 교수

李 近 萬(正會員)
1949年 9月 18日生. 1973年 2月 한양대학교 전자공학과 졸업. 1980年 한양대학교 대학원 전자공학과 졸업 공학석사학위 취득. 1992年 4月~현재 한양대학교 대학원 전자공학과 박사과정 재학중. 1992年~현재 청주대학교 전자공학과 부교수. 주관심분야는 VLSI 설계 등임.