論文 90-27-12-16

상위 수준 기술로부터 순차 회로의 자동 생성

(FSM Synthesis from High-Level Descriptions)

黃 善 泳* 柳 眞 秀*

(Sun Young Hwang and Jin Soo Yoo)

要 約

Thor functional/behavioral 시뮬레이터의 모델링 언어로 사용되는 상위 수준 하드웨어 기술 언어인 CHDL로부터 순차 회로를 자동 합성하는 시스템을 개발하였다. 본 논문에서는 컴파일 과정, 상태 최소화, 상태 할당 알고리듬에 대하여 기술한다. 제안된 상태 할당 알고리듬은 조건 행렬과 similarity 그래프를 이용하여 최적의 상태 벡터를 생성한다. 표준 테스트 회로인 MCNC 벤치마크에 대하여 실험하여, 제안된 알고리듬이 효율적임을 확인하였다.

Abstract

A synthesis system generating sequential circuits from a high-level hardware description language CHDL, modelling language for Thor functional/behavioral simulator, is developed. In this paper, we describe the semantic analysis process, state minimization and state assignment algorithms. Proposed assignment algorithm generates optimal state vectors using constraint matrix and similarity graph.

Expremental results for MCNC benchmarks, standard test circuits, show that the system implementing the proposed algorithms can be a viable tool for designing large finite state machines.

I. 서 론

최근 VLSI 기술이 급속히 발전하고, 설계하고자 하는 시스템의 규모가 커짐에따라 설계 자동화를 위한 CAD tool의 중요성이 증대되고 있다. 특히, 마이크로 프로세서를 비롯한 많은 디지탈 시스템의 제어 부분에 사용되는 큰 규모의 순차 회로의 경우, 설계자가 직접 설계하기가 어렵게 되어 자동 설계 시스템이 요구되었고, 이에따라 여러 자동설계 시스템이 개발되었다.[5,6] 이들 시스템은 상태표(state table)를 입력으로 받아들여 FSM(finite state machine)을 합성하기 때문에 규모가 작은 FSM 설계의 경우 문제가 없으나, 입력과 상태의 갯수가 많은 복잡한 FSM의 경우 동작의 기술과 검증에 어려움이 있다. 따라서 규모가 큰 FSM에 대해서도 효율적으로 설계하기 위하여 상위 수준 기술 언어(high-level description language) 혹은 하드웨어 기술 언어(hardware description language)의 사용으로 시스템의 기술과 documentation을 용이하게 하고, 이의 검증을 위한 시뮬레이션 지원과 상위 수준 기술로부터 자동적으로 하드웨어를 설계해주는 시스템의 개발이 중요하게 되었다. 하드웨어 기술 언어는 주어진 회로의 동

*正會員, 西江大學校 電子工學科
(Dept. of Elec. Eng., Sogang Univ.)

接受日字 : 1990年 9月 3日

작 혹은 구조 기술을 정확히 기술할 수 있어 표현과 이해가 쉬어야 하고, 구문 규칙이 간단해야 하며 합성력과 이식성을 갖추어야 한다.

한편, 설계 대상 시스템의 규모가 커짐에 따른 설계 복잡도에 대처하기 위하여 구조적이고 계층적인 설계 방법에 의한 규칙적인 기능 모듈의 자동합성은 구현된 회로의 정상 동작을 보장하면서 설계 시간의 감소를 가져온다. 이러한 CAD 기법은 종종 실리콘 면적의 낭비를 가져올 수 있어 VLSI 모듈의 자동합성에는 설계 최적화 과정이 포함되어야 한다. FSM을 하드웨어로 구현할 때에도 규칙적이고 구조적인 설계가 되어야하고, 구현된 회로의 면적과 성능이 최적화되어야 한다. FSM의 조합 논리 부분을 PLA로 구현하면 이러한 조건을 만족시킬 수 있다. PLA는 AND-OR 2단의 규칙적인 구조를 갖고있어 레이아웃 과정이 간단하고, 논리 최소화 tool[3]과 folding, partitioning 등의 topological optimization[4]을 수행하여 면적을 최적화할 수 있어, 짧은 시간안에 적은 실리콘 면적으로 상위 수준 언어로 기술된 동작사양을 만족하는 FSM 회로를 구성할 수 있다.

본 논문에서는 이식성이 뛰어나고, 시스템의 동작과 구조 기술이 용이하며, Thor functional/behavioral 시뮬레이터의 모델링 언어로 사용되는 CHDL (C hardware description language)[1,2]로 부터 정보를 추출하여 자동적으로 면적이 최적화된 PLA로 구현된 FSM을 합성하는 시스템의 개발에 대하여 기술한다. 시스템의 전체적인 구성도는 그림 1에 나타나 있다. CHDL로 기술된 순차 회로를 parsing하여 symbolic 상태 천이표를 추출해내고, 이 상태표를 이용하여 상태 최소화와 상태 할당을 수행한다. 이후 논리 최소화 과정을 거쳐 면적이 최적화된 PLA 구조의 FSM을 생성한다. 출력으로 나온 tt-format[3]으로 표현된 PLA는 본 연구실에서 개발한 PLA generator에 pipeline 되어 CIF 형태로 변환되어 최종 레이아웃이 생성된다.

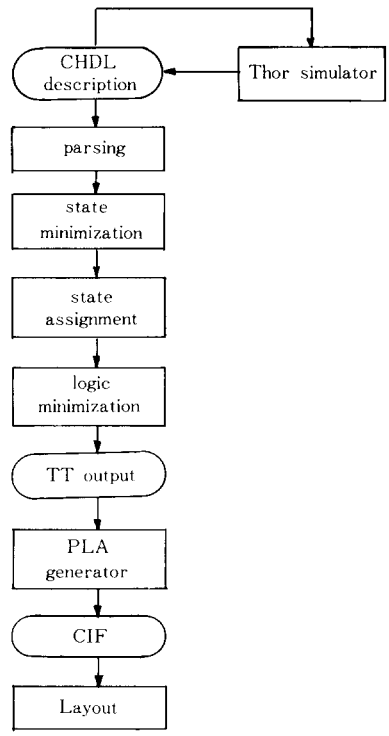

그림 1. FSM 합성 흐름도
Fig. 1. FSM synthesis flow diagram.

Ⅱ. CHDL로 부터의 FSM 합성

1. CHDL을 이용한 FSM 구조표현

상위 수준 기술언어는 특정한 하드웨어에 대한 구조적인 정보없이 machine의 동작 양식을 기술하여 이의 동작 확인과 하드웨어 설계가 가능하도록 한다. 본 시스템에서 사용하는 CHDL은 그림 2의 예제 프로그램에서 볼 수 있듯이, 인터페이스 부분, 초기화 부분, 동작 기술 부분으로 구성되어 있다. 인터페이스 부분은 입력, 출력 및 내부 상태 신호를 선언하며, 동시에 외부와의 인터페이스를 나타낸다. 초기화 부분은 회로의 초기화를 나타내며 시뮬레이션할 때 참조되고, 동작 기술 부분은 회로의 실제적인 동작을 나타낸다. CHDL은 switch 문과 if 문의 자유로운 사용과 default의 허용으로 machine의 동작 양식을 효율적으로 기술할 수 있어 설계의 효율을 높일 수 있다. 그림 2의 동작 기술 부분에서 fpack( )은 이진 벡터를 정수로 변환시키는 라이브러리 함수이고, funpack( )은 정수를 이진 벡터로 바꾸는 함수이다. 이러한 CHDL 파일은 하나의 하드웨어 모듈을 표현한다.

2. 상태표 생성

FSM의 동작을 기술한 CHDL을 읽어들여 상태 최소화와 상태 할당을 원활하게 하기 위하여 상태천이에 대한 정보를 유지하는 중간 형태를 생성한다. 그림 3은 이 중간 형태인 상태 천이에 대한 자료구조이다.

```
# define START 0
# define STAE1 1
# define STATE2 2
# define STATE3 3
# define STATE4 4
MODEL(example) {
/* Interface Section */
  IN-LIST
    SIG(insig1);
    SIG(insig2);
    GRP(ingrp, 2);/*2-bit signals */
  ENDLIST;
  OUTLIST
    SIG(outsig);
    GRP(outgrp, 2);/*2-bit signals */
  ENDLIST;
  ST_LIST
    GRP(state, 3);/*state register is 3-bit wide */
  ENDLIST;
  /*Initialize Section */
    INIT
      outsig=ONE;
      funpack(state, 0, 2, START);
    ENDINIT;
  /*Behavioral Description Section */
    switch(fpack(state, 0, 2)){
     case START:if(insigl==ONE){
                  outsig=ONE;
                  funpack(state, 0, 2, STATE1);
                 } else {
                  outsig=ZERO;
                  outgrp[0:1]=3;
                  funpack(state, 0, 2, START);
                 }
                  break;
     case STATE1:outgrp[0:1]=0;
                  funpack(state, 0, 2, STATE4);
                  break;
     case STATE2:switch(ingrp[0:1]){
                  case 1:funpack(state, 0, 2, STATE3);
                         break;
                  case 2:outsig=ONE;
                         funpack(state, 0, 2, STATE1);
                         break;
                  default:outgrp[0:1]=1;
                         funpack(state, 0, 2, STATE2);
                 }
                 break;
     case STATE3:if(ingrp[0:1]==0){
                  outgrp[0:1]=1;
                  funpack(state, 0, 2, STATE2);
                 }
                 break
     case STATE4:if(!(ingrp[0:1]==1||insig2==ZERO))
                  funpack(state, 0, 2, STATE1);
                 break;
     default;
    }
}
```

그림 2. CHDL 프로그램 예
Fig. 2. An example CHDL program.

```
struct cube_struct {
      char   *In;          /* input signals */
      char   *Out;         /* output signals */
      char   *Pstate;      /* present state */
      char   *Nstate;      /* next state */
      int    SymPstate;    /* symbolic present state */
      int    SymNstate;    /* symbolic next state */
      int    mark;
      struct cude_struct  *next;
     }
```

그림 3. 중간 형태의 자료구조
Fig. 3. Data structure of intermediate form.

여기에서 In, Out은 입,출력 신호를, SymPstate와 SymNstate는 현재 상태와 다음 상태에 대한 symbolic 표현이고, Pstate, Nstate는 현재 상태, 다음 상태를 나타내는 비트 벡터로 후에 상태 할당 과정에서 정해진 비트 벡터를 저장한다. 한편 CHDL에서 지원되는 제어 구문(if-, switch statements) 에 대한 정보를 유지하기 위한 자료구조는 그림 4에 보였다. 그림 4(a)에서 cond_next는 그림 2의 'case STATE4:if(!(ingrp[0:1]==1||insig2==ZERO))'와 같이, 입력 신호의 조건이 or(||), and(&&), not(!)등의 connection을 갖을 수 있으므로 이를 지원하기 위하여 사용된다.

```
struct condition_struct {
      char   *in;              /* input signals */
      int    ps;               /* symbolic present state */
      struct condition_struct   *cond_next;
      struct condition_struct   *next;
     } *condList, *ifcondList;
```

(a)

```
struct switch_struct {
      int    high;          /* high position in bit pattern */
      int    low;           /*low position in bit pattern */
      int    status;        /* distinguish 'In' and 'SymPstate' */
      struct condition_struct   *cndt;/* for 'default statement*/
      struct switch_struct      *next;
     } *switchList;
```

(b)

그림 4. 제어 구문을 위한 자료구조
(a) if문 처리를 위한 자료구조
(b) switch문 처리를 위한 자료구조
Fig. 4. Data structures for control constructs.
(a) data structure for if statement,
(b) data structure for swith statement.

FSM의 동작을 기술한 CHDL로 부터 상태표가 구성되는 과정은 다음과 같다.

[단계 1] 인터페이스 부분의 입력, 출력 그리고 상태 선언문을 읽어 입력과 출력에 대한 비트 벡터의 크기를 결정한다.

제어 구문의 semantics를 유지하기 위하여 condList와 ifcondList라는 전역변수를 사용하며, condList는 PLA의 입력 조건을 나타내고 ifcondList는 각 조건문의 조건을 나타낸다. SwitchList는 default를 처리할 때 참조된다. IfcondList, switchList, condList 는 인터페이스 부분의 처리 과정에서 다음과 같은 초기값을 갖는다.

condList: (in:--- ps:?)
ifcondList: NULL
switchList: NULL

CondList에서 ps:?는 ps에 상태값이 할당되지 않았음을 의미한다.

[단계 2] 초기화문(initialize section)을 읽어들여 initial cube를 구한다.

그림 2는 초기화가 기술되어 있으므로 다음과 같은 initial cube가 생성된다.

initial cube: … ? START 1 …

[단계 3] FSM의 실제적인 동작을 기술한 문장을 처리한다.

이 과정에서 nested 조건문을 처리하기 용이하도록 조건문을 읽었을 때 그 조건에 해당하는 벡터가 ifcondList에 push되고 이 ifcondList와 현재의 condList를 AND한 벡터 값이 condList에 push된다. 그리고 조건절이 끝나면 ifcondList와 condList에서 그 조건절에서 push된 조건에 해당하는 벡터가 pop된다

예를들어 'case START'를 읽었을 때 condList에 (In:… ps:START)이 push된다. 따라서 condList는 (In:… ps:START)→(In:… ps:?)이 된다. 다음 'if (insigl==ONE)'을 읽었을 때 ifcondList에 (In:1… ps:?)이 push되고, condList는 (In:… ps:START)와 (In:1… ps:?)를 AND한 (In:1… ps:START)가 push되어 (In:1… ps:START)→(In:… ps:START)→(In:… ps:?), ifcondList는 (In:1… ps:?)이 된다. 다음 출력 신호와 상태 천이에 대한 동작을 읽어 이 block에서 생성되는 cube는 (input part:1…START output part: STATE 1…)이 된다. 다음 '}'를 만나면 if (insig==ONE)조건이 끝났음을 의미하므로 ifcondList는 NULL이 되고, condList는 (In:1… ps:START)이 pop되어 (In:… ps:START)→(In:… ps:?)이 된다.

위와 같은 과정을 수행한 후 다음과 같은 상태 천이표가 구성된다.

표 1. 그림 2의 CHDL 기술로부터 생성된 상태 천이표

Table 1. State transition table from the CHDL description of figure2.

1---	START	STATE1	011
01--	START	START	1--
----	STATE1	STATE4	-00
--11	STATE2	STATE2	-01
--10	STATE2	STATE1	1--
--01	STATE2	STATE3	---
--00	STATE2	STATE2	-01
--00	STATE3	STATE2	-01
-100	STATE4	STATE1	---
-11-	STATE4	STATE1	---

FSM은 정상적인 경우에 현재 상태에서 주어진 입력에 따라 정해진 출력 신호를 내보내고 다음 상태로 간다. 하지만 정의되지 않은 입력이 들어오면 출력 신호와 다음 상태값을 예측할 수 없게 된다. 본 시스템은 기술되지 않은 입력이 들어왔을 때 설계자가 요구하는 상태로 보낼 수 있도록 초기화 부분에 기술할 수 있고, 기술되어 있으면 "subtract"를 수행하여 조정된 상태 천이표를 생성한다. 그림2의 경우 입력 file에 초기화가 기술되어 있으므로 initial cube (input part:… ?) output part:START 1…) 에 대하여 위의 표 1로 subtract한 결과는 다음과 같다.

00--	START	START 1--
--1-	STATE3	START 1--
--01	STATE3	START 1--
-0--	STATE4	START 1--
--01	STATE4	START 1--

따라서 조정된 상태 천이표는 다음과 같다.

표 2는 표1에 비하여 cube의 수가 많아졌지만 FSM 기술문에 명시되지 않은 입력이 들어왔을 때에도 설계자가 요구하는 적절한 대처를 할 수 있어 회로의 안정성을 높일 수 있다.

Ⅲ. 상태 최소화

설계자가 기술한 FSM에 대하여 state diagram을 구성해보면, 종종 diagram이 여분의 상태(redundant

표 2. 조정된 상태 천이표
Table 2. Adjusted state transition table.

00--	START	START	1--
01--	START	START	011
1---	START	STATE1	1--
----	STATE1	STATE4	-00
--11	TTATE2	STATE2	-01
--00	STATE2	STATE2	-01
--10	STATE2	STATE1	1--
--01	STATE2	STATE3	---
--01	STATE3	START	1--
--1-	STATE3	START	1--
--00	STATE3	STATE2	-01
--01	STATE4	START	1--
-100	STATE4	STATE1	---
-11-	STATE4	STATE1	---
-0--	STATE4	START	1--

state)를 포함하는 경우가 있다. 여분의 상태란 initial state에서 어떠한 input signal sequence에 의해서도 도달될 수 없는 상태 또는 다른 상태에 의해서 그의 기능이 수행될 수 있는 상태를 말한다.[7] FSM의 구현을 위해 필요한 메모리 소자의 수는 상태의 수에 직접적으로 관련이 있다. 즉, n개의 상태를 가진 FSM의 경우 최소한 $k=\lceil \log n \rceil$개의 상태 변수가 필요하다. 따라서 여분의 상태를 제거하여 상태의 수를 최소화하는 것은 많은 경우에 회로로 구현할 때 복잡도와 비용을 감소시킬 수 있다. 더구나 구현된 순차 회로가 여분의 상태를 포함하지 않으면 회로의 테스트와 고장 진단이 대단히 간단해진다. 본 시스템에서 구현한 상태 최소화 알고리듬은 다음과 같다.

[단계 1] 도달할 수 없는 상태들(unreachable states)의 제거
초기 상태로부터 어떠한 입력 조건으로도 도달할 수 없는 상태는 machine의 동작에 아무런 영향을 주지 못하므로 제거한다.

[단계 2] Compatible 상태를 찾아 새로운 상태로 대치
두 상태 SA, SB에서 어떠한 입력 신호에 대해서도 출력신호가 같으면 두 상태는 compatible하다. 이러한 compatible 쌍을 찾아 새로운 상태로 대치하여 reduced machine을 만드는 과정은 다음과 같다.
(1) 병합표(merger table)를 이용하여 가장 큰 compatible 쌍의 집합을 찾는다.
(2) Compatibile 쌍 사이의 의존 관계를 나타내는 compatiblility graph를 구한다.
(3) 위의 두 단계에서 얻은 결과를 이용하여, 최적의 compatible 쌍을 구한다.[7]

상태 최소화가 일어나면 해당되는 부분에 대한 정보와 최적의 보정된 결과를 설계자에게 알려주고 다음 단계의 수행을 계속한다.

표 3에 상태표의 예를 보였다. 이 상태표에서 state_4는 start상태로 부터 어떠한 input sequence로도 도달할 수 없는 상태이므로 제거한다. 단계2에서 병합표를 이용하여 maximum compatibles를 구하면 state_3,(start, state_2, state_7), (start, state_5, state_7), (start, state_6)의 4개의 상태로 줄일 수 있다. 다음 compatiblility graph를 이용하여 closed cover를 바꿈으로서 redundant한 부분을 제거하여 state_3, (start, state_2), (state_5, state_7), (start, state_6)으로 줄일 수 있다. 이 3개의 병합된 상태와 state_3를 state1, state2, state3, state4로 나타내어 표 4의 상태표를 구할 수 있다.

표 3. 상태 최소화 전의 상태표
Table 3. State table before state minimization.

1	start	state_7	0
0	state_2	state_2	0
1	state_2	state_3	0
0	state_3	state_6	0
1	state_3	start	1
1	state_4	state_6	1
0	state_5	state_2	0
1	state_5	state_5	0
0	state_6	state_7	1
1	state_6	state_5	0
0	state_7	start	0

표 4. 상태 최소화 후의 상태표
Table 4. State table after state minimization.

0	state1	state4	0
1	state1	state1	0
0	state2	state3	0
1	state2	state4	1
0	state3	state1	0
1	state3	state2	0
0	state4	state2	1
1	state4	state4	0

Ⅳ. 상태 할당

상태 할당이란 symbolic으로 표현된 machine의 내부 상태에 이진 벡터를 할당하는 과정이다. Machine의 메모리 부분이 차지하는 면적은 상태의 수에 대하여 logarithmic으로 증가하는데 비하여 조합 논리 부분은 combinatorial 또는 지수적으로 증가하기 때문에, 비용은 보통 조합 논리 부분의 면적으로 정의된다. 따라서 각 상태에 이진 벡터를 할당할 때에는 조합 논리 부분의 비용을 최소화 하도록, 조합 논리 부분이 이단 논리(two-level logic), 다단 논리(multi level logic)여부에 따라 면적 공식(cost function) 을 고려하여 상태 할당을 해야 한다. PLA에 기초한 FSM의 경우, PLA의 면적이 행과 열에 의해 결정되며 행에 해당하는 product term의 수가 상태 할당의 영향을 크게 받는다.

PLA의 면적을 최소로 하는 상태 할당은 computational complexity가 높은 과정으로 solution space 를 간단히 하기위한 가정이 필요하다. 우선 PLA의 면적을 감소시키기 위한 folding과 partitioning 같은 topological optimization[4]은 칩으로 구현할 때 이루어지므로 이에 대한 고려를 하지않고, PLA의 면적에 대한 요소를 간단히 한다. 또한 FSM의 메모리 부분에 대해서도 여러 플립플롭(D, T, J/K)이 있고 적용하는 부분에 따라 서로 장단점이 있지만, 여기에서는 D플립플롭을 상태 정보를 저장하는 메모리로 가정한다. 따라서 궤환(feedback)되는 상태 벡터에 의한 PLA의 열이 상태 벡터의 비트수 * 3이 되므로 식(1)과 같은 FSM의 PLA 부분에 대한 면적 공식을 얻을 수 있다.

$$(i*2+s*3+o)*\text{product term의 수} \qquad (1)$$

- i : FSM의 입력 변수의 수
- o : FSM의 출력 변수의 수
- s : FSM의 상태 벡터의 비트 수

상태 벡터의 비트 수와 product term의 수가 상태 할당 과정에 의존하고 식 (1)에 의해 PLA의 면적이 산출되므로, PLA의 면적은 상태 할당에 복잡한 의존 관계가 있다. 따라서 PLA의 면적을 최적화하기 위한 상태 할당 문제를 다음과 같은 두가지 접근 방법을 통해 단순화 시킨다.

1) 상태 코드의 길이를 고정시켜놓지 않고, PLA의 행의 수를 최소화하는 할당 중에서 가장 작은 코드 길이를 갖는 할당을 찾는다.[5]

2) 주어진 벡터 길이를 갖는 할당 중에서 행의 수(product term의 수)를 최소화 하는 할당을 찾는다.[9]

위의 두 접근 방법에 대하여 여러가지의 효율적인 알고리듬이 개발되었다. 특히 1)의 접근 방법을 사용한 KISS의 경우 논리 최소화 tool인 ESPRESSO-MV[8]를 각 상태에 벡터를 할당하기 전에 이용하여 상태 할당의 효과를 보다 효율적으로 예측할 수 있는 방법을 제안하였다. 본 시스템에서는 2)의 접근 방법을 사용하여 최소 길이를 갖는 상태 벡터를 산출하고 이 길이 내에서 product term의 수를 최소화하는 상태 할당을 수행한다.[11]

1. 조건행렬(Constraint Matrix)

조건행렬은 병합될 수 있는 symbolic implicant들의 집합을 표현하는 행렬로서 상태표에서 입력에 대하여 출력과 다음 상태가 일치하는 현재 상태들의 집합으로 부터 구성한다.

표 5. 상태표
Table 5. A state table.

Present State	Input=0		Input=1	
	Next State	Output	Next State	Output
state_1	state_8	11	state_1	00
state_2	state_8	11	state_4	01
state_3	state_1	00	stpte_6	10
state_4	state_8	11	state_5	10
state_5	state_8	11	state_5	10
state_6	state_8	11	state_4	01
state_7	state_2	10	state_8	11
state_8	state_1	00	state_7	00

예를들어 표 5의 상태표에서 입력이 0일때 다음상태가 state_1 이고 출력이 00인 symbolic implicant는 다음과 같다.

0 state_3 state_1 00
0 state_8 state_1 00

여기에서 state_3과 state_8사이의 거리가 1이 되도록 encoding을 할 경우 두 implicant가 병합되어 하나의 implicant로 대치되어 product term이 줄어드는 이득이 있다. 위와같이 표 5의 상태표로 부터 병합될 수 있는 모든 symbolic implicant의 집합을 찾아 그림 5의 조건 행렬을 구성할 수 있다.[5] 그림 5에서 행은 병합될 수 있는 각 symbolic implicant의 집합이고 열은 상태를 나타낸다.

2. Similarity 그래프

Similarity 그래프의 정점은 상태를 표시하며 간선

1	1	0	1	1	1	0	0
0	0	1	0	0	0	0	1
0	0	0	1	1	0	0	0
0	1	0	0	0	1	0	0

그림 5. 조건 행렬
Fig. 5. Constraint matrix.

은 두 상태 사이의 거리 조건(distance constraint)관계를 나타낸다. 이러한 similarity 그래프는 조건 행렬로 부터 구성하며 두 상태 사이의 거리 조건 관계를 나타내는 간선은 다음과 같은 두개의 필드를 가진다.

gravitational force:두 상태 사이의 거리가 1일때 두 상태의 병합에 의하여 줄어들 수 있는 PLA 행의 수를 나타낸다.

loss:두 상태가 아닌 다른 상태들의 거리 조건에 의해서 만족되어질 수 있는 정도를 나타낸다.

그림 5의 조건행렬을 이용하여 그림 6의 similarity 그래프를 구성한다. 구하는 과정은 예를들어 state_2 (S2)와 state_6(S6)사이의 관계를 보면, 그림5의 조건 행렬에서 다음과 같이 1행과 4행에서 이 두 상태가 1임을 알 수 있다.

1행 1 1 0 1 1 1 0 0
4행 0 1 0 0 0 1 0 0

따라서 이 두 상태 사이의 간선에서 gravitational force 필드는 2가 되고 loss 필드는 3이 된다.

그림 6의 similarity 그래프를 보면 간선(두 상태들 사이의 거리 조건 관계)이 너무 많아 이들을 모두 만족시켜 주는 할당은 불가능하다. 더구나 loss가 많은 것은 그만큼 중복이 많음을 의미한다. 따라서 중복되는 것을 제거하여 거리 조건 관계를 간단히 할 필요가 있다. 이 과정은 다음과 같다.

Similarity 그래프에서 gravitational force가 가장 큰 간선 중 loss가 작은 간선을 선택한다. 선택된 간선은 reduced similarity 그래프의 간선에 그때의 gravitational force를 기입한다. 또한 조건 행렬에서 두 상태를 포함하는 행을 찾아 두 상태와 다른 나머지 상태들 사이의 gravitational force는 1씩 감소시킨다. 이 과정을 similarity 그래프에서 모든 간선의 gravitational force가 0이 될때까지 반복한다. 그림6의 similarity 그래프로 부터 구한 reduced similarity 그래프를 그림 7에 보였다.

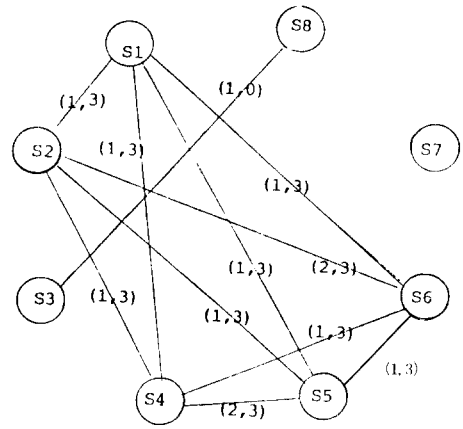

그림 6. 그림 5의 조건 행렬로 부터 구한 similarity 그래프
Fig. 6. Similarity graph obtained from the constraint matrix of figure5.

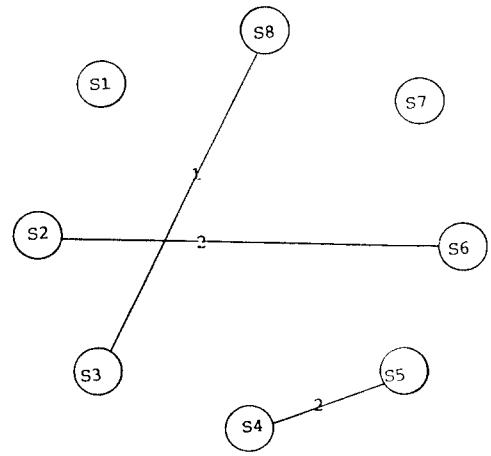

그림 7. Reduced similarity 그래프
Fig. 7. Reduced similarity graph.

3. Encoding

조건 행렬의 조건을 만족하는 상태 벡터를 찾는 것은 computational complexity가 높은 작업이다. KISS의 경우 상태 벡터의 길이를 늘려가며 최대한의 조건을 만족시키지만, product term의 수가 줄어도 상태 벡터의 길이가 증가하여 product term의 감소가 PLA 면적의 감소와 일치하지는 않는다. 본 알고리듬은 상태 벡터의 길이를 최소 크기로 고정시켜놓고, 조건 행렬을 바탕으로 reduced similarity 그래프를

만족하는 상태 벡터를 생성한다. 상태가 할당되는 과정은 다음과 같다.

[단계 1] 조건 행렬을 이용하여 global하게 같은 dimension 안에 있으면 병합이 많이 일어나 product term의 감소가 큰 상태 집합을 결정한다.

[단계 2] 집합에 있는 상태들 사이의 상대적인 위치를 reduced similarity 그래프를 이용하여 결정한다.

[단계 3] 모든 상태에 값이 할당될 때 까지 반복 수행한다.

예를들어 그림 5의 첫번째 행을 보면 다섯개의 상태 중 4개의 상태 벡터가 2-dimension을 이루도록 상태 할당이 이루어진다면 4개의 product term이 1개로 감소한다. 또한 그림 7의 reduced similarity 그래프를 보면 state_2와 state_6의 거리가 1이고, state_4와 state_5의 거리가 1이 되도록 하면 각각 product term의 갯수가 2개씩 줄어들을 수 있다. 따라서 조건 행렬과 reduced similarity 그래프를 모두 참조하여 그림 8과 같이 state_2, state_4, state_5, state_6과 2-dimension을 이루게 최적의 상태 할당을 한다.

state_4	state_5
state_2	state_6

state_2 010
state_4 000
state_5 001
state_6 011

그림 8. 최적 enoding의 예
Fig. 8. An example of optimal encoding.

다음 조건 행렬에서 2번째 행의 state_3과 siate_8에 거리가 1이 되도록 100과 101이 할당된다. 나머지 상태인 state_1과 state_7에는 각각 110과 111이 할당되어 그림 9(a)의 상태 벡터가 생성된다.

4. 이동(Shifting)

상태 벡터를 0으로 할당할 경우 가장 이득이 많은 상태를 찾아 그 상태를 0으로 하면서 위에서 구한상태 벡터들 사이의 거리 관계를 해치지 않도록 상태 벡터를 이동한다. 표 5의 상태표에서 다음 상태가 state-4인 symbolic implicant 중 출력 신호가 0인 implicant는 하나도 없지만 다음 상태가 state-1인 implicant에는 3개가 있어 state-1에 000을 할당한다면 PLA의 면적을 더욱 줄일 수 있다. 모든 상태에 대하여 110을 'XOR'하면 그림 9(b)의 이동된 상태 벡터를 구한다.

state_1	110	state_1	000
state_2	010	state_2	100
state_3	100	state_3	010
state_4	000	state_4	110
state_5	001	state_5	111
state_6	011	state_6	101
state_7	111	siate_7	001
state_8	101	state_8	011
(a)		(b)	

그림 9. 상태 벡터 할당 결과
(a) 이동 전의 상태 벡터
(b) 이동 후의 상태 벡터
Fig. 9. Result of state vector assignment
(a) State vector before sifting,
(b) State vector after shifting.

V. 실험 결과

본 논문에서 제시된 알고리듬은 UNIX상에서 C로 구현하였고, 제안된 알고리듬의 성능을 평가하기 위하여 표준 테스트 회로인 MCNC benchmarks에 대하여 실험하였다. 그림10은 MCNC benchmarks 의 하나인 tbk 회로로 본 시스템(Albatros)에서 상태 할당을 수행하고 논리 최소화 과정[8]을 거친 다음, PLA generator를 거쳐 생성된 레이아웃이다. 구성된 조건 행렬과 similarity 그래프를 만족하는 최적의 상태 벡터를 생성하고, 상태 벡터의 이동에서 567개의 product term의 감소가 일어나서 원래 회로의 1569개에서 최적화된 후 52개로 줄어들었다. 표 6은 본 시스템과 KISS를 MCNC benchmarks에 대하여 비교한 결과이다. Ex1과 ex2 회로를 제외하고 5%에서 20%의 성능향상을 보였고, 전체 benchmarks에 대하여 평균 10%정도의 면적 측면의 성능 향상을 보였다.

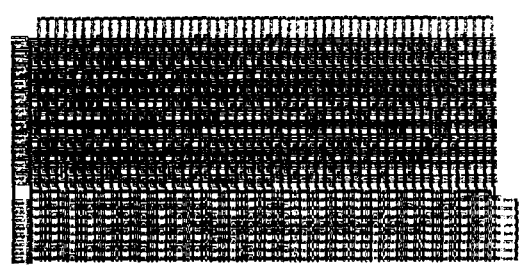

그림10. MCNC 벤치마크 회로 'tbk'의 최적화 후 레이아웃
Fig. 10. The optimized layout of 'tbk' circuit.

표 6. Albatros와 KISS의 성능 비교
Table 6. Performance comparison of Albatros with KISS.

				Albatros			KISS		
Examples	#ins	#outs	#states	#bits	#cubes	area	#bits	#cubes	area
bbara	4	2	10	4	28	616	5	26	650
bbsse	7	7	16	4	31	1023	6	27	1053
bbtas	2	2	5	3	11	165	3	13	195
cse	7	7	16	4	48	1584	6	45	1756
dk15	3	5	4	2	19	323	6	17	391
dk16	2	3	27	5	80	1760	12	55	2035
dk17	2	3	8	3	19	304	6	19	361
dk27	1	2	7	3	7	91	4	9	117
dk512	1	3	15	4	22	374	7	18	414
ex1	9	19	20	5	55	2860	7	42	2436
ex2	2	2	19	5	36	756	6	31	744
ex3	2	2	10	4	20	342	6	18	432
ex5	2	2	9	4	17	306	5	15	315
ex6	5	8	8	3	26	702	5	24	792
keyb	7	2	19	5	49	1519	8	47	1880
sand	11	9	32	5	100	4600	6	95	4655
tbk	6	3	32	5	52	1560	-	-	-

#ins 입력 신호의 수 #bits :상태 벡터의 비트수
#outs :출력 신호의 수 #cubes:product term의 수
#states:상태의 수

Ⅵ. 결 론

Thor functional/behavioral 시뮬레이터의 모델링 언어인 CHDL로부터 상태 최소화와 상태 할당을 수행하여 PLA에 기초한 FSM을 자동적으로 생성시키는 시스템을 개발하였다.

본 시스템은 기술된 FSM을 Thor 시뮬레이터를 이용하여 동작을 확인한 후 FSM을 자동적으로 생성하고, 생성된 FSM을 다시 모델링하여 동작의 검증을 행한다. 따라서 신뢰성을 갖는 FSM을 생성할 수 있고, 설계시간을 단축할 수 있다. 본 시스템에 적용된 상태 할당 알고리듬은 수행 시간이 빠르고, 최적의 상태 할당을 수행하여 VLSI 시스템의 제어부를 비롯한 여러 순차 회로를 효율적으로 설계할 수 있다. 또한 SSC(sogang silicon compiler)[10]의 control synthesis 결과로 나온 동작 기술을 입력으로 받아들여 controller를 생성한다.

앞으로 조건 행렬을 구성할 때 논리 최소화 tool을 이용하고 don't care 조건을 고려하여 조합 논리 부분을 더욱 최소화하는 routine을 구현하여 성능을 개선할 수 있다.

參 考 文 獻

[1] R. Alverson, T. Blank, K. Choi, S. Y. Hwang, A. Salz, L. Soule, T. Rokicki, "Thor user's manual: tutorial and commands," Technical Report: CSL-TR-88-348, Stanford University, January 198 .

[2] D. Bostick, G. D. Hachtel, R. Jacoby, M. R. Lightner, P. Moceyunas, C.R. Morrison, D. Ravenscroft, "The boulder optimal logic design system," in *Proc. ICCAD*, Nov. 1987, pp. 62-65.

[3] R.K. Brayton, G.D. Hachtel, C.T. McMullen, A. Sangiovanni-Vincentelli, *Logic Minimization Algorithms for VLSI Synthesis*, Kluwer Acadamic Publishers: Boston, Mass., 1984.

[4] G. De Micheli, M. Hoffman, A.R. Newton, A. Sangiovanni-Vincentelli, "A design system for PLA-based digital circuits," *Advances in computer-Aided Engineering*, A. Sangiovanni-Vincentelli (Ed.), Jai Press Inc., 1985.

[5] G. De Micheli, R.K. Brayton, A. Sangiovanni-Vincentelli, "Optimal state assignment for finite state machines," *IEEE Trans. Computer-Aided Design*, vol. CAD-4, no. 3, pp. 269-285, July 1985.

[6] S. Devadas, H. Ma, A. R. Newton, A. Sangiovanni-Vincentelli, "MUSTANG: state assignment of finite state machine targeting multilevel logic implementations," *IEEE Trans. Computer-Aided Design*, vol. CAD-7 no. 12, pp. 1290-1299, Dec. 1988.

[7] Zvi Kohavi, *Switching and Finite Automata Theory*, McGraw-Hill, 1970.

[8] R. Rudell and A. Sangiovanni-Vincentelli, "ESPRESSO-MV: algorithms for multi-valued logic minimization," in *Proc. Custom Int. Circ. Conf.*, pp. 23-28, Oct. 1984.

[9] D. Varma and E.A. Trachtenberg, "A fast algorithm for the optimal state assignment of large finite state machines," in *Proc. ICCAD*, pp. 156-159, Nov. 1988.

[10] 전홍신, 이해동, 황선영, "서강 Silicon Compiler," 한국정보과학회 90년 춘계 학술대회 논문집, 제17권 1호, 1990년 4월, pp. 411-414.

[11] 유진수, 이기중, 황선영, "상위 수준 설계 기술에서의 FSM 자동 생성 시스템" 대한전자공학회 하계종합학술대회 논문집 제13권, 1990년 7월, pp. 533-537.

著 者 紹 介

黃 善 泳(正會員)

1954年 5月 20日生. 1976年 2月 서울대학교 전자공학과 졸업. 1978年 2月 한국과학원 전기및 전자공학과 공학석사 취득. 1986年 10月 미국 Stanford 대학 공학박사 학위 취득. 삼성반도체 주식회사 연구원, Stanford 대학 CIS 연구소 연구원. Fairchild Semiconductor 기술 자문. 1989年 3月～현재 서강대학교 전자공학과 교수. 주관심분야는 CAD 시스템, Computer Architecture 및 System Design, VLSI 설계 등임.

柳 眞 秀(準會員)

1967年 9月 16日生. 1990年 8月 서강대학교 전자공학과 졸업. 1990年 9月～현재 서강대학교 대학원 전자공학과 석사과정 재학중. 주관심분야는 FSM synthesis, logic synthesis 등임.