論文95-32A-10-12

Device fitting이 고려된 PLD 설계용 Tool 개발

(The development of a tool for PLD Design with device fitting)

元 忠 常 *, 金 熙 碩 **

(C. S. Won and H. S. Kim)

요 약

본 논문은 디바이스 피팅이 고려된 PLD 설계용 툴 개발에 관한 것이다. 디지탈 회로를 PLD로 설계하기위한 PLD 설계용 툴은 부울식 생성 단계, 핀 맵 단계, 퓨즈 맵과 JEDEC 화일 생성단계등의 여러 단계를 거쳐야 한다. 특히, 하나의 PLD로 설계할 수 없는 큰 회로를 PLD를 이용하여 설계하기 위해서는 PLD에 적합한 분할 기능인 디바이스 피팅 기능을 필요로 한다. 따라서 본 논문에서는 PLD 소자의 입출력과 지연시간을 고려하여 임의로 입출력 변수를 분할 하였고, 임의로 분할된 입출력 변수들을 디바이스 피팅하여 기존의 다른 PLD 설계용 툴(PALASM)과 결과를 비교하여 동일함을 입증하였다. 또한, 본 논문에서 개발한 PLD 설계용 툴을 이용하여 생성된 JEDEC 화일을 PLD writer (ALL-07)를 이용하여 PLD로 구현하였다.

Abstracts

This paper describes a development of the PLD design tool in considering with a device fitting. To design digital circuit with PLDs, several steps in the developed PLD design tool are needed such as Boolean description step, pin map step, FUSE map and JEDEC steps ... etc. Especially, we have considered the device fitting to design large digital circuits with PLDs developed the device fitting algorithms based on the PLD device fitting and compared with the results of a another PLD design tool(PALASM). Also, we have proved that the developed PLD design tool is successfully implemented by the connection with a PLD writer(ALL-07), in the case of design digital circuits.

I. 서 론

최근 디지털 논리 회로 설계에 고집적도 ASIC을 이용한 설계 방법들이 폭 넓게 사용되고 있으며, 특히 규칙적인 내부구조를 갖는 PLD(Programmable Logic Device)[1][2]는 설계 변경이 용이하고, 최적 회로를 설계할 수 있다는 장점 때문에 ABEL, CUPL, PALASM등과 같은 PLD 설계용 툴이 보급되어 널리 사용되고 있다.[8]

그러나, 기존의 PLD 설계용 툴은 2단 레벨(two-level)의 부울식만을 설계 대상으로 정했기 때문에, 다단 레벨(multi-level)로 기술된 부울식을 PLD에 적용하기가 매우 어려웠다.

따라서 본 논문에서는 다단 레벨로 기술한 부울식을

* 正會員, 忠州産業大學校 컴퓨터工學科
(Dept. of Comp. Eng., Chungju National Univ.)
** 正會員, 淸州大學校 電子工學科
(Dept. of Elec. Eng., Chongju Univ.)
接受日字: 1995年3月24日, 수정완료일: 1995年10月9日

2단 레벨의 구조를 갖는 PLD로 실현하기 위해 디바이스 피팅(device fitting) 알고리듬을 개발하여 PLD 설계용 툴[8]에 적용하였다.

특히, 기술된 부울식이 순차 논리 회로(sequential logic circuit)일 경우에는 순환 시간(cycle time)[3]을 고려하여야만 한다. 제안된 디바이스 피팅 알고리듬은 조합 논리 회로의 분할 시 회로의 지연(delay) 문제를 해결하기 위하여 입출력 변수의 연관관계를 알 수 있는 인접 행렬(adjacency matrix)를 구성한 다음 이 인접 행렬를 참조함으로서 회로의 중간 변수를 결정한다. 중간 변수가 결정되면 초기 입력(primary input)부터 최종 출력(primary output)까지의 가장 긴 지연을 가진 최장 경로(critical path)가 구성된다. 최장 경로의 중간 변수를 PLD의 입출력을 고려하여 우선순위를 정하여 선택된 PLD에 피팅(fitting)을 하였으며, 제안된 디바이스 피팅 알고리듬을 실제의 회로에 적용하여 효율성을 입증하였다.

Ⅱ. PLD을 이용한 디지털 회로 설계

PLD 설계 툴을 이용하여 디지털 논리 회로를 설계하는 과정은 순차 회로 설계 시에 상태도를 상태 그래프 편집기(State graphic editor)[8]에 입력하여, 상태표와 부울식을 추출한다. 상태 그래픽 편집기는 아이콘 메뉴(icon menu) 방식을 사용하였으며, 마우스의 클릭(click) 상태에 따라 현재 상태와 다음 상태를 받아 들여 상태표를 생성하는 편집기로서, 그림 1은 상태 그래픽 편집기를 이용한 설계예제로서 스텝 모터의 상태도를 작성한 것이다. 상태 그래픽 편집기에서 추출된 부울식은 논리최소화과정을 거쳐 최소화된 부울식을 생성한다. 생성된 부울식은 핀 맵 에디터(Pin map editor)[7][8][9][10][11]를 이용하여 디바이스 피팅(device fitting)을 하게 된다. 여기에서 디바이스 피팅이란, 생성된 부울식의 변수들을 선택된 PLD의 핀과 일치하여 입력시키는 것을 의미한다. 그러나 다단 레벨로 기술된 부울식은 하나의 PLD로 회로를 구현할 수 없으므로 몇 개의 PLD로 분할하여 회로를 구현해야 한다. 따라서 다단 레벨로 기술된 부울식을 분할하여 여러 개의 PLD로 구현할 수 있는 디바이스 피팅 알고리듬을 필요로 하며 본 논문에서 디바이스 피팅 알고리듬을 개발하여 PLD 설계 툴에 추가하였다. 다단 레벨로 기술된 부울식을 몇 개의 PLD로 회로를 구현하려면 핀 맵 에디터에서 PLD를 선택하고 디바이스 피팅을 수행한 후, 선택한 PLD로 회로를 구현하기 위해 필요한 Fuse map과 JEDEC(Joint Electronic Device Council)[6] 화일을 생성한다. 생성된 JEDEC

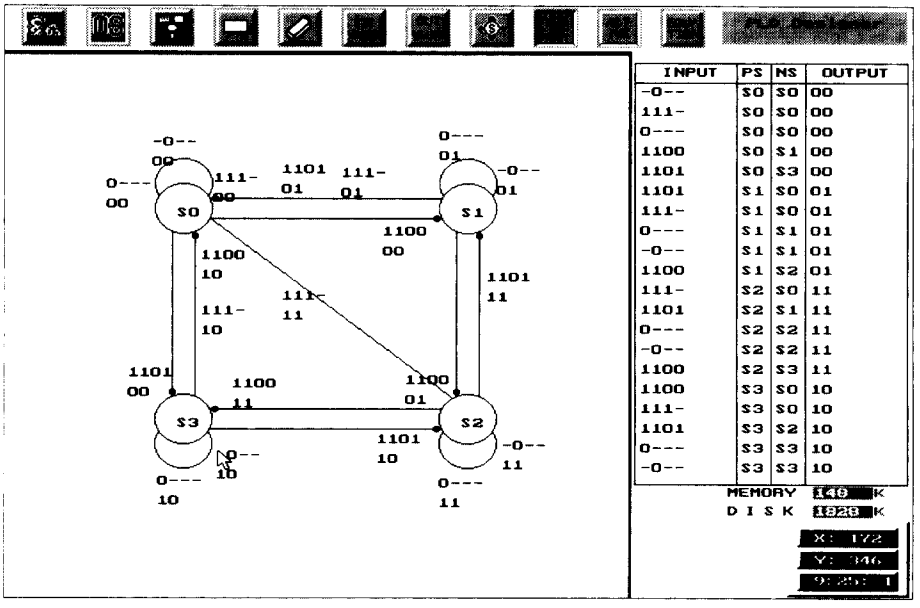

그림 1. 상태 그래픽 편집기를 이용하여 설계한 스텝모터 제어도
Fig. 1. State diagram of stepping motor controller using state graphic editor.

화일은 PLD writer(ALL-07 등)를 사용하여 PLD로 구현한다. 그림 2는 설계된 PLD 설계 툴[10]의 설계 과정이다.

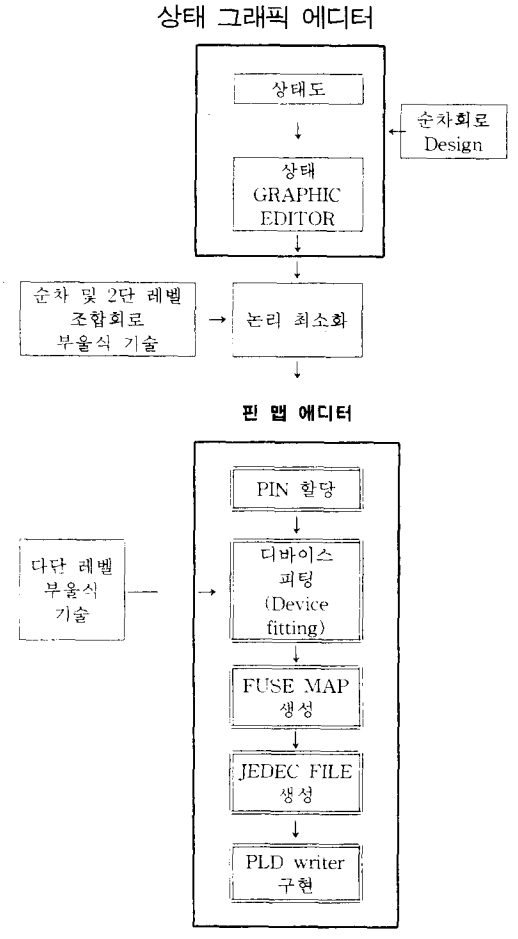

그림 2. 개발된 PLD 설계 툴의 설계 과정
Fig. 2. Design procedure of developed PLD design tool.

1. 디바이스 피팅(device fitting)을 고려한 PLD 설계

다단 레벨로 기술된 부울식을 FPGA 또는 PLD로 구현할 때 디바이스 피팅 알고리듬을 필요로 한다. 그러나 기존의 개발된 디바이스 피팅을 위한 FPGA용 분할 알고리듬[4]은 LUT(Lookup table)의 제한된 입출력 개수(5입력, 1출력)를 가지므로 기술된 부울식을 분할 처리하기 위하여 회로의 노드(node)를 인식하는 FirstFitDecreasing 알고리듬과 인식한 노드를 LUT의 제한된 크기에 맞도록 분할하는 DecomposeNode 알고리듬 및 공통된 입력이 있는 경우 분할할 수 있는 Reconverge 알고리듬과 공통된 입력 노드가 있는 경우 분할할 수 있는 MaxShareDecreasing 알고리듬들을 사용하여 다단 레벨로 분할한다. 따라서 FPGA 분할 알고리듬은 2단 레벨 구조인 PLD에 적용하기가 어렵다. 그러므로, FPGA와는 달리 다단 레벨로 기술된 부울식을 여러 개의 PLD로 분할할 수 있는 알고리듬 개발을 필요로 하며, 회로의 지연 문제를 고려하여 부울식을 분할하여야 한다. 본 논문에서 제안된 디바이스 피팅 알고리듬은 회로의 입출력 변수들을 논리 버텍스(vertex)로 표현하여 입력에서부터 출력까지의 모든 논리 버텍스들의 연관 관계를 인접 행렬형태로 저장한다. 저장된 인접 행렬은 부울식과 대응하는 그래프로 표기된 논리 회로 그래프에서 가장 큰 지연시간을 가지고 있는 최장 경로를 구하는데 필요한 참조 데이터로 이용된다.

지연 시간을 고려하여 조합 논리 회로를 분할하고자 하면 가장 큰 지연시간을 가지고 있는 최장 경로를 우선적으로 구하여, 이 경로를 구성하고 있는 논리 버텍스들, 즉, 중간 경로에 위치하고있는 논리 버텍스들을 우선적으로 선택된 PLD 블럭(block)에 피팅하여야 한다.

또한, 순차 논리 회로의 경우에는 궤환(feedback)이 있는 경로중에서 궤환이 가장 긴 경로를 구한 다음 경로에 위치한 논리 버텍스의 우선 순위를 정하여 선택된 PLD블럭에 피팅하여야 한다.

따라서, 조합 논리 회로의 경로와 순차 논리 회로의 경로 모두 인접 행렬를 참조하여 결정되는데, 최종 출력에서 인접한 논리 버텍스가 있으면 궤환이 존재하므로 순차 논리 회로로 해석되고, 인접한 논리 버텍스가 없으면 궤환이 없으므로 조합 논리 회로로 해석된다.

그림 3의 CRITICAL 알고리듬은 인접 행렬를 참조하여 조합 논리 회로의 최장 경로(longest path)와 순차 논리 회로의 순환 경로(cycle path)를 찾기 위한 알고리듬이다. 즉, 조합 논리 회로의 경우 초기 입력을 source vertex로 하며 값은 0으로 한다.

(정의 1) $s_i = \max\{ s_i, (s_q + w_{q,i})\}$, i=0,1, · · · · ,n
: source vertex(s_q)로부터 연관관계가 있는 논리 버텍스를 방문하여 방문한 논리 버텍스의 값($w_{q,i}$)을 relaxation하여 논리 버텍스의 값중 가장 큰 값을 가지고 있는 논리 버텍스까지의 경로를 최장 경로(critical path)라 한다.

인접 행렬을 참조하여 source vertex로부터 연관관계가 있는 논리 버텍스를 방문하여 방문한 논리 버텍스의 값을 relaxation[5]하여 relaxation한 논리 버텍스의 값 중 가장 큰 값을 가지고 있는 논리 버텍스까지의 경로를 최장 경로로 정한다. 모든 논리 버텍스의 값은 선택된 PLD로 논리 버텍스를 맵핑 했을 때 같은 지연값을 가지므로 값을 1로 한다. 순차 논리 회로의 경우도 궤환이 있는 논리 버텍스를 source로 하고 인접 행렬를 참조하여 방문한 논리 버텍스의 값을 relaxation하며 가장 큰 값을 갖는 경로를 구한다. 이들 경로의 논리 버텍스들은 분할 시 선택된 PLD 블록에 우선 순위를 정하여 피팅되어야 하므로 경로의 논리 버텍스를 Queue로 저장한다.

```
CRITICAL(G(V,E,W)){
    s_o = source; /* 조합논리회로의 경우는 초기
                     입력,
                  순차논리회로의 경우는 궤환이
                     되는 논리 버텍스 */
    s_0 = 0;
    for ( i=1 to n )
        s_i = w_0,i;        /* w : 버텍스의 값 */
    repeat {
        s_q가 가장 큰 값인 v_q를 선택;
        방문한 논리 vertex v_q;
        foreach (방문하지 않은 논리 vertex v_i)
            s_i = max{s_i, (s_q + w_q,i)}; /* 가
                장 큰 지연시간을 계산 */

        방문한 모든 논리 vertex의 relaxation된
                값을 s_i에 저장;
        Queue(v_m);     /* 방문한 논리 버텍스를
                queue로 저장 */

        Sort (s_i, v_m); /* 저장된 최장 경로의 논
                리 버텍스를 sort */
        }
        until(모든 논리vertex를 방문);
}
```

그림 3. 최장 경로와 순환 경로를 탐색하는 알고리듬
Fig. 3. Algorithm for search a critical path and cycle path

순환 시간은 최장 경로의 지연 시간보다 작게[3] 결정되어야 하므로 최장 경로의 지연 시간을 고려하여 순차회로의 출력에 적합한 순환 시간를 결정하여야 한다. 그림 4의 **Clock_period** 알고리듬은 **CRITICAL** 알고리듬에서 결정된 조합 논리 회로의 최장 경로와 순차 논리 회로의 순환 경로의 논리 버텍스 수와 선택된 PLD의 출력수를 고려하여 선택한 PLD에 맵핑했을 때 몇 단의 PLD로 구현되는가를 결정하고, 선택된 PLD의 지연시간을 계산하여 출력에 적합한 순환 시간을 결정하는 알고리듬이다.

```
Clock_period( s_i, v_m){
    PAL_comb_out : 선택된 PAL의
                   combinational 출력수
    PAL_Block_Delay = 선택된 PAL의 지연 시간;

    C_map=( s_i/=PAL_comb_out);
            /* s_i : 최장 경로 또는 궤환이 있는
                   경로의 논리 버텍스 수
               C_map : 선택한 PLD의 소자를
                   사용하여 최장 경로
                   또는 순환 경로를 맵핑
                   할 때 필요한 소자의 수
               */

(PAL_Block_Delay*C_map)+PAL_Block_Delay;
            /* 선택된 PLD 소자의 지연 시간을
                   고려하여 순환 시간을 계산 */
}
```

그림 4. 최소 순환 시간를 정하기 위한 알고리듬
Fig. 4. A algorithm for decision to minimum cycle time

본 논문에서는 디바이스 피팅 알고리듬의 예제로서 BCD 가산기를 선정하였으며 다단 레벨로 기술한 BCD 가산기의 부울식은 그림 5에 나타내었다.

$$b_0 = \overline{cs}\cdot\overline{x_0}\cdot y_0 + \overline{cs}\cdot x_0\cdot\overline{y_0} + cs\cdot\overline{x_0}\cdot\overline{y_0} + cs\cdot x_0\cdot y_0$$

$$b_1 = \overline{c_0}\cdot\overline{x_1}\cdot y_1 + \overline{c_0}\cdot x_1\cdot\overline{y_1} + c_0\cdot\overline{x_1}\cdot\overline{y_1} + c_0\cdot x_1\cdot y_1$$

$$b_2 = \overline{c_1}\cdot\overline{x_2}\cdot y_2 + \overline{c_1}\cdot x_2\cdot\overline{y_2} + c_1\cdot\overline{x_2}\cdot\overline{y_2} + c_1\cdot x_2\cdot y_2$$

$$b_3 = \overline{c_2}\cdot\overline{x_3}\cdot y_3 + \overline{c_2}\cdot x_3\cdot\overline{y_3} + c_2\cdot\overline{x_3}\cdot\overline{y_3} + c_2\cdot x_3\cdot y_3$$

$$c_0 = cs\cdot x_0 + cs\cdot y_0 + x_0\cdot y_0$$

$$c_1 = c_0\cdot x_1 + c_0\cdot y_1 + x_1\cdot y_1$$

$$c_2 = c_1 \cdot x_2 + c_1 \cdot y_2 + x_2 \cdot y_2$$

$$Carry = c_2 \cdot x_3 + c_2 \cdot y_3 + x_3 \cdot y_3 + b_1 \cdot b_3 + b_2 \cdot b_3$$

$$\overline{Sum_0} = \overline{b_0}$$

$$\overline{Sum_1} = carry \cdot b_1 + \overline{carry} \cdot \overline{b_1}$$

$$\overline{Sum_2} = b_2 \cdot \overline{b_2} + \overline{carry} \cdot \overline{b_2} + carry \cdot \overline{b_1} \cdot b_2$$

$$\overline{Sum_3} = carry \cdot b_1 \cdot b_2 + carry \cdot b_2 \cdot b_3 + \overline{carry} \cdot \overline{b_3} + \overline{b_1} \cdot \overline{b_2} \cdot \overline{b_3}$$

$$\overline{cs} = \overline{carry} + Reset$$

그림 5. BCD 가산기의 부울식
Fig. 5. Boolean equation of BCD adder.

BCD 가산기 부울식을 인접 행렬로 구성하면, BCD 가산기의 초기 입력에서부터 최종 출력까지의 모든 경로와 논리 버텍스들의 연관관계를 알 수 있다. 그림 5는 BCD 가산기의 부울식을 동일 형태(isomorphic) 논리 회로 그래프로 작성한 것이다.

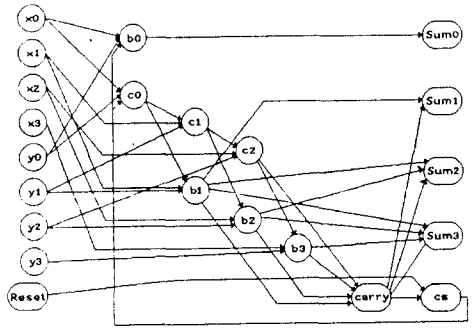

그림 5. BCD 가산기의 동일 형태 논리 회로 그래프
Fig. 5. Isomorphic logic network graph of BCD adder.

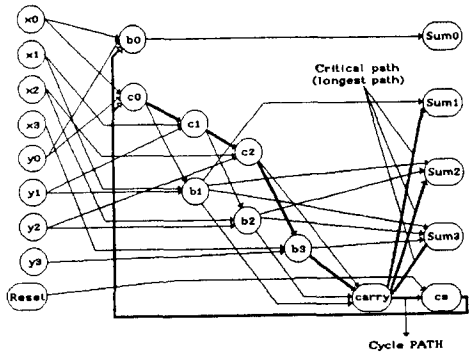

그림 6. 조합 논리 회로의 최장 경로와 순차 논리 회로의 순환 경로
Fig. 6. Critical path of combinational logic circuit and cycle path of sequential logic circuit.

BCD 가산기에 **CRITICAL** 알고리듬을 적용한 결과 그림 6과 같이 조합 논리 회로의 최장 경로는 c0부터 c1, c2, b3, carry의 중간 변수를 포함하고 있는 Sum1, Sum2, Sum3까지의 경로이고, 궤환이 있는 순차 논리 회로의 순환 경로는 c0부터 c1, c2, b3, carry, cs까지의 경로이다.

CRITICAL 알고리듬에 의해 탐색된 BCD 가산기의 조합 논리 회로의 최장 경로로 부터 생성된 netlist를 표 1에 나타내었다. 표 1에서 논리 버텍스의 relaxation된 값은 source vertex로부터 인접 행렬를 참조하여 인접한 논리 버텍스의 값을 relaxation한 값이고, 괄호안의 논리 버텍스까지의 모든 버텍스수는 모든 경로를 통한 논리 버텍스까지의 논리 버텍스수를 나타낸다. 여기에서 PLD level수는 PLD로 구현했을 때 필요한 PLD의 수이다. 또한 PLD 블록의 지연 시간(delay time)은 선택된 PLD의 블록 지연 시간을 나타낸 것이다. 표 2는 궤환이 있는 순환 경로로 부터 생성된 netlist이다.

표 1. CRITICAL 알고리듬으로부터 생성된 조합 논리 회로의 netlist.
Table 1. Generated netlist of combinational logic circuit from **CRITICAL** algorithm.

경로에서 생성된 논리 vertex(V)와 출력 변수(PO)	논리 vertex의 relaxation된 weight (논리 vertex까지의 모든 vertex수)	PLD블록의 Netlist Comb: combinational output, Reg: register output (IO:입출력, O:출력)	PLD level 수
c0 (V)	1(0)	Comb_c0(IO)	1
c1 (V)	2(1)	Comb_c1(IO)	1
c2 (V)	3(2)	Comb_c2(IO)	1
b0 (V)	1(0)	Comb_bo(IO)	1
b1 (V)	4(1)	Comb_b1(IO)	1
b2 (V)	3(2)	Comb_b2(IO)	1
b3 (V)	4(3)	Comb_b3(IO)	1
carry (V)	5(6)	Comb_carry(IO)	1
Sum0 (PO)	2(1)	Comb_Sum0(O)	2
Sum1 (PO)	6(7)	Comb_Sum1(O)	2
Sum2 (PO)	6(7)	Comb_Sum2(O)	2
Sum3 (PO)	6(7)	Comb_Sum3(O)	2

표 2. CRITICAL 알고리듬으로부터 생성된 순차 논리 회로의 netlist

Table 2. Generated netlist of sequential logic circuit from CRITICAL algorithm.

경로에서 생성된 논리 vertex(V)와 출력 변수(PO)	논리 vertex의 relaxation된 weight (논리 vertex까지의 모든 vertex수)	PLD블록의 Netlist Comb: combinational output, Reg: register output (IO:입출력, O:출력)	PLD level 수
c0 (V)	0(0)	Comb_c0(IO)	1
c1 (V)	1(1)	Comb_c1(IO)	1
c2 (V)	2(2)	Comb_c2(IO)	1
b1 (V)	1(1)	Comb_b1(IO)	1
b2 (V)	2(2)	Comb_b2(IO)	1
b3 (V)	3(3)	Comb_b3(IO)	1
carry (V)	4(6)	Comb_carry(IO)	1
cs (PO)	5(7)	Reg_cs(O)	2

BCD 가산기는 순차 논리 회로를 포함하고 있으므로 순환 경로의 지연시간에서 순환 시간을 결정해야 한다. 위의 표 1에서 생성된 netlist로부터 궤환을 가진 가장 긴 경로인 c0에서 cs까지의 중간 변수들인 c0, c1, c2, b3, carry를 PLD에 맵핑하려면 7개의 조합 논리 회로 출력을 필요로 한다. 따라서 8개의 조합 논리 회로 출력이 있는 PAL16L8을 선택하면 1개의 PLD로 중간 변수의 맵핑은 가능하다. 그러나 cs는 레지스터 (register) 출력이고, Sum0, Sum1, Sum2, Sum3은 조합 논리 회로 출력이므로 이에 적합한 PAL16R4를 선정하면 2개의 PLD(PAL16L8, PAL16R4)로 전체 회로 구현이 가능하다. PLD수에 따라 앞단의 지연시간을 고려하면 조합 논리 회로를 PLD로 구현하기 위해 선정한 PAL16L8의 지연시간 35ns와 최종 출력인 cs를 맵핑한 PAL16R4의 지연 시간 25ns를 합하여 60ns의 최소 순환 시간이 결정된다.

2. 핀 맵 에디터(PIN map editor)

다단 레벨로 기술한 부울식이 선택한 소자의 용량보다 클 경우 디바이스 피팅 알고리듬에 의해 디바이스 피팅을 수행하였다. 디바이스 피팅 알고리듬에 의해 부울식은 2개의 PLD로 구현하게 되며, 핀 할당 및 디바이스 피팅등은 핀 맵 에디터[8][9][10][11]상에서 수행하게 하였다.

BCD 가산기를 디바이스 피팅 알고리듬에 적용하면 netlist가 생성되고, 생성된 netlist를 에디터상에서 직접 보면서 디바이스 피팅을 할 수 있도록, 에디터에 선택한 소자와 소자의 각 핀에 맞는 입력 및 중간변수, 출력 변수들을 나타내었다. 또한 선택된 PLD의 지연시간을 계산하여 최소의 순환 시간도 에디터에 함께 나타내었다.

그림 7은 디바이스 피팅 알고리듬을 적용해 2개의

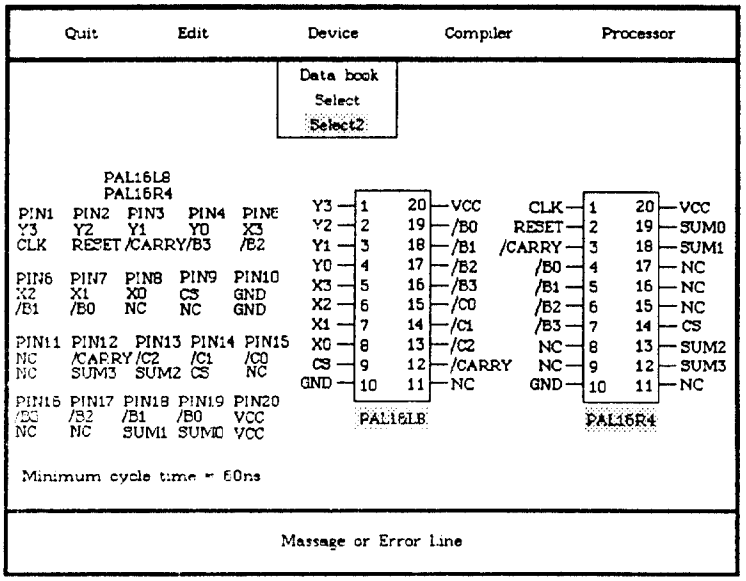

그림 7. BCD 가산기의 핀 맵 에디터

Fig. 7. The PIN map editor of BCD adder.

소자로 분할된 BCD 가산기의 핀 맵 에디터이며, 그림 5의 부울식과는 다르게 출력이 부논리로 표시된 것은, 선택된 PAL16L8의 출력이 부논리의 출력형태이기 때문이다.

디바이스 피팅을 이용한 핀 할당이 되고 난 다음 PLD로 구현하기 위해 JEDEC[7] 화일을 생성하여야 한다. 핀 할당된 부울식의 입력 변수와 출력 변수의 수를 고려하여 JEDEC 화일 생성 알고리듬에 의해 FUSE MAP과 JEDEC 화일을 생성된다. JEDEC 화일 중 CHECKSUM은 FUSE[7][8][9]된 상태의 합이다.

그림 5에서 예제로 선택한 BCD 가산기의 부울식을 선택한 PLD인 PAL16L8과 PAL16R4로 핀 할당한 후 FUSE MAP과 JEDEC 화일 생성 알고리듬으로 각 소자의 핀 할당된 부울식을 구현하면 그림 8, 9와 같은 JEDEC 화일이 생성되며, 생성된 각각의 JEDEC 화일을 PLD writer(ALL-07)에 연결하여 2개의 PLD로 구현된다.

```
PAL16L8
_BCDL8.*
QP20*
QF2048*
G0*F0*
L0000 11111111111111111111111111111111*
L0032 11111111011111111111111110111011*
L0064 11111111101111111111111101111011*
L0096 11111111101111111111111110110111*
L0128 11111111011111111111111101110111*
L0256 11111111111111111111111111111111*
L0288 11110111111111111101101111111111*
L0320 11111011111111111101011111111111*
L0352 11111011111111111101101111111111*
L0384 11110111111111111110011111111111*
L0512 11111111111111111111111111111111*
L0544 01111111111111111011110111111111*
L0576 10111111111111110111110111111111*
L0608 10111111111111111011111011111111*
L0640 01111111111111110111110111111111*
L0768 11111111111111111111111111111111*
L0800 11011111111110111111111111011111*
L0832 11101111111101111111111111011111*
L0864 11101111111101111111111111101111*
L0896 11011111111101111111111111101111*
L1024 11111111111111111111111111111111*
L1056 11111111111111111111111101110111*
L1088 11111111011111111111111111110111*
L1120 11111111011111111111111101111111*
L1280 11111111111111111111111111111111*
L1312 11111111111111111110011111111111*
L1344 11110111111111111110111111111111*
L1376 11110111111111111111011111111111*
L1536 11111111111111111111111111111111*
L1568 11111111111111110111111011111111*
L1600 01111111111111111111111011111111*
L1632 01111111111111110111111111111111*
L1792 11111111111111111111111111111111*
L1824 11111111111110111111111111101111*
L1856 11011111111111111111111111101111*
L1888 11011111111101111111111111111111*
L1920 11111110111111101111111111111111*
L1952 11111111111011101111111111111111*
C90D0*
```

그림 8. PAL16L8의 JEDEC 화일(퓨즈 되지 않은 address line은 생략하였음)

Fig. 8. JEDEC file of PAL16L8.(Unfused address line is omit)

```
PAL16R4
_BCDR4.*
QP20*
QF2048*
G0*F0*
L0000 11111111111111111111111111111111*
L0032 11111111111111111111101111111111*
L0256 11111111111111111111111111111111*
L0288 11111011111111110111111111111111*
L0320 11110111111111111011111111111111*
L1280 11111011111111111111111111111111*
L1312 10111111111111111111111111111111*
L1536 11111111111111111111111111111111*
L1568 11111111111110111011111111111111*
L1600 11110111111110111111111111111111*
L1632 11111011111110111011111111111111*
L1792 11111111111111111111111111111111*
L1824 11111011011111111011111111111111*
L1856 11111011011110111111111111111111*
L1888 11110111101111111111111111111111*
L1920 11111111101101110111111111111111*
C3E3F*
```

그림 9. PAL16R4의 JEDEC 화일(퓨즈 되지 않은 address line은 생략하였음)

Fig. 9. JEDEC file of PAL16R4.(Unfused address line is omit)

4. 결과 비교

본 논문에서 적용한 디바이스 피팅 알고리듬을 이용한 결과로서 PALASM의 데이타 북(data book)에서 선정한 몇 가지 예제의 실행시간과 CHECKSUM 및 클럭 주기를 상용 툴인 PALASM과 비교하여 그 결과를 표 1에 나타내었다.

표 3에서 PLD 설계 툴의 실행시간이 PALASM에 비해 약간 우세함을 보였고, PALASM의 CHECKSUM과 클럭 주기를 고려한 디바이스 피팅 알고리듬을 추가한 PLD 설계 툴에서 생성된 CHECKSUM 값이 동일함을 입증하였다.

표 3. 개발된 PLD 툴과 PALASM과의 결과비교
Table 3. The result of comparing with a PLD design tool and PALASM.

예 제 명	사용 Device	PALASM		PLD Designer		
		실행시간	CHECKSUM	실행시간	CHECKSUM	Clock period
Wave form generator.src	PAL16R4	4.5sec	EAB2	4sec	EAB2	25ns
Hex Decoder.src	PAL16L8×3	4sec	EDEC	3.5sec	EDEC	-
4진 Counter.src	PAL16R4	3.5sec	47F0	3sec	47F0	25ns
STEP motor controller.src	PAL16R4 GAL16V8	4sec 4.5sec	3458 354A	4sec	3458 354A	25ns
Digital clock.src	PAL22V10×3	5sec	1A4D	4.5sec	1A4D	1sec
Traffic-light contoller.src	PAL16R4	4sec	197C	3.5sec	197C	25ns
※ Dynamic Memory state sequencer.src	PAL16R6 PAL16R4	9sec	4E25 194B	9sec	4E25 194B	25ns, 25+25=50ns
※ BCD Adder.src	PAL16L8, PAL16R4	3.5sec, 3.5sec	3E3F 90D0	3.1sec	3E3F 90D0	35+25=60ns

※. 디바이스 피팅 고려

Ⅲ. 결 론

본 논문에서는 다단 레벨(multi level)로 기술된 부울식의 경우 여러 개의 PLD로 회로를 구현하고, 순차 회로의 경우는 클럭의 주기를 결정할 수 있는 디바이스 피팅 기능을 개발하였다.

디바이스 피팅은 다단 레벨로 기술된 부울식이 선택된 PLD의 입출력 수보다 많을 경우 여러 개의 PLD를 이용하여 회로를 구현할 수 있도록, 논리 버텍스들 간의 연관 관계를 행렬(matrix)형태를 저장한 인접 행렬를 참조하여 CRITICAL 알고리듬에서 조합논리 회로의 경우 최장 경로를 구하고, 순차 회로의 경우 순환 경로를 구하여 이 경로의 모든 논리 버텍스를 저장하고 netlist를 생성하였다. 생성된 netlist를 이용하여 핀 맵 에디터에서 선택된 PLD에 피팅하였으며, 순차 논리 회로는 사용 PLD 및 구현된 PLD 단수에 따라 Clock_period 알고리듬을 이용하여 클럭 주기를 결정하였다.

디바이스 피팅을 수행한 후, FUSE MAP 및 JEDEC 화일과 CHECKSUM을 생성하여 PLD로 구현하였다.

실제로 본 논문에서 개발한 디바이스 피팅을 이용하여 설계한 BCD 가산기를 PLD writer(ALL-07)에 연결하여 구현한 결과 현재 상용 툴인 PALASM의 결과와 동일함을 입증하였으며, STEP MOTOR CONTROLLER와 4진 COUNTER, BCD ADDER, DIGITAL CLOCK, HEX DECODER, TRAFFIC-LIGHT CONTROLLER등의 예제도 PALASM의 결과와 동일함을 입증하였다.

앞으로의 과제는 다단 레벨로 기술된 부울식을 2단 레벨로 변경한후 자동으로 분할을 할 수 있는 분할 알고리듬과 TEST VECTOR를 자동 생성할 수 있는 ATPG 알고리듬 및 본 논문에서 개발한 디바이스 피팅 알고리듬을 대규모 PLD인 CPLD에 적용할 수 있는 알고리듬 연구 등이 절실히 요구된다.

참 고 문 헌

[1] Signetics, "*Programmable Logic Device Data Manual*"

[2] PARAG K. LALA, "*PLD DIGITAL SYSTEM DESIGN USING PROGRAMMABLE LOGIC DEVICE*", Prentice-Hall,

Inc. 1990.
[3] Giovanni De Micheli, *"SYNTHESIS AND OPTIMIZATION OF DIGITAL CIRCUIT"*, McGRAW-HILL, 1994
[4] Stephen D. Brown, Robert J. Francis, Jonathan Rose, Zvonko G. Vranesic, *"FIELD-PROGRAMMABLE GATE ARRAYS"*, Kluwer Academic Publishers, 1992
[5] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, *"INTRODUCTION TO ALGORITHMS"*, 1992.
[6] 김희석, 변상준, 하재희, 정홍구, "PAL 합성을 위한 JEDEC 화일 생성알고리듬에 관한 연구" 대한전자공학회 CAD, 전자 계산, 반도체·재료및 부품 연구회 합동 학술발표 논문집, pp.91-94, (1992)
[7] 김희석, 변상준, "논리 최소화를 위한 simplify algorithm 개선에 관한 연구", 대한전자공학회 반도체·재료및 부품 CAD 연구회 합동 학술 발표 논문집, pp.185-188, (1989)
[8] 정홍구, 김재진, 김성무, 변상준, 김희석, "STATE GRAPHIC EDITOR 개발에 관한 연구", 대한전자공학회 전자계산, 반도체,재료 및 부품, CAD 및 VLSI설계 합동 학술발표논문집, pp. (1993)
[9] 김재진, 김성무, 변상준, 김희석, "개선된 PLD 설계용 상태 Graphic Editor에 관한 연구", 대한전자공학회, 하계종합학술대회 논문집, pp. 634-637 (1994)
[10] 김희석, 원충상, "PLD 설계용 툴 개발에 관한 연구", 한국정보처리응용학회 논문집, 제1권, 제3호, pp 391-397 (1994)
[11] 김재진, 김성무, 조남경, 변상준, 원충상, 김희석, "PLD partition을 고려한 PLD 설계용 툴 개발", 대한전자공학회 추계종합학술대회 논문집, pp.1415-1418, (1994)

저 자 소 개

元 忠 常(正會員)

1973년 한양대학교 전자공학과 졸업(학사). 1983년 8월 한양대학교 대학원 졸업(공학석사). 1993년 청주대학교 전자공학과 박사과정 수료. 1981년 충주공업전문대학 전임강사. 1983년 ~ 현재 충주산업대학교 컴퓨터공학과 부교수. 관심분야는 IC 설계환경 구축, VLSI 설계자동화 등임

金 熙 碩(正會員)

1977년 한양대학교 전자공학과 졸업(학사). 1980년 한양대학교 대학원 전자공학과 졸업(공학석사). 1985년 한양대학교 대학원 전자공학과 졸업(공학박사). 1981년 ~ 83년 청주대학교 전자공학과 전임강사 1993년 ~ 87년 청주대학교 전자공학과 조교수. 1987년 ~ 88년 미국 Colorado Univ. 전자공학과 VLSI 설계실 연구원. 1987년 ~ 92년 청주대학교 전자공학과 부교수. 1993년 ~ 현재 청주대학교 전자공학과 교수. 관심분야는 IC 설계환경 구축, VLSI 설계자동화 등임