論文97-34C-2-2

BDD를 이용한 complex term의 최소화와 Cellular Architecture FPGA에의 응용

(Minimization of Complex Terms using BDD and it's Application to Cellular Architecture FPGA)

金 美 英 *, 李 貴 相 *

(Mi-Young Kim and Guee-Sang Lee)

요 약

본 논문에서는 cellular architecture FPGA에 대한 효과적인 합성 방법을 제시한다. 이 방법은 Cellular Architecture FPGA구조에 직접 매핑될 수 있는 complex term이라 불리는 논리표현을 생성하기위해 기존의 합성 방법처럼 SOP나 ESOP최소화 도구를 이용하지 않고 BDD(Binary Decision Diagram)로 변환된 논리함수를 연산에 사용한다. 이 과정에서, 각 노드의 1-간선과 0-간선, 그리고 이들 두 간선을 exclusive-OR하여 생성된 새로운 간선의 각각의 비용(complex term의 수)을 측정하여 루트 노드부터 터미널 노드까지 탐색하며 논문에서 제시하는 알고리즘에따라 complex term들을 생성해내는 것이다. 여러 벤치마크 회로에 실험한 결과 기존의 방법[7]에 비해 complex term의 수가 줄었으며 이는 FPGA 합성시 면적을 줄일 수 있는 좋은 결과를 주었다.

Abstract

An efficient synthesis method of cellular architecture FPGA is proposed in this paper. To generate a logical representation called complex term which is to be directly mapped onto the cellular architecture FPGA, an SOP or ESOP minimization tool was used in previous methods. Instead, we use a logic function transformed into BDD(Binary Decision Diagram) in the actual generation of the complex terms. In this process it estimates the cost(i.e. the number of complex terms) for three branches, 0-branch and 1-branch of each node of BDD, and another new branch obtained by exclusive-ORing those 0 and 1-branches. This process is continued over the whole BDD to do such computation, and we observed that the number of complex terms has been reduced compared to the previous results[7].

I. 서 론

최근 들어 프로토타입의 빠른 구현을 위하여 프로그램 가능한 논리소자들이 기존의 ASIC 디자인을 대신하여 자주 사용되고 있다. 그 중에서도 특히 FPGA (Field Programmable Gate Array)에 대한 연구가 집중적으로 이루어져 왔다. 기존의 FPGA 형태로는 LUT(Look-up Table) based FPGA와 Multiplexor-based FPGA를 들 수 있다. 본 논문에서 다루고자 하는 분야는 이러한 FPGA들 보다 좀더 새로운 형태인 Cellular Architecture FPGA에 대한 합성 도구의 개발이다. Cellular Architecture FPGA는 대칭구조를 갖는 소규모의 논리 블럭들을 주로 local connection으로 연결하는 것으로서 논리 블럭들은 표준셀과 같은 형태를 갖고 이들은 매우 제한된 입출력 선들을 갖고 있다.

* 正會員, 全南大學校 電算統計學科

(Dept. of Computer Science & Statistics, Chonnam Univ.)

接受日字:1996年4月25日, 수정완료일:1997年2月5日

현재 Concurrent Logic Inc., Toshiba, Plessey, Algotronix, Motorola등 많은 업체에서 이러한 칩들을 제공하고 있지만 그 합성 방법은 본격적으로 연구되지 못한 상태이다. 이러한 Cellular Architecture FPGA는 "Fine Grain", "Cellular" 또는 "Sea of Gates" FPGA 등으로 불리며 이들에 대해서는 기존의 LUT based FPGA[1-3] 또는 기존의 논리 합성 방법(예를 들면 MisII[10] 등의 방법)은 효과적으로 적용될 수 없으며, 전혀 새로운 합성 방법이 개발되어야 한다.

일반적인 Cellular Architure type FPGA는 논리 블럭들을 배열로 나열하고 각각의 논리 블럭들은 입출력선에 의해 s개의 이웃들과 연결되어 있다. 이들은 각각 n개의 입력을 이웃으로부터 받아들이며, m개의 출력을 이웃으로 보낼 수 있다. 또한 k개의 입력을 수평 또는 수직의 로칼 버스로부터 받아들이며 마찬가지로 l개의 출력이 이들 로칼 버스로 연결될 수 있다. 논리 블럭들은 임의의 논리함수를 구현하며 이러한 구조에서는 각 셀(논리 블럭)이 작을수록 전체의 면적이용률이 좋아지므로 일반적으로 이러한 논리 블럭들은 단순한 게이트로 제한된다(xor게이트 포함). 본 논문에서는 이러한 배경과 함께 합성 방법을 간단히 설명하기 위해 우선 각 논리 블럭 들이 다음과 같이 매우 간단한 형태를 갖는다고 가정한다.(AT6000시리즈에서 각 셀은 3개의 입력과 2개의 출력이 가능하도록 되어 있다. 또한 버스로부터는 오직 하나의 입력만 가져올 수 있고 로칼 버스의 갯수는 4개로 한정되어 있다. 그러나 다음의 가정에 의한 논리합성은 실제 구현되는 경우 folding기법 등으로 매우 쉽게 AT6000시리즈로 매핑될 수 있다.)

(1) 각 셀은 2-입력 1-출력의 기본 논리 게이트라고 가정한다. 이들은 AND, OR, XOR 게이트들을 말하며, 셀의 입출력은 프로그램가능한 인버터가 사용될 수 있다. 따라서 각 셀은 임의의 2-입력 함수를 구현할 수 있다.

(2) 수평 버스들은 그 행에 있는 모든 셀에 연결되어 있으며 수직 버스들은 그 열에 있는 모든 셀에 연결되어 있다.

(3) 각 셀은 이웃한 4개의 셀에 연결된다. 구석이나 변의 셀들은 이웃한 2개 또는 3개의 셀과 연결된다.

(4) 각 셀은 이웃한 4개의 셀중 2개로부터 입력을 받아들이거나, 하나는 이웃한 셀에서 그리고 다른 하나는 버스로부터 받아들일 수 있다.

(5) 각 셀은 임의의 이웃한 셀이나 버스로 출력을 내보낼 수 있다. 단, 입출력을 같은 이웃한 셀이나 같은 버스에 동시에 연결할 수 없다.

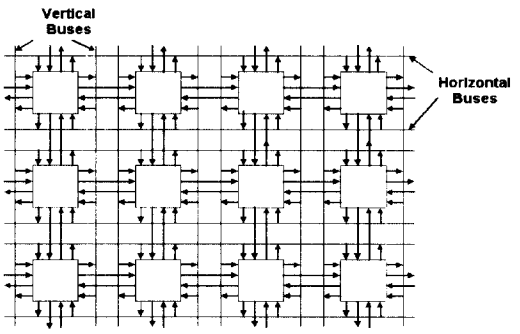

그림 1. Cellular Architecture FPGA의 기본구조
Fig. 1. Basic architecture of Cellular Architecture FPGA.

본 논문에서 채택한 구조는 그림 1.에 나타나 있다. 이러한 구조에서의 FPGA 매핑에 대한 연구는 주로 Portland Univ. 연구팀에 의해 연구되었으며, 이제 본격적인 연구의 시작 단계에 있다. 다음에서는 먼저 기존의 연구를 간단히 알아보고 본 논문에서 채택하고자 하는 방법을 소개한다.

II. 기존의 최소화된 ESOP표현을 이용한 방법

Perkowski[7]의 방법은 complex term의 개념에 입각하여 FPGA 합성방법을 설명하고 있다. 여기에서도 본 논문과 같은 셀 구조를 가정하고 있다. 먼저 complex term의 개념과 필요성에 대해 살펴보고 주어진 함수가 cellular 구조에 어떻게 구현되는지를 설명하겠다.

【정 의】 complex term이란 리터럴들이 임의의 연산자에 연결되어 있는 스트링을 말하며, 이 때 반드시 하나의 입력 변수는 단 한번만 나타나며, 연산자는 AND, OR, XOR 또는 이들의 complement이다. 이러한 연산자들에 대한 우선 순위는 없으며 좌로부터 우로 진행된다. 다음의 예제1과 2는 complex term들의 예와 complex term이 될 수 없는 경우를 보여준다.

【예제 1】 다음은 complex term들의 예이다.

$(ab')+c$

(a+b)c
(a⊕b)+c′
(cb′+a)⊕d

【예제 2】 다음은 complex term이 아닌 경우이다. (ab + b′) c 경우는 b가 두번 나타났기 때문이고, a + bc′ + d 는 operation이 좌에서 우로 의 순서로 전개되지 않았기 때문이다. 그러나 만일 변수의 순서를 b,c,a,d로 바꾸면 (bc′)+a+d 는 complex term 이 될 수 있다.

위의 정의에 의하면 임의의 곱항은 complex term이다. 그러나 complex term은 예1,2에서 본바 와 같이 단순한 곱항외에 더 많은 부울 함수를 표현할 수 있다. 위의 complex term 들은 cellular 구조에서 쉽게 구현될 수 있다. 반면 complex term이 아닌 경우에는 routing wire가 필요하게 되며, 이는 cellular구조에 적용될 때 버스나 routing을 위한 셀을 필요로 한다. 만일 함수가 복잡해지면 routing wire의 수도 늘어나게 되고 ASIC 디자인시 routing 복잡도도 커지게 된다. 따라서 FPGA구현시 routing 때문에 많은 수의 셀이 필요하게 된다. 그림 2.는 f=(a+b)(c+d)를 구현하기 위해 routing wire가 필요함을 보여준다. 반면에 그림 3.은 f = ((a+b)c+d)e로서 complex term이므로 routing wire가 필요치 않으므로 ASIC과 FPGA구현을 쉽게 한다.

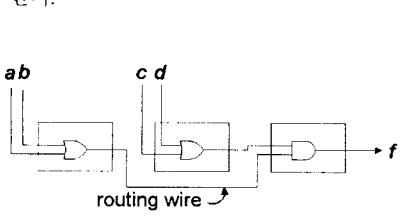

그림 2. f = (a+b)(c+d), routing wire를 필요로 함
Fig. 2. f = (a+b)(c+d), routing wire is necessary.

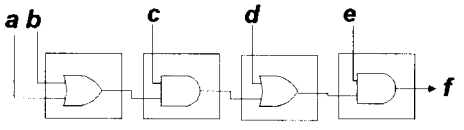

그림 3. f=(((a+b)c)+d)e, routing wire가 필요치 않음.
Fig. 3. f=(((a+b)c)+d)e, routing wire is not necessary.

다음은 주어진 함수가 어떻게 cellular 구조에 구현되는지를 설명하고자 한다. 예를 들어 다음과 같은 3-출력 함수 F를 고려해 보자

$f_0 = ac \oplus a'bd \oplus bc'd$
$f_1 = b \oplus d$
$f_2 = c \oplus bd \oplus a$

f_0의 최소화된 complex term들을 얻기 위하여 그림 4.와 같이 큐브들을 재구성한다. 즉, A=ac, B=a′bd, C=bc′d를 A′=ac, B′=bcd, C′=abd로 재구성하여 다음단계로 진행한다. 이는 보다 쉽게 complex term을 구성하기 위하여 함수의 큐브 형태들을 조정하는 것이다. 따라서 f_0는 다음과 같이 전개되며 또한 각각의 함수에 대해 다음과 같이 complex term으로 구현될 수 있음을 알 수 있다.

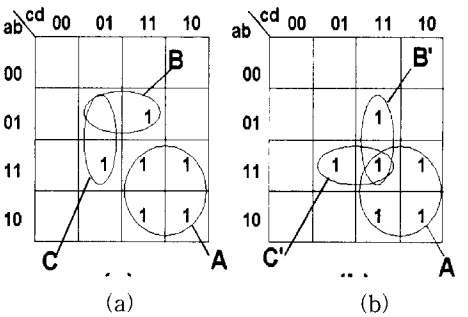

그림 4. f_0의 (a)원래의 cube들, (b)재구성된 결과
Fig. 4. (a)the original cubes of f_0, (b)reshaped cubes of f_0.

$f_0 = ac \oplus a'bd \oplus bc'd$
$= ac \oplus bcd \oplus abd$
$= ac + bcd + abd$
$= (bd + a)c + bda$ 두개의 complex term
$f_1 = b \oplus d$ 한개의 complex term
$f_2 = bd \oplus a \oplus c$ 한개의 complex term

이들은 그림 5.와 같이 구현될 수 있다.

Perkowski[7]가 제안한 cellular구조에서의 기본적인 합성방법은 주어진 최소화된 SOP 또는 ESOP ([7]에서는 ESOP 만 적용함)를 입력으로 받아들여 이들을 조합하여 complex term들로 재구성한다. 이 때, complex term들은 대개의 경우 주어진 SOP 또는 ESOP의 항 수보다 작아진다. 기본적으로 이러한 complex term들은 직접적으로 cellular구조에 적용될 수 있으며 여기에 부수적으로 folding등의 개념을 사용

하여 주어진 셀들이 가능한 한 많이 이용되도록 한다.

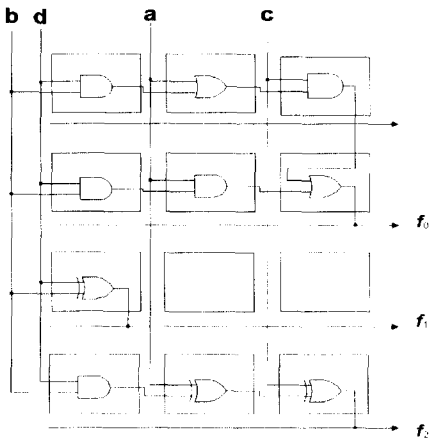

그림 5. 함수 F=(f_0,f1,f_2)의 cellular FPGA구현
Fig. 5. cellular FPGA realization of function F=(f_0,f_1,f_2).

위의 방법은 다음과 같은 단점들을 갖고 있다.

(1) 위의 방법을 적용하기 위해서는 최소화된 SOP 또는 ESOP 표현이 필요하며 이는 그러한 최소화도구를 요구하게 된다. SOP에 대해서는 일반적으로 유용한 ESPRESSO가 존재하나 ESOP의 최소화는 아직은 완전히 정착되지 못한 상태이며 이는 경우에 따라 매우 높은 비용을 요구할 수도 있다. 또한 사실상 SOP 형태만을 가지고 [7]의 방법을 적용할 수는 없으며 (SOP 표현을 사용하면 complex term의 발생이 매우 제한적이다.), 이 점이 [7]에서 ESOP 표현만을 사용한 이유이기도하다. 따라서 필연적으로 ESOP 최소화 도구를 요구하게 되는데 이는 주어진 CAD 환경에 따라 이 방법을 적용하는데 결정적인 장애 요인이 될 수 있다.

(2) 위의 방법의 초기 표현인 SOP나 ESOP 표현은 함수에 내재된 complex term 표현을 왜곡시킬 수 있다. 즉, SOP나 ESOP 표현에 집착할 경우 반드시 필요한 complex term들의 최소화를 이룰 수 없다는 것이다.

따라서 SOP나 ESOP 표현에 구애되지 않은 함수의 표현으로부터 complex term들을 직접 구할 수 있는 방법이 요구된다.

본 논문에서는 최소화 도구를 거치지 않은 함수의 임의의 표현으로부터 최소화된 complex term들을 발생시키는 방법을 제시하며 추후로 이를 대표적인 논리 합성 도구인 SIS에 추가 구현하고자 한다.

Ⅲ. Complex term의 생성 방법

본 논문에서 사용하는 방법은 Davio expansions을 이용하여 만들어진 항이 complex term이라는 사실에 근거하는 방법으로 Davio expansions중에서 가장 최소의 complex term수를 나타내는 decomposition을 선택하게 된다. 기본적인 함수 표현으로는 BDD (Binary Decision Diagram)를 사용한다. 이것은 BDD가 함수를 간결하게 표현하고 여러 가지 연산을 효과적으로 수행할 수 있기 때문이다.

(1) 주어진 함수는 최근에 거의 표준적으로 사용되고 있는 BDD 형태로 변환한다. BDD는 임의의 논리함수를 쉽고 효과적으로 실행하므로 이 후의 연산에 가장 유리한 함수 표현이다. 또한 여기서는 어떤(SOP나 ESOP) 최소화도구도 필요로 하지 않는다.

(2) complex term 생성을 위한 입력변수들의 순서를 정한다. 여기서는 주어진 BDD의 변수 순서로 한다. 차후의 연구로는 (1)단계를 시행하기 전에 최선의 변수 순서를 계산할 수 있으나 이는 본 논문의 범위를 벗어나므로 여기서는 단순히 BDD에서 사용된 임의의 변수 순서에 따른다.

(3) 결정된 변수 순서에 따른다.(현재의 입력변수를 v 이고 현재의 BDD 노드가 구현하는 함수를 f 라 하자)

BDD의 노드의 간선 f_l, f_r 와 f_x을 아래와 같이 가정한다.

f_l = f(v=0) 현 BDD 노드의 왼쪽 간선

f_r = f(v=1) 현 BDD 노드의 오른쪽 간선

f_x = $f_l \oplus f_r$ 현 BDD 노드의 두 간선을 exclusive-OR

함수 f를 구현하는 방법은 다음의 3가지가 있을 수 있다.

$$f(v) = v \cdot f(v=1) + v' \cdot f(v=0) \quad (1)$$

$$= v \cdot g \oplus f(v=0) \quad (2)$$

$$= f(v=1) \oplus v' \cdot g \quad (3)$$

만일 f(v=1)=1이라 가정하면 (1)식은

$$= v + v' \cdot f(v=0)$$
$$= v + f(v=0) \quad (1)'$$

이러한 과정을 구현하기 위해 주어진 함수의 BDD의 각 노드에서 f_x를 갖는다고 생각하고 각 노드에 대해 생성해 낼 수 있는 최소수의 complex term을 계산한다. 이에 대한 알고리즘은 그림6.과 같다.

```
cost(d) /*BDD의 노드 d가 필요로하는 complex
term의 수를 측정함 */{
    if (d가 방문되었음) return d -> cost;
    if (d가 path의 마지막 변수) return 1;
    if (d가 상수) return 0;
    l = cost(d->else);
    r = cost(d->then);
    x = cost(d->xor); /* d->xor = d->else ⊕
        d->then */
    return d->cost = l + r + x - max(l,r,x);
}
```

그림 6. complex term의 수를 측정하는 알고리즘
Fig. 6. Algorithm for the estimation of the number of the complex terms.

이렇게 노드의 비용이 결정되면 BDD에서 최소수의 complex term을 생성해낸다(그림 6.)

```
complex_gen(d)
{
    if (d가 터미널 노드) {
            근노드부터 노드 d 까지의 path 생성;
            return;
    }

    l = cost(d->else);
    r = cost(d->then);
    x = cost(d->xor);
    l, r, x중에서 작은 두 값을 선택한다.
    if (l 과 r이 선택) { /* Shannon decomposi-
        tion을 사용 */
        arr = arr+"v ·" , complex_gen(d->
        then); /* arr은 path를 저장 */
        arr = arr+"v′ ·" , complex_gen(d->
        else); /* v는 d의 변수 */
    }
    if (r 과 x가 선택) { /* positive Davio de-
        composition을 사용 */
        complex_gen(d->then);
        arr = arr+"v ·", complex_gen(d->xor);
    }
    if (l 과 x가 선택) { /* negative Davio de-
        composition을 사용*/
        complex_gen(d->else);
        arr = arr+"v′ ·", complex_gen(d->xor);
    }
    return;
}
```

그림 7. complex term을 생성해내는 알고리즘
Fig. 7. algorithm for the generation of complex terms.

위 방법을 다음과 같이 예를 통하여 살펴보자. 주어진 함수 f에 대하여 그림8과 같이 f_l, f_r, f_x를 BDD를 사용하여 표현하고 그림6의 알고리즘을 이용하여 비용을 구하면 그림9과 같은 결과를 얻게되는 것이다. 이를 바탕으로 그림7의 알고리즘을 complex term의수를 최소로하는 decomposition을 선택하면서 complex term을 만들어낸다. 그림10에서는 f_l과 f_r의 선택에따라 (1)′ 과 (1)식이 선택되었을 때의 확장 예이다.

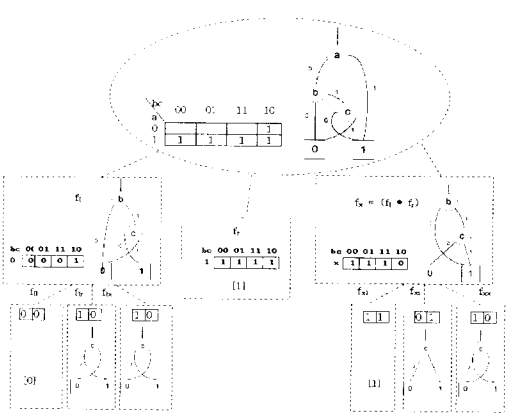

그림 8. 함수 f의 분할과정
Fig. 8. decomposition of function f.

$f = a' \cdot fl + a \cdot fr$

$f(a=1) = 1$이므로 식(1)′에 의해 계산

$= fl + a(1)$

그런데 $fl = b' fl + bfr$

$= b'(0) + b(c') = bc'$ 이므로

$= bc' + a$

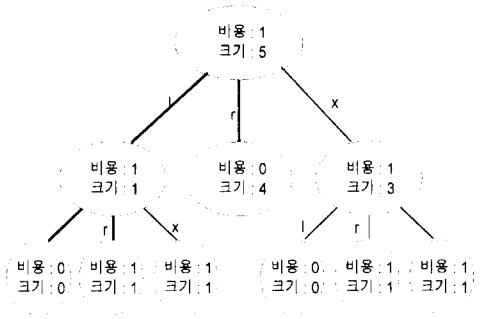

그림 9. 비용함수 계산
Fig. 9. computation of cost function.

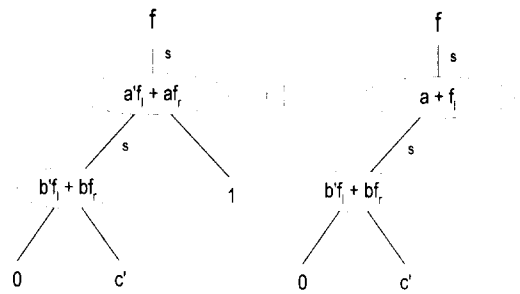

그림 10. complex term의 생성
Fig. 10. generation of complex term.

위 예는 결국 f = bc'+ a, 하나의 complex term으로 구성된 결과를 얻을 수 있다. 이때 실제 FPGA로 구현할 때의 면적은 complex term의수 * (입력의 수 + 출력의 수)가 되고 각 회로의 입출력의 수는 정해져있으므로 결국 complex term의 수가 면적을 결정하게된다. 또한 기존결과와 비교할 수는 없으나 delay측면에서 고찰해 보면 delay는 통과하는 셀의 수로 측정할 수 있다. 즉, delay = 통과하는 셀의 수 = 입력의 수 + complex terms의 수가 된다. 다음절에서는 이러한 실험결과를 제시하고 있다.

Ⅳ. 실험 결과

본 논문에서 제안한 비용함수를 사용하여 IBM PC(90MHZ)에서 C언어로 구현하였다. 수정된 BDD의 구성에는 Carnegie Mellon University에서 개발한 BDD 패키지를 사용하였으며 [7]에서 실행된 Benchmark 회로와 결과를 비교하였다. 표 1의 결과에서와 같이 면적에서 약 10%정도의 향상을 보였으나 minimized cube form이 BDD에 반영되지 않는 종류의 벤치마크 회로(clip,clog8등)에 대해서는 개선된 결과를 볼 수 없었다.

표 1. complex term수 의 비교
Table 1. Comparison of the number of complex terms.

Circuit	PI	PO	# Complex terms			
			[7]의 결과	위의 방법	실행시간(초)	delay(cell의 수)
5xp1	7	10	33	23	0.18	30
card4	8	5	30	20	0.24	28
clip	9	5	63	86	1.45	95
clog8	8	8	84	98	1.56	106
cu	14	11	15	19	0.15	33
f51m	8	7	30	21	0.20	29
mlp3	6	6	17	17	0.13	23
rd53	5	3	13	9	0.08	14
rd73	7	3	36	19	0.27	26
sao2	10	4	26	31	0.45	41
t481	16	1	18	8	1.39	24
vg2	25	8	179	138	2.31	163
SUM			544	489		
z4ml	7	4	-	15	0.11	22
mul3	6	6	-	19	0.32	25
mul4	8	8	-	64	2.31	72
add4	8	8	-	30	1.52	38
9sym	9	1	-	46	0.43	55

(: not available)

Ⅴ. 결 론

본 논문에서는 Cellular Architecture FPGA를 자동적으로 합성하기 위한 효과적인 방법을 제시하였다. 이 방법은 CA-type FPGA에 직접 매핑이 가능한 complex term을 생성하는데 최소화된 ESOP최소화 도구를 사용하는 방법보다 더 좋은 결과를 보였다. 이는 SOP최소화 도구인 ESPRESSO와 같은 적립된 ESOP최소화 도구의 부재 상황에서 ESOP최소화를 거치지않고 BDD를 이용하여 complex term을 생성하여 약 10.1%의 향상된 결과를 얻은 것이다. 또한 일반적인 Sea-of-gates array의 설계에도 응용될 수 있으리라 보인다. Davio expansions를 이용하므로 KFDD (Kronecker Functional Decision Diagram) 최소화와 매우 유사한 형태를 갖는데 이러한 사실을 이용하

면 KFDD의 최소화도 구현할 수 있을 것으로 보인다. 즉, KFDD에서는 전체 DAG의 노드 수를 최소화하고자 하는데 비하여 본 논문의 방법은 루트에서 터미널까지의 모든 path의 수를 최소화하는 것이 다르다. 따라서 본 논문의 방법이 KFDD의 최소화에 직접 또는 간접적으로 응용될 수 있으리라 보인다.

앞으로의 연구는 더 나은 결과를 얻기 위해 최소의 complex term들을 결정하는 모든 경우를 고려해야 하겠으며, output folding을 고려한다면 complex term의 수를 더욱 줄일 수 있겠고, SIS와 같은 실제적인 논리 합성 도구에 추가 구현하고자 한다.

참 고 문 헌

[1] R.J. Francis, "A Tutorial on Logic Synthesis for Lookup-Table Based FPGAs," *ICCAD '92*, pp 40-47,1992.

[2] R.P. Jacobi and A.M. Trullemans, "A New Logic Minimization Methods for Multi-lpexor Based FPGA Syntehsis," *EDAC-93*, pp. 312-317, 1993.

[3] I. Schafer and M.A. Perkowski, "Synthesis of Multi-level Multiplexer Circuits for Incompletely Specified Multioutput Boolean Functions with Mapping to Muliplexer Based FPGAs," *IEEE Tr. on CAD*, vol. 12, no. 11, pp. 1655-1664, nov. 1993.

[4] I. Schafer, M.A. Perkowski and H. Wu, "Multilevel Logic Synthesis for Cellular FPGAs Based on Orthogonal Expansion," *IFIP Workshop on Applications of the Reed Muller Expansion in Circuit Design, Hamburg, Germany*, pp. 42-51, sep. 1993.

[5] A. Sarabi, P.F.Ho, K. Iravani, W.R. Daasch, and M.A. Perkowski, "Minimal Multi-Level Realization of Switching Functions Based on Kronecker Functional Decision Diagrams," *Proc. of IWLS '93*, Tahoe City, CA, May 1993.

[6] M.A. Perkowski, "A Fundamental Theorem on Exor Circuits," *IFIP Workshop on Applications on Reed Muller Expansions in Circuit Design, Hamburg, Germany*, pp 52-60, Sep 1993.

[7] A. Sarabi, Ning Song, M. Chrzanowska-Jeske and M.A. Perkowski, "A Comprehensive Approach to Logic Synthesis and Physical Design for Two-Dimensional Logic Arrays," *Design Automation Conference*, pp. 321-326, 1994.

[8] R. Drechsler, A. Sarabi, M. Theobald, B. Becker and M. A. Perkowski, "Efficient Representation and Manipulation of Switching Funuctions Based on Ordered Kronecker Function Decision Diagrams," *DAC* 1994.

[9] N. Song and M.A. Perkowski, "A New Design Methodology for Two-dimensional Logic Arrays," *Proc. of IWLS'93*, Tahoe City, May 1993.

[10] R.K. Brayton et al., "Logic Minimization Algorithms for VLSI synthesis," *Kluwer Academic Publishers.*

[11] R.K. Brayton et al., "MIS: A Multi-level Logic Optimization System," *IEEE Transactions on CAD*, pp. 1062-1081, Nov. 1987.

저 자 소 개

金 美 英(正會員)

1961년 9월 24일생. 1983년 2월 전남대학교 계산통계학과 졸업(이학사). 1985년 9월 이화여자대학교 수학과 전산 전공 졸업(이학석사). 1986년 3월 ~ 현재 목포전문대학 전자계산과 부교수. 1994년 3월 ~ 현재 전남대학교 전산학과 박사과정 수료. 주관심분야는 VLSI 설계자동화, 논리합성, 멀티미디어

李 貴 相(正會員) 第 32卷 A編 第 6號 參照

주관심분야는 설계자동화, 테스팅, 논리합성, 멀티미디어 등임