Complex term을 이용한 OPKFDD의 입력변수 순서 방법

(A Variable Ordering Method for OPKFDDs using Complex Terms)

정 미 경 † 김 미 영 †† 이 귀 상 †††

(Mi Gyoung Jung)(Mi Young kim)(Guee Sang Lee)

요 약 OPKFDD(Ordered Pseudo-Kronecker Functional Decision Diagram)는 각 노드에서 다양한 decomposition을 취할 수 있는 Ordered-DD(Decision Diagram)의 한 종류이다. OBDD(Ordered Binary Decision Diagram)에서 각 노드는 Shannon decomposition만을 이용하는 반면, OPKFDD는 각 노드마다 Shannon, positive Davio, negative Davio decomposition 중의 하나를 사용하도록 하며 많은 경우 매우 적은 수의 노드로 함수를 표현할 수 있다. 그러나 각 노드마다 각기 다른 확장 방법을 선택할 수 있는 특징 때문에 입력 노드에 대한 확장 방법과 입력 변수 순서의 결정에 의해서 OPKFDD의 크기가 좌우되며 이에 대한 최적의 해를 구하는 것은 매우 어려운 문제로 알려져 있다. 본 논문에서는 DD 크기의 기준을 노드 수로 하여 기존의 BDD(Binary Decision Diagram) 자료구조에서 OPKFDD를 효율적으로 유도해내는 방법을 제시하고 complex term을 이용하여 이를 최소화하는 알고리즘을 제시한다. 그리고 입력변수 순서 결정을 위하여 다출력함수의 경우 함수간의 포함관계를 고려한 그룹-sifting과 각 노드의 확장 방법을 제안하고 실험 결과를 제시한다.

Abstract OPKFDD(Ordered Pseudo-Kronecker Functional Decision Diagrams) is one of ordered-DDs(Decision Diagrams) in which each node can take one of the three decomposition types, Shannon, positive Davio and negative Davio in contrast to the OBDD(Ordered Binary Decision Diagram) which uses only Shannon decomposition in each node. In many cases it has been shown that OPKFDD can represent functions in more compact form than any other DDs, e.g. OBDD(Ordered Binary Decision Diagram). However, this leads to the very difficulty of getting an optimal solution in the minimization of OPKFDD. Since an appropriate decomposition type has to be chosen for each node, the size of the representation is decided by the selection of a decomposition type as well as a variable ordering of the diagram. We propose a method for generating OPKFDD efficiently from BDD(Binary Decision Diagram) of a given function with a group-sifting algorithm for the decision of a good variable ordering of multi-output functions using complex terms. Experimental results demonstrate the performance of the algorithm.

1. 서 론

최근에 불리안 함수로 대변되는 논리회로의 표현을 위해 그래프를 이용한 표현 방법인 DDs(Decision Diagrams)가 제시되어 논리합성, 테스트 그리고 회로검증(verification)과 같은 여러 가지 설계자동화 문제에 활발히 적용되어 왔다. 여러 가지 종류의 DD 중에서 실제로 가장 많이 사용하고 있는 것은 OBDDs(Ordered Binary Decision Diagrams)이다[1,2,3]. 현재 OBDDs의 응용범위는 더욱 넓어져가고 있으며 이는 불리안 함수 외에도 여러 가지 이산함수(discrete function) 즉, 다치함수, 큐브 집합(cube set), 산술함수(arithmetic function)등의 경우에도 적용되며 이와 같은 논리함수의 그래프표현은 전산, 전자분야 뿐 아니라 다른 많은 분야에서도 매우 유용하게 쓰이고 있다. 또한 장기적으로 이

† 정 회 원 : 전남대학교 전산학과
mgjung@chonnam.chonnam.ac.kr
†† 정 회 원 : 담양대학 전산·정보통신공학부 교수
kimmee@damyang.damyang.ac.kr
††† 종신회원 : 전남대학교 전산학과 정보통신연구소 교수
gslee@chonnam.chonnam.ac.kr
논문접수 : 1999년 6월 5일
심사완료 : 2000년 7월 18일

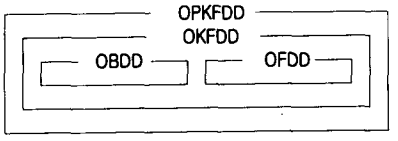


그림 1 DD들간의 관계

러한 그래프 표현은 그 특성인 정규적(canonic) 표현임과 동시에 사용의 편리함 등으로 인하여 더욱 그 사용이 늘어날 전망이다. DD의 사용이 활발해짐에 따라 여러 가지 형태의 DD가 제시되었으며 그 중에서도 두드러진 것은 *OFDD(Ordered Functional Decision* Diagram)이며[4,5] 이는 같은 변수를 나타내는, 즉 같은 level의 노드에서 positive Davio와 negative Davio 중의 하나를 사용한다. OFDD로부터 OKFDD (Ordered Kronecker Functional Decision Diagram)이 제안되었으며[6] 이는 같은 레벨의 노드에서 세 가지 확장(decomposition)인 Shannon, positive Davio, negaive Davio 중의 하나를 사용한다. 또한 이를 더욱 발전시킨 형태인 OPKFDD[6]는 각 노드마다 위의 세 확장 중에서 임의의 하나를 선택하도록 한 것이다. OPKFDD는 각 변수에 대해서 적절한 확장 방법을 선택함으로써 OBDD와 OFDD보다 주어진 함수를 훨씬 적은 노드 수를 가지고 표현할 수 있다[5]. 이러한 DD들의 상관관계는 (그림 1)과 같다[6].

각 노드의 선택의 폭이 넓어짐에 따라 표현공간을 최소화의 가능성은 더욱 높아지나[7,8] OPKFDD의 경우 n개의 입력변수에 대해서 3^{2^n-1}개의 서로 다른 확장을 할 수 있으므로 이를 최소화하는 연구는 상대적으로 더 복잡성을 띄게 된다[6]. OPKFDD를 최소화하기 위해서는 두 가지 관점에서의 연구가 필요하다. 첫째, 입력 변수가 n개 일 때, $n!$가지의 입력 변수 순서 중에서 DD의 크기를 작게 하는 입력변수 순서를 결정하는 것이다. 둘째, OPKFDD의 경우 Shannon 확장, Positive Davio 확장 그리고 negative Davio 확장 중에서 각 노드에 대한 확장 방법을 선택하는 것이다.

OBDD에서의 입력 변수 순서를 결정하는 문제는 NP-complete 임이 증명되었으며[9] 따라서 이를 해결하기 위한 방법들은 여러 가지 휴리스틱(heuristic)에 의존하고 있는 실정이다[10,11]. 또한 OFDD와 OKFDD의 입력변수 순서를 결정하는 최적화에 대한 연구가[4,5,7,8] dynamic reordering[13]의 개념을 이용하여 이루어지고 있다. 이들 연구에서 제안한 방법은 sifting과 인접변수 교환 방법을 이용하며 OBDD와 비교해서 더 간소화된 함수표현이 가능하나 입력이 많은 함수에 대해서는 실질적인 시간 내에 최적의 순서를 찾는다는 것은 매우 어려운 문제로 알려져 있다. 또한 이 문제를 해결하는 한 방법으로서 유전자(genetic) 알고리즘을 적용한 연구[12]가 이루어지고 있다. 여기서는 자연의 유전학(natural genetics)과 자연선택(natural selection)의 원리에 바탕을 둔 통계적(stochastic) 최적해 탐색 방법을 이용한다. 유전자 알고리즘은 기존의 최적해 탐색이 국부탐색이었는데 비하여 여러 해를 동시에 탐색하는 전역 탐색(global search)을 함으로써 전역적인 최적해(*global optimal* solution)를 찾을 확률이 기존의 최적화 탐색에 비해 큰 것이 특징이다. 이와 같은 원리를 입력변수 순서와 각 확장 방법을 결정하는 방법에 적용하고 있다. 이들 방법은 최적의 해에 근사한 해를 구하지만 최적의 해를 구하기 위한 파라미터의 값을 결정하기가 매우 어렵고, 상당히 많은 연산 시간이 소요되는 문제점을 안고 있다. 그러나 OPKFDD에서는 OKFDD와는 다르게 각 노드에서의 확장방법을 선택하는 문제가 새로이 고려되어야 하며 이러한 특성 때문에 입력변수의 순서를 결정하는 것은 매우 어려운 문제로 알려져 있다[16].

본 논문은 sifting 방법을 기반으로 하여 다중 출력 함수의 경우 각 부분 함수의 관계를 고려한 입력변수의 순서를 결정하는 방법을 제시한다. 각 노드에서의 확장방법의 선택을 위하여 *complex term*의 개념[14]를 이용한다. Complex term은 원래 PSDKRO (Pseudo-Kronecker Expansions)[6]의 최소화에 이용하는 방법으로서 본 논문에서는 PSDKRO와 OPKFDD의 유사성을 이용하여 확장방법을 선택하는 비용함수로 이용한다. PSDKRO는 OPKFDD의 각 경로를 하나씩 나열함으로써 얻을 수 있다[6]. Complex term에 의한 비용함수는 노드의 수에 대하여 선형시간에 계산할 수 있으므로 이를 이용하는 경우 빠른 시간 안에 확장방법을 선택할 수 있다는 장점이 있다. 마지막으로 이 논문에서 제안한 방법과 CUDD(Colorado University Decision Diagram) Package[15]에서 제공하는 기존의 reordering 방법[13,17,18,19]를 비교한 실험결과를 제시한다.

본 논문의 구성은 다음과 같다. 2장에서는 *OPKFDD*의 정의와 *complex term*의 개념을 제시하고, 3장에서는 OPKFDD에서의 확장 방법과 입력변수 순서 결정 방법을 제안한다. 그리고 4장에서는 3장에

서 제안한 알고리즘의 실험결과를 제시하며, 마지막으로 결론과 향후 연구 방향을 언급한다.

2. 관련연구

2.1 OPKFDDs

DD, OFDD, OKFDD 그리고 OPKFDD의 정의는 다음과 같다[6]. OFDD, OKFDD 그리고 OPKFDD는 확장된 DD의 개념을 갖는다.

[정의 1] Decision Diagram

(1) DD는 입력변수 $X_n := \{x_1, x_2, \ldots, x_n\}$에 대해 방향성 비 순환 그래프(Directed Acyclic Graph) 형식을 갖는다.

(2) 두 개의 단말 노드(terminal node)는 논리 값 0과 1을 나타낸다.

(3) 각 내부 노드(internal node)는 하나의 입력 변수를 갖는다.

(4) 각 내부 노드는 두 개의 간선(edge)을 갖는데, 이들은 노드에 해당하는 변수가 1 일 때의 부분 함수와 0 일 때의 부분 함수를 가리킨다.

[정의 2] DD의 각 경로에서 각 입력 변수가 최소한 한번 있고 각 경로에서 입력변수의 순서가 동일하다면 DD를 ordered하다고 한다.

DD는 아래와 같은 확장 타입을 사용하여 불리안 함수를 표현할 수 있다.

$$\overline{x_i} f_{x_i=0} + x_i f_{x_i=1} \quad : \text{확장형 } d_i = S \quad \text{(Shannon Decomposition)} \quad (1)$$

$$f_{x_i=0} \oplus x_i \cdot g \quad : \text{확장형 } d_i = pD \text{ (Positive Davio Decompositin)} \quad (2)$$

$$f_{x_i=1} \oplus \overline{x_i} \cdot g \quad : \text{확장형 } d_i = nD \text{ (Negative Davio Decomposition)} \quad (3)$$

$$g = f_{x_i=0} \oplus f_{x_i=1}$$

여기서 x_i=0 와 f_{xi}=1 는 각각 x_i=0와 x_i =1에 대한 f의 공통인자(cofactor) 이고, +는 OR 연산자이고 ⊕은 Exclusive-OR 연산자이다.

DD의 한 노드가 식(1)과 같이 shannon 방법으로 확장했을 때 그 노드를 S-노드라 하고 식(2)와 같이 positive davio 방법으로 확장했다면 그 노드를 pD-노드라 한다. 또한 식(3)과 같이 확장되었을 때는 그 노드를 nD-노드라 한다.

[정의 3] OFDD는 입력변수 x_n에 대한 ordered DD이고 DD에서 같은 레벨에 있는 입력변수는 확장 타입 중 pD와 nD로 구성된다.

예를 들면 입력 변수가 n개이고 각 입력변수가 확장된 경우 (그림 2)와 같고 OFDD는 각 노드가 pD와 nD 중 한 가지로 확장되기 때문에 2^n개의 서로 다른 확장 방법이 있을 수 있다.

[정의 4] OKFDD는 입력변수 x_n에 대한 ordered DD이고 DD에서 같은 레벨에 있는 입력변수는 S, pD 그리고 nD 중에서 한 가지의 확장 타입으로 구성된다.

예를 들면 입력 변수가 n개인 경우 (그림 3)와 같고 OKFDD는 각 노드가 S, pD 그리고 nD 중에서 한 가지로 확장되기 때문에 3^n개의 서로 다른 확장 방법이 있을 수 있다.

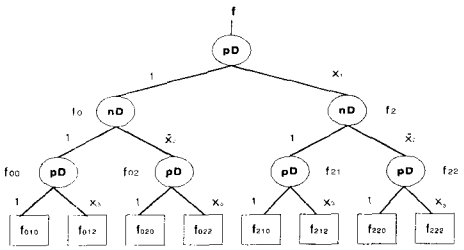


그림 2 n변수 함수에 대한 OFDD의 확장 방법

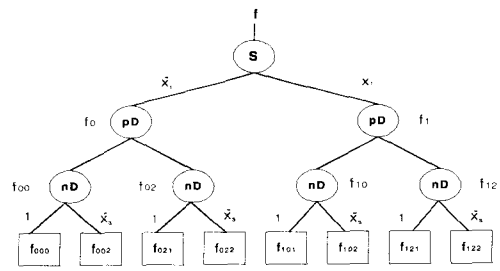


그림 3 n변수 함수에 대한 OKFDD의 확장방법 선택

[정의 5] OPKFDD는 입력변수 x_n에 대한 ordered DD이고 각 노드마다 각기 다른 확장 타입으로 구성된다.

예를 들면 OPKFDD가 입력변수가 n개인 경우 (그림 4)와 같이 각 노드는 서로 다른 확장 방법을 선택할 수 있으므로 OPKFDD는 3^{2^n-1}개의 서로 다른 확장 방법이 있을 수 있다.

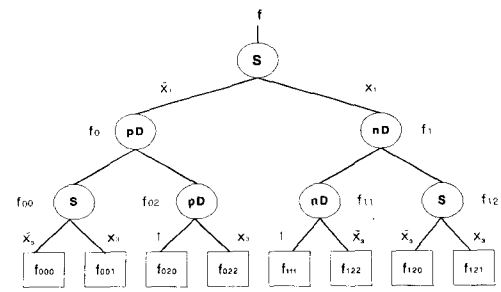


그림 4 n변수 함수에 대한 OPKFDD의 확장방법 선택

2.2 complex term의 개념

[정의 7] 리터럴(literal)은 양극성(positive polarity)나 음극성(negative polarity)을 갖는 변수를 말한다.

[정의 8] Support는 임의 함수에서 사용하는 입력 변수들의 집합이다.

예를 들어 함수 f_1과 f_2가 다음과 같이 주어졌을 때

$$f_1 = ab \oplus a'cd$$
$$f_2 = abe \oplus a'cde$$

함수 f_1의 support는 a, b, c, 그리고 d 이다. 그리고 함수 f_2의 support는 a, b, c, d, 그리고 e 이다.

[정의 9] complex term은 다음과 같이 재귀적으로 정의될 수 있다[14].

1. 하나의 리터럴은 complex term이다.
2. 만일 M이 complex term이고 a가 리터럴이면 $M \cdot a$, $M \oplus a$, $M + a$은 complex term 이다. 단, 하나의 변수는 complex term에 한 번 이상 나타날 수 없다.

위의 정의에 의하면 임의의 product term은 complex term이다. 그러나 complex term은 다음 예제에서 보듯이 단순한 product term 외에 더 많은 불리안 함수를 표현할 수 있다.

(예제 1) 다음은 complex term 들의 예제이다.

$(ab') + c$
$(a + b)c$
$(a \oplus b) + c'$
$(cd + a) \oplus d$

(예제 2) 다음은 complex term이 아닌 경우이다.

$(ab + b')c$
$a + bc' + d$
$(a + b)(c + d)$

$(ab + b')c$은 리터럴 b가 term에서 2번 나타나고, $a + bc' + d$와 $(a + b)(c + d)$는 연산이 좌에서 우로 재귀적으로 전개되지 않았기 때문에 complex term이 아니다.

위의 complex term들은 셀룰라(cellular)구조에서 쉽게 표현될 수 있다[14]. Complex term이 셀 배열에서 구현될 수 있음을 보이는 것은 어렵지 않다. 다음의 (그림 5)와 (그림 6)를 살펴보자. 논리 표현 $(a+b)c+d$는 셀 배열에서 쉽게 구현될 수 있으나 $(a+b)(c+d)$는 complex term이 아니기 때문에 셀룰라 구조에 구현할 때 (그림 5)와 같이 별도의 경로 지정 wire가 필요하다. 다시 말하면 complex term들은 셀룰라 구조 FPGA(Cellular Architecture Field Programmable Gate Array)[14]에서 쉽게 구현될 수 있는 반면에 complex term이 아닌 경우에는 routing wire가 필요하게 되며, 이는 셀룰라 구조 FPGA에 적용될 때 버스나 routing을 위한 셀을 필요로 한다. 만일 함수가 복잡해지게 되면 그에 따라 routing wire의 수도 늘어나게 되고 ASIC 설계 시 routing 복잡도가 커지게 된다. 따라서 FPGA 구현 시 routing으로 인한 많은 수의 셀을 필요로 하게 된다. 이러한 개념을 이용하여 이차원 형태의 셀룰라 FPGA 합성을 위한 연구가 이루어져 왔다[14].

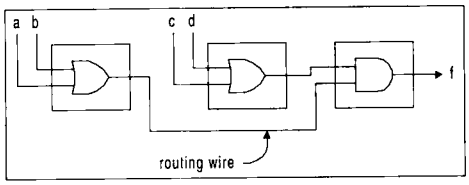


그림 5 f = (a+b)(c+d)

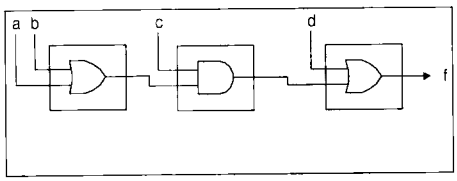


그림 6 f = (a+b)c+d

3. OPKFDD 생성 방법과 입력변수 순서 결정 방법

3.1 OPKFDD의 생성 및 complex term을 이용한 확장형의 결정

여기서는 먼저 OBDD로부터 OPKFDD를 생성하는 방법을 설명한다. OBDD의 각 노드는 Shannon에 의하여 확장된 두 개의 부분함수를 가리키는 THEN-간선과 ELSE-간선을 가진다. 이 때, 노드함수를 f라하고 노드에 해당하는 변수를 v라 하면 THEN-간선은 $f_{v=1}$을 가리키고 ELSE-간선은 $f_{v=0}$을 가리킨다. OPKFDD를 유도하기 위해 먼저 각 노드의 두 간선을 XOR하여 또 $f_{v=0} \oplus f_{v=1}$을 가리키는 XOR-간선을 만든다. OPKFDD에서 사용하는 S, nD, 그리고 pD의 세 가지 확장방법은 이 세 간선을 이용하여 구현할 수 있으므로 이 세 간선 중에서 두 개를 선택한다. 이를 위하여 각 노드마다 비용함수를 설정하여 최소의 complex term을 생성하도록 각 노드의 확장방법을 결정한다. 임의의 노드로부터 XOR-간선을 생성하는 알고리즘이 (그림 7)에 나와 있다.

```
OPKFDD_Decomposition_Choice(d)
/* 각 노드의 확장방법 선택 */
{
  if(d 단말노드이면) {
      근 노드로부터 노드 d까지의 path 생성;
      return;}
  lcost = OPKFDD_GetCost(d->else);
  /* ELSE-간선 노드의 비용함수(그림 9)*/
  rcost = OPKFDD_GetCost(d->then);
  /* THEN-간선 노드의 비용함수 */
  xcost = OPKFDD_GetCost(d->xor);
  /* XOR-간선 노드의 비용함수 */
  int buf = 0;  /* path를 저장 */
  /* lcost, rcost, xcost 중에서  최소의 비용함수를 갖는
  두 개를 선택  */
  if( lcost와 rcost 선택)  {      /* S 확장 사용 */
    node_free(d->xor);
    buf = buf + "v' ·", OPKFDD_Decomposition_
          Choice(d->else); /* v는 d의 변수 */
    buf = buf + "v ·", OPKFDD_Decomposition_
          Choice(d->then);
    }
  if( rcost와 xrcost 선택) {       /* nD 확장 사용 */
    node_free(d->else);
    OPKFDD_Decomposition_Choice(d >then);
    buf = buf + "v' ·", OPKFDD_Decomposition_
          Choice(d >xor);
    }
   if( lcost과  xcost 선택) {       /* pD 확장 사용 */
    node_free(d->then);
    OPKFDD_Decomposition_Choice(d->else);
    buf = buf + "v ·", OPKFDD_Decomposition_
          Choice(d->xor_node);
    }
   return;
}
```

그림 7 확장 방법 선택 알고리즘

위 방법을 예제를 통하여 설명한다. 어떤 함수 F가 $a\bar{b} + a\bar{c} + \bar{a}bc$라 하자. 그리고 입력변수 순서가 a, b, c일 때 (그림 8)와 같이 각 노드의 ELSE-간선($f_{x_{i-0}}$)과 THEN-간선($f_{x_{i-1}}$)에 ELSE-간선($f_{x_{i-0}}$)과 THEN-간선($f_{x_{i-1}}$)을 XOR ($f_{x_{i-0}} \oplus f_{x_{i-1}}$) 시킨 XOR-간선을 갖도록 한다.

(그림 7)의 알고리즘에서 확장방법을 선택하기 위해 (그림 9)의 complex term의 수를 계산하는 비용함수를 이용한다. 이 함수는 어떤 노드를 근 노드로 하여 전개되는 모든 가능한 complex term의 수를 계산하며 주어진 노드 수 n에 대하여 $O(n)$ 시간에 수행되므로 매우 빠른 시간에 계산이 가능하며 OPKFDD의 노드 수에 대해서 매우 근접한 평가함수(estimation function)로 사용할 수 있다.

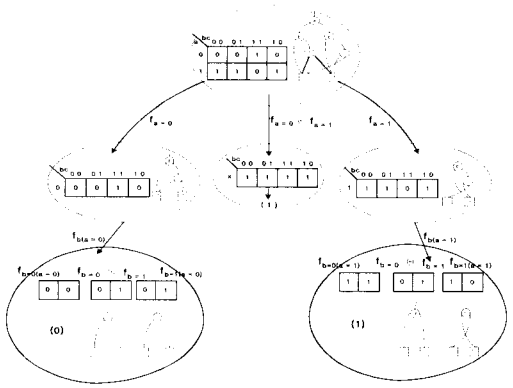


그림 8 각 노드에 대한 XOR-간선 생성

```
/* OPKFDD의 각 간선에서 생성되는 complex term 수
   계산하는 함수 */
OPKFDD_GetCost(d)
{ /* 노드 d에서의 complex term 수를 return */
  if (d is visited) return d->cost
  If( d is the last variable in the path) return 1;
  /* ELSE-간선의 complex term 수 계산 */
  l = OPKFDD_GetCost(d->else);
  /* THEN-간선의 complex term 수 계산 */
  r = OPKFDD_GetCost(d->then);
  /* XOR-간선의 complex term 수 계산 */
  x = OPKFDD_GetCost(d->xor)
  /* d->xor = d->then ⊕ d->else */
  return d->cost = l + r + x - max(l, r, x);
}
```

그림 9 complex term 수를 계산하는 비용함수

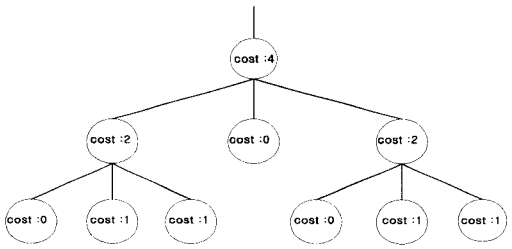


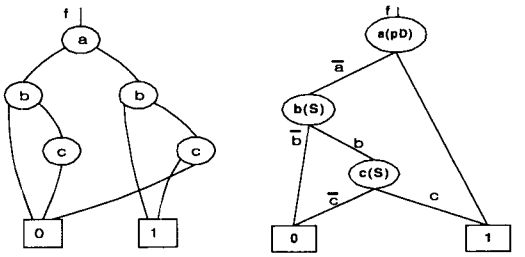
그림 10 노드의 비용

위의 함수 F에 (그림 9)의 비용함수를 적용하면 (그림 10)과 같다. (그림 10)의 예제에서 계산된 결과를 이용해서 각 노드의 ELSE-간선, THEN-간선 그리고 XOR-간선 중에서 최소의 complex term 수를 갖는 2개의 확장 방법을 선택한다. 즉, 노드가 만약

최소의 complex term 수를 갖는 ELSE-간선과 THEN-간선이 선택되면 그 노드는 *S*-노드가되고, ELSE-간선과 XOR-간선이 선택되면 *pD*-노드이고 THEN-간선과 XOR-간선 선택되면 *nD*-노드가 된다.

이제 계산된 노드의 비용을 이용하여 (그림 11)의 (b)와 같이 위의 함수 *F*에 대한 OPKFDD를 생성한다. 입력변수 *a*는 비용이 적은 ELSE-간선과 XOR-간선을 선택하여 *pD*형으로 확장하고 입력변수 *b*와 *c*는 S 형 확장을 한다.

(그림 11)의 (a)와 (b)를 비교해 보면 같은 함수를 각 노드의 확장 방법을 어떻게 선택하느냐에 따라 DD의 크기가 달라진다. (a)는 모든 노드를 shannon으로 확장하는 OBDD형태로 표현한 경우이고 (b)는 본 논문에서 제시하는 방법으로 OPKFDD를 생성한 경우이다. 같은 함수를 표현하지만 DD 종류에 따라서 즉, 각 노드의 확장형에 따라서 노드 수가 다름을 알 수 있다.



(a) OBDD (b) OPKFDD

그림 11 OBDD 대 OPKFDD 노드의 확장 방법

3.2 함수간의 포함관계를 이용한 입력변수 순서 결정 방법

일반적으로 DD의 최소화는 입력 변수의 순서를 고려함에 따라 그 크기에 있어서 많은 차이가 있다. 본 연구에서의 OPKFDD는 각각의 노드에서 어떤 확장 방법을 선택하느냐에 따라 사용되어 지는 노드의 수가 결정되므로 입력 변수의 순서와 확장 방법의 선택을 함께 고려하여야만 한다. 앞 절에서 언급한 것과 같이 노드 확장 방법은 BDD를 수정하여 OPKFDD 형성 시 선형시간 내에 결정되므로 여기에서는 입력 변수의 순서에 대해서 살펴보기로 하겠다.

인접 변수 교환 방법을 확장시킨 sifting 알고리즘[13]은 각 변수별로 노드의 수에 따라 정렬한 다음 다른 변수들은 고정한 채로 각 변수에 대하여 최적의 위치를 찾는 방법이다. 본 연구에서는 함수간의 포함관계를 고려하여 변수들의 그룹을 구하고 각 그룹별로 그룹내의 변수의 지역적인(local) 순서를 정하기 위해 sifting 알고리즘[13]을 확장한 그룹간의 sifting방법을 사용한다.

본 연구에서는 다 출력 함수의 입력변수 순서 결정은 함수들의 각각을 가지고 결정하기보다는 함수간의 포함관계를 고려하여 결정한다. 함수간의 포함관계는 support를 이용하여 구하며 이를 하나의 그룹으로 간주하여 그룹간의 부분적인(partial) 순서를 결정한다. 그리고 마지막으로 각 그룹내의 입력변수들을 sifting 알고리즘을 이용하여 수정하여 나감으로써 최종적인 입력변수의 순서를 결정한다.

예를 들어 다음의 출력 함수 f_0와 f_1을 보자.

$$\begin{aligned} f_0 &= ab \oplus a'cd \\ f_1 &= abe \oplus a'cde \\ &= e(ab \oplus a'cd) = ef_0 \end{aligned}$$

출력 함수 f_0는 두 개의 complex term으로 구성되고, 출력 함수 f_1은 두 개의 complex term으로 구성된다. 만일 이들이 서로의 포함 관계를 고려하지 않고 각각 complex term을 구성한다면 4개의 term 이 필요하게 된다. 그러나 만일 f_0을 complex term을 구성하는 하나의 입력으로 사용한다면 $f_1 = e(ab \oplus a'cd) = ef_0$로 하나의 complex term으로 f_1을 구성할 수 있다. 이러한 주출력 함수들의 포함 관계를 이용하기 위해서는 본 연구에서 사용하는 함수 표현 방법인 OPKFDD에서의 변수 순서가 적절히 결정되어야 한다. (그림 12)는 이러한 주출력 함수들의 포함 관계를 고려한 입력 변수 순서의 결정 알고리즘을 보여준다. 임의의 함수에서 사용하는 입력변수들의 집합을 그 함수의 support라고 정의할 때, 각 주출력 함수에서의 support를 찾은 후에 이들의 포함 관계를 고려하여 변수의 순서를 결정한다. 그리고 이 알고리즘의 수행시간은 주출력 함수의 수 n에 대하여 $O(n^2)$ 시간에 수행된다. 이 방법은 4절의 실험결과에 제시된 다른 reordering 방법에 비해서 빠른 시간에 수행된다. 예를 들어 위의 함수 f_0의 support는 $\{a,b,c,d\}$ 이고 함수 f_1의 support는 $\{a,b,c,d,e\}$ 일 때 각 출력함수의 support 사이의 포함관계를 이용하여 그룹간의 부분적인 입력변수 순서가 다음과 같이 만들어진다.

$$\{a,\ b,\ c,\ d,\ e\} \ > \ \{a,\ b,\ c,\ d\}$$

만일 support(f_0)과 support(f_1)이 서로 포함 관계를 갖지 않으면 위의 관계가 성립하지 않는다. 이러한

방식으로 부분순서가 결정되면, 각 그룹들을 서로 disjoint한 부분들로 나누고 support에서 공통 입력변수를 찾아내어 그룹으로 나눈다. 그리고 이 그룹들은 다음과 같은 입력변수 부분적인 순서(partial ordering)를 구할 수 있다.

$$\{e\} > \{a,b,c,d\}$$

즉, 그룹 $\{a,b,c,d\}$ 는 그룹 $\{e\}$의 앞에 위치해야 사용되는 complex term의 수를 줄일 수 있다는 것을 알 수 있다. 이렇게 나누어진 그룹들에 대해서 그룹 내의 입력변수들을 sifting 알고리즘[13]을 적용하여 최종적인 입력 변수의 순서를 결정한다.

```
/* 각 함수의 포함관계를 고려한 입력변수 순서를 결정 */
Group_sift(f) /* f에 대한 입력변수 순서 결정 */
{
  /* 각 함수의 support의 포함관계를 이용하여 그룹간의
     partial order 결정 */
  for i=1, num_POs { /* PO는 Primary output */
     for j=1, num_POs {
       if support(fi) ⊂ support(fj) ∀i, j {
         group(i) = support(fi);
         group(j) = support(fj) - support(fi);
         group(j) > group(i);
  /* > 은 그룹들 사이의 partial ordering */
       }
  }
  /* 각 그룹들을 서로 disjoint한 그룹으로 partition함 */
   for i=1 num_group {
     for j=1 num_group {
       if group(i) ∩ group(j) ≠ φ
         new_group = group(i) ∩ group(j);
         group(i) = group(i) - group(j);
         group(j) = group(j) - group(i);
     }
   }
  /* 각 그룹들 내의 입력변수의 local 순서결정 */
  for i=1, num_group {
     for j= 1, num_variable_group {
       /* 각 그룹 내에 있는 변수들에 대해서 참고문헌 [13]
          의 방법을 적용 */
         Local_Sift();
     }
  }
}
```

그림 12 다 출력 함수의 입력변수 결정 알고리즘

4. 구현 및 실험결과

본 논문에서 제시한 방법은 C언어로 구현했고, 펜티엄 PC에서 실행했다. 실험 결과는 주어진 벤치마크(benchmark) 회로에 대하여 본 논문에서 제안한 방법과 CUDD에서 제공하는 reordering 방법을 노드 수로 비교하였다. 그리고 실행 시간은 user time으로 하고 단위는 초(second)이다.

표 1 DD 종류에 따른 노드 수 비교

Benchmark	OBDD		OKFDD		OPKFDD	
회로	#node	time	#node	time	#node	time
5xp1	142	0.33	142	0.27	88	0.27
card4	151	0.30	151	0.29	74	0.26
cmlp4	434	6.10	414	4.53	247	3.67
cnrm	403	2.54	405	2.25	315	1.84
cu	574	3.41	100	1.86	100	1.47
f5lm	151	0.37	141	0.34	78	0.33
inc	191	1.46	172	1.55	143	1.30
mlp3	87	0.31	75	0.27	51	0.25
rd53	66	0.16	66	0.17	31	0.18
rd73	289	0.34	137	0.29	101	0.28
sao2	236	1.41	341	1.63	213	1.19
t481	1680	0.75	32	0.52	32	0.41
total	4404	17.48	2176	13.97	1473	11.45

<표 1>의 결과는 각 노드의 확장 방법에 따른 BDD OKFDD 그리고 OPKFDD의 노드 수를 비교한 것이고, <표 2>는 CUDD에서 제공하는 reordering 방법과 본 논문에서 제안한 방법을 비교하여 제시한다. 단 제시한 결과는 각각의 order 결정 방법에 따라서 구해진 입력순서를 가지고 본 논문에서 제안한 OPKFDD 구성을 위한 노드 확장 방법을 적용한 결과다.

<표 1>에서 알 수 있는 것은 확장 방법에 따라서 DD의 크기가 다름을 알 수 있다. OPKFDD는 OBDD나 OKFDD와 비교해서 노드 수가 작음을 알 수 있다. 특히 OBDD와 비교해서 1/3 정도 감소함을 알 수 있다. 또 OKFDD와 비교해도 67% 정도 노드 수의 감소를 관측할 수 있다. 즉, 각 노드의 확장 방법을 잘 선택함으로써 DD의 크기를 최소화할 수 있다.

A : 함수간의 포함관계를 이용하여 그룹 shifting 적용한 결과.

- CUDD의 OBDD ordering을 그대로 사용함[15] OBDD ordering 결과로부터 시작하여 각 노드에서 XOR-간선을 형성하고 complex term 비용함수에 의하여 OPKFDD로 노드 확장함수를 선택한 결과

#2 : CUDD_REORDER_RANDOM

입력변수 중에서 랜덤하게 두 개의 변수를 선택한 하나의 변수 쌍을 입력순서에서 서로 교환하여 최적의 입력변수 순서를 찾아가는 방법

표 2 reordering 방법에 따른 노드 수 비교

Benchmar	#2		#4		#6		#8		#14		A	
k 회로	#node	time	#node	time	#node	time	#node	time	#node	time	#node	time
5xp1	147	10.38	88	5.37	88	5.76	137	2.58	100	6.59	88	0.27
card4	77	9.14	88	5.36	88	5.86	91	1.26	82	6.19	74	0.26
cmlp4	274	14.93	283	12.59	283	12.98	249	5.98	275	14.88	247	3.67
cnrm	403	12.04	448	10.88	448	11.54	420	5.35	435	11.92	315	1.84
cu	98	9.33	96	5.15	96	5.65	106	1.39	102	6.43	100	1.47
f51m	79	9.37	78	5.14	78	5.97	78	1.26	79	6.18	78	0.33
inc	163	9.98	172	6.71	172	6.98	123	1.53	172	7.42	143	1.30
mlp3	61	8.76	58	5.07	58	5.83	55	1.19	56	6.11	51	0.25
rd53	31	8.75	31	4.93	31	5.73	31	1.08	31	6.65	31	0.18
rd73	101	8.83	101	5.27	101	5.88	101	1.22	101	6.52	101	0.28
sao2	327	10.23	307	6.15	307	6.78	312	2.67	301	7.71	213	1.19
t481	39	10.51	39	6.44	39	7.04	39	2.07	39	8.15	32	0.41
total	1800	122.25	1789	79.06	1789	86	1742	27.58	1773	94.73	1473	11.45

#4 : CUDD_REORDER_SIFT[13]

#6 : CUDD_REORDER_SYMM_SIFT[17]

#8 : CUDD_REORDER_WINDOW2[18]

[18]의 알고리즘을 적용하고 window 크기가 2인 방법

#14 : CUDD_REORDER_그룹_SIFT[19]

<표 2>의 실험결과를 보면, CUDD에서 제공하는 reordering 방법과 비교해서 본 논문에서 제안하고 있는 변수 순서 결정 방법이 노드 수가 18% 정도 감소함을 알 수 있다. 그리고 실행시간을 A와 비교해 보면, #2 방법보다 6.3배 빠르고 #8 방법보다 1.5배 빠름을 알 수 있다.

5. 결 론

본 논문에서는 OPKFDD을 이용한 불리안 함수의 최적화 표현을 위해서 각 노드의 확장 방법과 입력변수 순서를 결정하는 방법을 제안하였다. 각 노드의 확장 방법은 OBDD의 ELSE 간선과 THEN 간선을 exclusive-OR 시킨 XOR 간선을 추가한 후에 최소의 노드 수를 갖는 complex term의 개념을 이용한 비용함수를 사용하여 두 개의 간선만을 선택함으로써 결국에는 최소의 크기를 갖는 OPKFDD로 불리안 함수를 표현하여 최적화 한다. 또한 입력변수 결정 방법은 다 출력 함수의 경우 함수 각각을 개별적으로 취급하는 것이 아니라 함수의 포함 관계를 고려하여 입력변수들을 그룹으로 나누고, 그룹의 종속관계에 따라 그룹의 순서를 결정한 다음 다시 그룹 내에 있는 변수들을 local order를 결정하기 위해 sifting 방법을 적용하여 최종적인 입력변수 순서를 결정하는 그룹-sifting 알고리즘을 제시했고, 이것을 CUDD 패키지에 적용하여 보았다. 그리고 기존의 reordering 방법과 비교해 볼 때 개선된 결과를 보인다.

참 고 문 헌

[1] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Trans. on Computer*, pp.677-691, 1986.

[2] K. S. Brace, R. C. Rudell, and R. E. Bryant, "Efficient Implementation of a BDD Package," *In Proceedings 27th Design Automation Conf.*, pp.40-46, 1990.

[3] R. Drechsler, N. Drechsler, and W. Gunther, "Fast Exact Minimization of BDDs," *In Proceedings 34th Design Automation Conf.*, pp.200-205, 1998.

[4] B. Becker, R. Drechsler, and M. Theobald. "On the Implementation of a Package for Efficient Representation and Manipulation of Functional Decision Diagrams," *IFIP Workshop on the Applications on the Reed Muller Expansions*, pp.162-169, 1993.

[5] R. Drechsler, M. Theobald, and B. Becker. "Fast OFDD based Minimization of Fixed Polarity Reed-Muller Expansions," *In Proceeding of EDAC*, pp.2-7, 1994.

[6] T. Sasao and M. Fujita, *Representations of Discrete Functions*, Kluwer Academic Publishers, 1996.

[7] R. Drechsler, and B. Becker, "On Variable Ordering and Decomposition Type Choice in OKFDDs," *IEEE Tran. on Computer*, pp.1398-1403, Nov. 1998.

[8] R. Drechsler, and B. Becker, "Ordered Kronecker Functional Decision Diagrams- A Data Structure for Representation and Manipulation of Boolean Functions," *IEEE Tran. on Computer-Aided Design of Intergration Circuits and Systems*, pp.965-973, Oct. 1998.

[9] B. Bollig, P. Savicky, and I. Wegener, "On the Improvement of Variable Orderings for OBDDs," *IFIP Workshop on Logic and Architecture Synthesis, Grenoble*, pp.71-80, 1994.

[10] S. Malik, A. Wang, R. Brayton, and A. Sangiovanni Vincentelli, "Logic Verification Using Binary Decision Diagrams in Logic Synthesis Environment," *In Proceedings International Conference on Computer-Aided Design*, pp.6-9, Nov. 1988.

[11] S. Minato, N. Ishiura, and S. Yajima, "Shared

Binary Decision Diagram with Attributed Edges for Efficient Boolean Function Manipulation," *In Proceedings 27th Design Automation Conf.*, pp.52-57, June 1990.

[12] Rolf Drechsler, Bernd Becker and Nicole Drechsler, "Minimization of OKFDDs by Genetic Algorithms," *International Symposium on Soft Computing, Reading*, pp. B:528-B:263, 1996.

[13] R. Rudell, "Dynamic Variable Ordering for Ordered Binary Decision Diagrams," *In IEEE International Conference on Computer-Aided Design*, pp.42-47, 1993.

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

[15] *ftp://vlsi.colorado.edu/pub/cudd-2.2.0.tar.gz.*

[16] R. Drechsler, personal communication, 1999.

[17] S. Panada, F. Somenzi, and B. F. Plessier, "Symmetry Detection and Dynamic Variable Ordering of Decision Diagrams," *In proceedings International Conference on Computer-Aided Design*, pp.628-631, 1994.

[18] M. Fujita, Y. matsunaga, and T. Kakuda, " On Variable Ordering of Binary decision Diagrams for the Application of Multi-level Logic Systhesis," *In proceedings of the European Conference on Design Automation*, pp.50-54, 19914.

[19] S. Panda and F. Somenzi, "who are the Variables in your Neighborhood," *In proceedings International Conference on Computer-Aided Design*, pp.74-77, 1995.

정 미 경

1987년 전남대학교 전산통계학과 이학사. 1989년 전남대학교 대학원 전산통계학과 이학석사. 2000년 전남대학교 대학원 전산통계학과 이학박사. 관심분야는 VLSI/CAD, 논리합성, 인공지능

김 미 영

1983년 전남대학교 계산통계학과 이학사. 1985년 이화여자대학교 대학원 이학석사. 1997년 전남대학교 대학원 전산통계학과 이학박사. 1986년 ~ 1997년 목포과학대학 전자계산과 부교수. 1998년 ~ 현재 전남도립담양대학 전산·정보통신공학부 부교수. 관심분야는 CAD/VLSI, 멀티미디어 등

이 귀 상

1980년 서울대 공대 전기공학과 학사. 1982년 서울대 대학원 전자계산기공학과 석사. 1982년 금성통신 연구소 근무. 1991년 Pennsyivania 주립대학 박사. 1984년 ~ 현재 전남대 전산학과 정보통신 연구소 부교수. 관심분야는 VLSI/CAD, 멀티미디어 시스템, 테스팅, 논리합성