韓國産業應用學會論文誌 제3권 제1호 17-27, 2000.2
J. of Korea Society of Industrial Application, Vol.3, No.1, 17-27, February. 2000

부울 분해식 산출 방법
Boolean Factorization

권 오 형*
Oh-Hyeong Kwon

<Abstract>

A factorization is an extremely important part of multi-level logic synthesis. The number of literals in a factored form is a good estimate of the complexity of a logic function, and can be translated directly into the number of transistors required for implementation. Factored forms are described as either algebraic or Boolean, according to the trade-off between run-time and optimization. A Boolean factored form contains fewer number of literals than an algebraic factored form. In this paper, we present a new method for a Boolean factorization. The key idea is to build an extended co-kernel cube matrix using co-kernel/kernel pairs and kernel/kernel pairs together. The extended co-kernel cube matrix makes it possible to yield a Boolean factored form. We also propose a heuristic method for covering of the extended co-kernel cube matrix. Experimental results on various benchmark circuits show the improvements in literal counts over the algebraic factorization based on Brayton's co-kernel cube matrix.

Key words : factored form, kernel, co-kernel

1. 서 론

논리함수는 리터럴(literal) 개수가 다른 여러 형태의 논리식으로 표현될 수 있다. MOS 회로의 경우 트랜지스터의 개수는 논리식의 리터럴 개수와 비례한다. 즉, 논리식의 리터럴 개수를 최소화하면 회로의 트랜지스터 개수를 최소화할 수 있게 된다. 따라서, 논리식 최적화의 한 분야가 최소 개수의 리터럴을 갖는 논리식을 산출하는 것이다. 일반적으로, 분해식(factored form)은 단순식(sum-of-products form)보다 적은 리터럴 개수로 동일한 논리함수를 표현할 수 있는 수단이다. Lawler[1]는 최적 분해식을 산출할 수 있는 알고리즘을 제안하였다. 이 알고리즘은 주어진 논리함수에 대하여 가능한 모든 분해식을 산출하고 그 중에서 가장 적은 리터럴 개수를 갖는 분해식을 산출하는 전수 탐색(exhaustive search) 방법을 사용한다. 따라서, 매우 작은 논리함수 또는 회로에 대한 분해식을 산출하는 데를 제외하고는 장시간의 수행

*정회원, 위덕대학교 컴퓨터공학과, 전임강사, 工博
포항공대 졸업
780-713 경북 경주시 강동면 유금리 산50번지
E-mail ohkwon@mail.uiduk.ac.kr

Full-time Lecturer, Dept. of Computer Engineering, Uiduk University, Ph.D.

시간이 소요되기 때문에 실용성이 없다. 따라서, 분해식 산출에 대해서는 주로 선험 방법(heuristic method)이 적용되고 있다. 분해식 산출의 선험 방법은 분해식 산출 수행 시간과 리터럴 개수의 최적화 중에 어디에 더 중점을 두느냐에 따라 대수 분해 방법과 부울 분해 방법으로 구분한다. 대수 분해 방법에 있어서는 논리함수를 대수 다항식으로 간주한다.

Brayton 등[2-5]은 코커널/커널을 이용하여 대수 분해식을 산출하는 방법 즉, 커널 기반 방법(kernel-based method)을 제안하였다. 이 분해 방법은 분해식 산출 속도를 빠르게 향상시켰으나, 때때로 최소 개수의 리터럴을 갖는 분해식을 산출할 수 없다. 그러나, 부울 분해 방법은 대수 분해 방법과 비교하면 보다 적은 개수의 리터럴을 갖는 분해식을 산출할 수 있는 장점을 갖는다.

다음은 최근의 부울 분해 방법들이다. Liao 등[6]은 다치 함수(multi-valued function)와 분지 및 한계 커버 방법 (branch - and - bound covering)을 이용한 부울 분해 방법을 제안하였다. Stanion 등[7]은 Liao의 방법이 대수 분해 방법에 비해 리터럴 개수를 줄이는 데 효과가 매우 작다고 지적하였다. 그래서, 효과를 높이기 위해서 Stanion 등은 부울 분해식을 산출하는 데 Binary Decision Diagram (BDD)를 이용하였다. 또한, 이들은 부울 나눗셈과 부울 분해를 수행하기 위해서 BDD를 활용하는 방법을 제시하였다. 그러나, 이 방법은 때때로 대수 분해에 의한 분해식보다 리터럴 개수가 늘어나는 결과를 초래한다. 이처럼 최근의 연구들을 볼 때, 최소 개수의 리터럴을 갖는 부울 분해식 산출에 대한 연구가 더 필요하다.

본 논문에서는 선험 부울 분해 방법을 제안한다. 제안한 방법은 Brayton 등이 제안한 SIS의 커널 기반 방법(즉, 대수 분해 방법)을 개선한 것이다. Brayton의 커널 기반 방법은 코커널/커널들로부터 코커널 큐브 행렬을 만들고, 이를 이용하여 대수 분해식만을 산출한다. 그러나 본 논문에서 제시한 방법은 코커널 큐브 행렬을 확장하여 부울 분해식이 산출되도록 고안한 것이다. 논문의 구성은 다음과 같다. 1장은 지금까지 서술한 서론이다. 2장에서는 본 논문에서 제시하는 부울 분해 방법과 실험 결과에 대하여 서술한다. 3장에서 본 논문의 결론이다.

2. 부울 분해 방법

본 논문에서 제시하는 분해 방법을 서술하는 데 기본이 되는 용어에 대하여 서술한다. 다음 커널/커널과 확장된 코커널 큐브 행렬에 대하여 소개한다. 확장된 코커널 큐브 행렬을 이용하여 부울 분해식을 산출하는 알고리즘을 제시한다.

정의 1: **변수**(variable)는 부울 공간(Boolean space)에서 한 좌표를 나타내는 문자다. **리터럴**(literal)은 변수 그 자체 또는 그의 보수(complement)다. **큐브**(cube)는 리터럴들의 집합으로 만일 리터럴 a가 존재하면, 그의 보수 리터럴 a'을 포함하지 않는다. **단순식**(expression 또는 sum-of-products(SOP) form)은 큐브들의 집합이다.

예 1: 문자 a는 변수다. a와 a'은 리터럴이다. 리터럴 집합 $\{a, b\}$는 큐브, 그러나 $\{a, a'\}$은 큐브가 아니다. $\{\{a, b'\}, \{b, c\}\}$는 단순식이다.

본 논문에서는 큐브와 단순식을 표현하는 경우 집합 표기와 보편적으로 사용되는 수식 표기를 모두 사용한다. 따라서 큐브 $\{a, b\}$는 ab와 동일한 표현이며, 단순식 $\{\{a, b'\}, \{b, c\}\}$는 $ab' + bc$와 동일한 표현이다.

정의 2: **서포트**(support)는 단순식 F에 사용된 모든 변수들이다. 이 때, 단순식 F의 서포트를 $sup(F)$로 표기한다. 그러면,

$sup(F) = \{x \mid$ 큐브 $C \in F$ 에 대하여, $x \in C$ 또는 $x' \in C\}$.

예 2: 단순식 $a + bc'$의 서포트는

$sup(a + bc') = \{a, b, c\}$ 이다.

정의 3: 단순식을 구성하는 모든 큐브들에 대하여, 공통으로 쓰인 리터럴이 없다면 그 단순식은 **큐브면제**(cube-free) 되었다고 한다. 단순식이 어떤 큐브로부터 나누어졌을 때 몫이 큐브면제라면, 그 몫을 **커널**(kernel)이라 한다. 이 때 커널을 산출한 큐브를 **코커널**(co-kernel)이라 한다.

예 3: 단순식 $ab+c$는 큐브면제, 그러나 $ab+ac$와 abc는 큐브면제가 아니다. 단순식 $F=bc'd'e+ab'c+ab'e+ac'd'$에 대하여, F의 커널과 코커널의 예를 보이기 위해서 편의상 다음과 같이 표현하자. 즉,
$F=bc'd'e+a(b'c+b'e+c'd')$ 로 표현된 경우, $b'c+b'e+c'd'$는 커널이 되며, 이 때 코커널은 a이다.

정의 4: **분해식**(factored form)은 단순식들이 합과 곱으로 표현된 것이다. 구체적인 정의는 다음과 같다.

1) 리터럴은 분해식이다.
2) 분해식들의 곱은 분해식이다.
3) 분해식들의 합은 분해식이다.

분해식은 단순식들이 합과 곱으로 반복해서 표현된 논리식이다.

정의 5: 등멱법칙 ($a\cdot a=a$)과 보수법칙 ($a\cdot a'=0$) 을 적용할 필요없이 분해식을 구성하는 단순식들을 곱할 수 있는 경우 **대수 분해식**(algebraic factored form)이라 한다. 대수 분해식 이외의 경우 **부울 분해식** (Boolean factored form)이라 한다.

예 4: $F=bc'd'e+ab'c+ab'e+ac'd'$ 와 $F=a(b'(c+e)+c'd')+bc'd'e$ 모두 대수 분해식이다. $F=(a+be)(b'(c+e)+c'd')$는 부울 분해식이다.

2.1 커널/커널 산출

커널/커널들은 코커널/커널들로부터 산출한다. 본 논문에서는 커널/커널 및 코커널/커널을 표시할 때 괄호를 이용한다. C를 코커널들의 집합이라 하고, K를 커널들의 집합이라 하자. 그리고, (c_i, k_i)와 (c_j, k_j)를 각각 코커널/커널이라 하고, $c_i\in C$, $c_j\in C$, $k_i\in K$, $k_j\in K$이고 $i\neq j$이라고 하자. 이 때 $c_i\in k_j$이고 $c_j\in k_i$인 경우 (k_i, k_j)를 커널/커널이라 한다.

예 5: 단순식 $F=bc'd'e+ab'c+ab'e+ac'd'$가 주어졌다고 하자. 그러면, F에 대한 코커널/커널들은 다음과 같다.

코커널	커널
a	$b'c+b'e+c'd'$
ab'	$c+e$
$c'd'$	$be+a$
e	$bc'd'+ab'$

위의 코커널/커널 집합에서 두 개의 코커널/커널들 $(a, b'c+b'e+c'd')$과 $(c'd', be+a)$의 경우를 보자. $(a, b'c+b'e+c'd')$를 첫 번째 코커널/커널이라 하고, $(c'd', be+a)$를 두 번째 코커널/커널이라 하자. 그러면, 첫 번째 코커널/커널에 속하는 커널 $b'c+b'e+c'd'$는 두 번째 코커널/커널에 속하는 코커널과 동일한 큐브 $c'd'$를 포함한다. 또한 두 번째 코커널/커널에서 커널 $be+a$는 첫 번째 코커널/커널에 속하는 코커널 a와 동일한 큐브를 포함한다. 이러한 경우 첫 번째와 두 번째의 코커널/커널들에서 커널들만을 추출하여 커널/커널을 산출한다. 이 경우 $(b'c+b'e+c'd', be+a)$인 커널/커널이 얻어진다. 마찬가지로, 두 개의 코커널/커널 $(ab', c+e)$와 $(e, bc'd'+ab')$으로부터 커널/커널 $(c+e, bc'd'+ab')$이 산출된다. 단순식 F에 대한 커널/커널을 정리하면 다음과 같다.

커널	커널
$b'c+b'e+c'd'$	$be+a$
$c+e$	$bc'd'+ab'$

2.2 확장된 코커널 큐브 행렬의 산출

단순식 F가 주어졌을 때, 확장된 코커널 큐브 행렬을 만들기 위해서 코커널/커널들 및 커널/커널들을 산출한다. 또한 단순식 F의 각 큐브 C_i에 양의 정수인 인덱스, 즉 $index(C_i)$를 부여한다. 이러한 인덱스들은 코커널 큐브 행렬의 원소 값으로 이용된다. 지금부터 대수 분해식 산출에 이용된 코커널 큐브 행렬은 B로 표기하고, 본 논문에서 제시하는 확장된 코커널 큐브 행렬은 M으로 표기한다. CS를 코커널들의 집합, KS를 커널들에 속하는 큐브들의 집합, 그리고 SS를 커널/커널들에 포함된 큐브들의 집합이라 하자. 그러면, M의 각 행은 $CS \cup SS$에 속하는 각 큐브에 상응하고, 각 열들은 KS의 큐브에 상응한다. α를 M의 임의의 행이라 하고, β를 M의 임의의 열이라 하자. 그러면, $\alpha \in CS \cup SS$, $\beta \in KS$ 이다. M의 원소 $M(\alpha,\beta)$는 다음과 같은 값을 갖는다.

$$M(\alpha,\beta)=\begin{cases} B(\alpha,\beta), & \alpha \in CS-SS,\ \beta \in kernel(\alpha) \\ index(\alpha\beta), & \alpha \in SS-CS,\ \alpha\beta \neq 0,\ \alpha와\ \beta는 커널/커널의\ 각\ 커널의\ 큐브 \\ *, & \alpha \in SS-CS,\ \alpha\beta=0,\ \alpha와\ \beta는\ 커널/커널의\ 각\ 커널의\ 큐브 \\ 0, & 그\ 외의\ 경우 \end{cases}$$

여기서, $kernel(\alpha)$는 코커널 α에 상응하는 커널이다. $B(\alpha,\beta)$는 다음과 같다.

$$B(\alpha,\beta)=\begin{cases} index(\alpha\beta), & \beta \in kernel(\alpha) \\ 0, & \beta \notin kernel(\alpha) \end{cases}$$

위의 정의에서 $M(\alpha,\beta)=B(\alpha,\beta)$는 M이 B의 0이 아닌 모든 원소를 포함함을 의미한다. $M(\alpha,\beta)=index(\alpha,\beta)$와 $M(\alpha,\beta)=*$는 각각 부울 공리인 등멱법칙과 보수법칙에 따라 정의한 것이다. 따라서, M은 B의 수퍼 행렬(super matrix)이다.

예 6: 예5에서 사용한 동일한 단순식 $F=bc'd'e+ab'c+ab'e+ac'd'$의 코커널/커널들과 커널/커널들을 이용하여 확장된 코커널 큐브 행렬을 구한다.

먼저 F의 각 큐브에 양의 정수를 배당한 Table 1의 인덱스 테이블을 만든다. 또, 코커널 집합 $CS=\{a, ab', c'd', e\}$와 커널에 속하는 큐브들의 집합 $KS=\{b'c, b'e, c'd', c, e, be, a, bc'd', ab'\}$을 얻는다.

그러면, F에 대한 코커널 큐브 행렬 B는 Fig.1과 같다. 에 5로부터 커널/커널에 포함된 큐브들의 집합은 $SS=\{b'c, b'e, c'd', be, a, c, e, bc'd', ab'\}$이 된다.

그러면, $SS-CS=\{b'c, b'e, be, c, bc'd'\}$. $SS-CS$에 포함된 각 큐브는 코커널 큐브 행렬 B에 새로이 추가될 행을 나타낸다. Fig. 2는 확장된 코커널 큐브 행렬 M을 나타낸 것이다. Fig. 2에서 위로부터 다섯번째 행 $b'c$, 좌로부터 일곱번째 열 a에 해당하는 원소 $M(b'c,a)$의 값은 커널/커널 $(b'c+b'e+c'd', be+a)$로부터 다음과 같은 방법으로 계산된다.

Fig. 2에서 다섯번째 행의 큐브 $b'c$는 커널 $b'c+b'e+c'd'$에 포함되며, 일곱 번째 열의 큐브 a는 커널/커널 $(b'c+b'e+c'd', be+a)$의 커널 $be+a$ 에 포함된다.

여기서 두 개의 큐브 $b'c$와 a의 곱 $ab'c$는 Table1의 인덱스 테이블에서 $index(ab'c)=2$가 된다. 따라서, $M(b'c,a)=2$ 가 된다.

또, 다섯 번째행의 큐브 $b'c$와 여섯 번째열의 큐브 be는 커널/커널 $(b'c+b'e+c'd', be+a)$의 각 커널에 포함되며, 그들의 곱은 $(b'c)\cdot(be)=0$ 임으로, $M(b'c,be)=*$가 된다.

확장된 코커널 큐브 행렬에서 나머지 원소의 값은 위와 유사한 방법으로 산출된다.

Table 1. Indexes of $F=bc'd'e+ab'c+ab'e+ac'd'$.

큐브	$bc'd'e$	$ab'c$	$ab'e$	$ac'd'$
인덱스	1	2	3	4

	$b'c$	$b'e$	$c'd'$	c	e	be	a	$bc'd'$	ab'
a	2	3	4	0	0	0	0	0	0
ab'	2	3	0	2	3	0	0	0	0
$c'd'$	0	0	0	0	0	1	4	0	0
e	0	0	0	0	0	0	0	1	3

Fig. 1. Co-kernel cube matrix B of $F = bc'd'e + ab'c + ab'e + ac'd'$.

	$b'c$	$b'e$	$c'd'$	c	e	be	a	$bc'd'$	ab'
a	2	3	4	0	0	0	0	0	0
ab'	0	0	0	2	3	0	0	0	0
$c'd'$	0	0	0	0	0	1	4	0	0
e	0	0	0	0	0	0	0	1	3
$b'c$	0	0	0	0	0	*	2	0	0
$b'e$	0	0	0	0	0	*	3	0	0
be	*	*	1	0	0	0	0	0	0
c	0	0	0	0	0	0	0	*	2
$bc'd'$	0	0	0	*	1	0	0	0	0

Fig. 2. Extended co-kernel cube matrix M of $F = bc'd'e + ab'c + ab'e + ac'd'$.

2.3 사각형(rectangle) 커버

정의 6: 확장된 코커널 큐브 행렬 M에서 **사각형**은 코커널 큐브 행렬의 행과 열의 부분 집합 (R, C)이며 다음 조건을 갖는다. $\alpha, \gamma \in R$ 및 $\beta, \delta \in C$에 대하여 $M(\alpha,\beta) \neq 0$이며 $\alpha \neq \gamma$이고 $\beta \neq \delta$인 경우 $M(\alpha,\beta) > 0$이고 $M(\gamma,\delta) > 0$이면 $M(\alpha,\beta) \neq M(\gamma,\delta)$. **프라임 사각형**은 다른 사각형에 포함되지 않는 사각형이다. 사각형 (R, C)의 **사각형 비용**은 행 R과 열 C에 포함되는 리터럴들의 개수로 정의한다. 인덱스의 비용은 그 인덱스에 상응하는 큐브의 리터럴 개수로 정의한다. 함수 *cost*는 사각형과 인덱스의 비용을 산출하는 함수로 정의한다.

위의 정의는 사각형 또는 프라임 사각형에 포함되는 원소들은 서로 다른 양의 정수 값과 다수의 *(don't-care)를 포함할 수 있음을 의미한다.

예 7: Fig. 2와 같은 확장된 코커널 큐브 행렬 M이 주어졌다고 하자. $(\{a\}, \{b'c\})$는 사각형이다. 그러나, $(\{a\}, \{b'c\})$은 $(\{a, be\}, \{b'c, b'e, c'd'\})$에 포함되기 때문에 프라임 사각형은 아니다. $(\{a, be\}, \{b'c, b'e, c'd'\})$을 포함하는 사각형이 없기 때문에 프라임 사각형이라 한다. $(\{a\}, \{b'c, b'e, c'd', c\})$는 $M(a, c) = 0$를 포함하기 때문에 사각형이 아니다. 사각형 $(\{a, be\}, \{b'c, b'e, c'd'\})$의 사각형 비용은 행과 열을 나타내는 리터럴의 개수 합으로 9가 된다. 이 경우 $cost(\{a, be\}, \{b'c, b'e, c'd'\}) = 9$로 표기한다. Table 1에서 인덱스 4, 즉 $ac'd'$의 비용은 리터럴의 개수로 정의하였기 때문에 3이 된다. 이 때, $cost(4) = 3$으로 표기한다.

확장된 코커널 큐브 행렬 M에서 $M(\alpha,\beta) > 0$인 모든 원소가 $\alpha \in R_k$, $\beta \in C_k$인 어떤 사각형 (R_k, C_k)에 포함된다면, 코커널 큐브 행렬 M은 사각형 집합 $\{(R_k, C_k)\}$들로 커버된다고 한다. 사각형 커버 문제는 최소 커버 비용으로 코커널 큐브 행렬 M을 커버하는 사각형 집합을 찾는 것이다. 사각형 커버 문제를 풀기 위해서 확장된 코커널 큐브 행렬에서 모든 양의 정수 원소를 커버하는 프라임 사각형들을 찾는다. 그리고, 최소 커버 비용으로 코커널 큐브 행렬을 커버하는 프라임 사각형들의 부분 집합을 산출한다. Fig. 3의 알고리즘 1은 확장된 코커널 큐브 행렬에서 프라임 사각형들을 산출한다. 이 때, 사각형 정의에 따라 프라임 사각형은 다수의 *를 포함할 수 있으나, 다수의 동일한 인덱스를 포함할 수 없다. 알고리즘 1에서 PRIME_RECT는 각 $M(\alpha,\beta) > 0$인 행 α와 β에 대하여 EXPAND

Algorithm 1: Generation of all prime rectangles.
Input: An extended Boolean matrix M.
Output: A set P of prime rectangles, $P=\{P_k\}$, where $P_k=(R_k, C_k)$, and R_k and C_k are subsets of rows and columns, respectively.
Method:

```
PRIME_RECT( M)
    P= φ;
    for α in set of rows in M do
        for β in set of columns in M do
            if (M(α, β) > 0)
            then P= P ∪ EXPAND (α, β, M);
    return( P );

EXPAND (α, β, M)
    LP= φ;
    RM= set of rows in M;
    CM= set of columns in M;
    find all subset of columns, C= { C_k }, such that
    C_k= { β } ∪ { l | M(α, l)≠0, ∀ l ∈ CM, and if M(α, l) > 0
                then M(α, l)≠M(α, c), ∀ c ∈ C_k };
    for each C_k ∈ C do
        find subset of rows, R_k, such that
        R_k= { α } ∪ { l | M(l, m)≠0, ∀ l ∈ RM, ∀ m ∈ C_k, and if M(l, m) > 0
                    then M(l, m)≠M(γ, c), ∀ γ ∈ R_k, ∀ c ∈ C_k };
        LP= LP ∪ { (R_k, C_k) };
    return( LP);
```

Fig. 3. Algorithm 1.

를 호출한다. EXPAND는 $M(\alpha,\beta) > 0$인 원소에 대하여, 행 α에서 열 β를 포함하는 열들의 부분 집합들 $\tilde{C}$를 찾는다. 다음, EXPAND는 $C_k \in \tilde{C}$에 대한 행들의 부분 집합 R_k를 찾는다. 그러면, 이 행과 열의 부분 집합이 프라임 사각형 (R_k, C_k)이다.

예 8: Fig. 2의 확장된 코커널 큐브 행렬 M에 대하여 Fig. 3의 알고리즘 1이 프라임 사각형들을 산출하는 과정을 보인다. 여기서, PRIME_RECT가 $\alpha=a$, $\beta=b'c$일 때 EXPAND를 호출한다고 가정하자. 그러면, EXPAND는 행 a에 대한 열들의 부분 집합 $\tilde{C}=\{\{b'c, b'e, c'd'\}\}$을 산출한다. 다음

$\mathcal{C}=\{\{b'c,b'e,c'd'\}\}$을 산출한다. 다음 $\mathcal{C}$에 포함되는 열들의 부분 집합, 즉 $C_1=\{b'c,b'e,c'd'\}$으로부터 for-loop에서 $R_1=\{a,be\}$을 산출한다. 따라서 $LP=(\{a,be\},\{b'c,b'e,c'd'\})$. EXPAND는 프라임 사각형 $(\{a,be\},\{b'c,b'e,c'd'\})$을 PRIME_RECT로 리턴한다. 기타 다른 프라임 사각형 역시 동일한 방법으로 산출된다.

Fig. 3의 알고리즘 1은 Table 2에 보인 4개의 프라임 사각형을 산출한다.

인덱스 집합에 포함되고, 사각형 P_i에 속하나 P_j에 속하지 않는 인덱스들에 대하여 다음의 비용 관계가 성립되면 사각형 P_i가 사각형 P_j를 **지배한다**고 정의한다.

$$cost(P_i) \le cost(P_j) + \sum_{x \in (P_i - P_j)} cost(x)$$

여기서, x는 P_i에 속하나 P_j에는 속하지 않는 인덱스다.

Table 2. Prime rectangles for $F=bc'd'e+ab'c+ab'e+ac'd'$.

프라임 사각형	인덱스	$cost(P_i)$
$P_1=(\{a,be\},\{b'c,b'e,c'd'\})$	1,2,3,4	9
$P_2=(\{ab',bc'd'\},\{c,e\})$	1,2,3	7
$P_3=(\{c'd',b'c,b'e\},\{be,a\})$	1,2,3,4	9
$P_4=(\{e,c\},\{bc'd',ab'\})$	1,2,3	7

Algorithm 2: Removal of equivalent and dominated prime rectangles.
Input: A set P of prime rectangles.
Output: A subset P' such that $P' \subseteq P$.
Method:

```
IRREDUNDANT_RECT( P)
    P' = P;
    for each prime rectangle P_i ∈ P do
        if ( P_i is equivalent to P_j ∈ P' − { P_i } ) then P' = P' − { P_i };
    P = P';
    for each prime rectangle P_i ∈ P' do
        if ( P_i is dominated by P_j ∈ P − { P_i } ) then P = P − { P_i };
    return( P );
```

Fig. 4. Algorithm 2.

정의 7: 사각형 $P_i=(R_i, C_i)$에 포함되는 인덱스들과 P_i의 비용이 사각형 $P_j=(R_j, C_j)$와 동일한 경우, 사각형 P_i와 P_j는 **합동**이라 한다. 사각형 P_j의 인덱스 집합이 사각형 P_i의

예 9: Table 2의 프라임 사각형들이 주어졌다고 하자. 프라임 사각형 P_1과 P_3는 동일한 인덱스들 1, 2, 3, 4를 포함하고 동일한 비용 $cost(P_1)=cost(P_3)=9$임으로, P_1과 P_3는 합동이다. 또, P_2와 P_4 역시 위와 동일한 조

Algorithm 3: Selection of a prime rectangle with a minimum cost.
Input: A set P of prime rectangles.
Output: A prime rectangle with a minimum cost.
Method:
SELECT(P)
$best\_cost = \infty$;
for each $P_i \in P$ **do**
let $x_1, x_2, \ldots, x_n$ be indexes which P_i does not contain;
$P_i.cost = cost(P_i) + \sum_{x_i \in \{x_1, x_2, \ldots, x_n\}} cost(x_i)$;
if ($P_i.cost < best\_cost$) **then** $best\_rect = P_i$;
else $best\_cost = P_i.cost$;
return($best\_rect$);

Fig 5. Algorithm 3.

건으로 합동이다. P_2에 포함되는 인덱스들은 역시 P_1에 포함된다. P_1에만 포함되는 인덱스 4, 즉 $ac'd'$의 비용은 $cost(4)=3$이고, $cost(P_1) < cost(P_2) + cost(4)$가 성립함으로 사각형 P_1은 사각형 P_2를 지배한다.

Fig. 3의 알고리즘 1이 수행된 후 산출된 프라임 사각형들에는 합동과 지배 관계가 존재할 수 있다. 합동인 프라임 사각형과 지배되는 프라임 사각형은 잉여 프라임 사각형임으로 프라임 사각형 집합에서 제거한다. Fig. 4의 알고리즘 2는 단순히 합동과 지배되는 프라임 사각형들을 프라임 사각형 집합에서 제거하여 전체 프라임 사각형의 개수를 줄인다. 알고리즘 2에서 첫 번째 for-loop는 합동인 프라임 사각형 중에서 하나를 제거한다. 두 번째 for-loop는 지배되는 프라임 사각형을 제거한다.

예 10: Table 2의 프라임 사각형들에 대하여 Fig. 4의 알고리즘 2를 적용하자. 그러면, 알고리즘 2의 첫 번째 for-loop는 프라임 사각형 P_3와 P_4를 제거한다. 여기서, P_3는 P_1과 합동이고 P_4는 P_2와 합동이다. 두 번째 for-loop에서는 P_1이 P_2를 지배하기 때문에 P_2를 제거한다. 따라서, 알고리즘 2는 프라임 사각형 P_1만을 리턴한다.

최소 커버 비용을 갖는 프라임 사각형들을 선택하기 위해서 본 논문에서는 그리디(greedy) 방법을 사용한다. Fig. 5의 알고리즘 3은 프라임 사각형들 중에서 그리디 방법으로 프라임 사각형 하나를 선택한다. 여기서 프라임 사각형 선택 방법은 다음과 같다. 각 프라임 사각형에 대하여 프라임 사각형의 비용과 프라임 사각형에 포함되지 않는 인덱스들의 비용을 더하고, 그 중 가장 작은 값을 갖는 프라임 사각형을 선택한다. 이를 수식으로 표현하면 다음과 같다. $\tilde{P}$를 프라임 사각형들의 집합이라 하자.

$$\mathrm{MIN}_{P_i \in \tilde{P}} \{ cost(P_i) + \sum_{x \in (I - P_i)} cost(x) \}$$

여기서, I는 주어진 단순식으로부터 산출된 인덱스 집합이다.

예 11 : 예 10의 결과로부터 $P_1 = (\{a, be\}, \{b'c, b'e, c'd'\})$만이 산출되며 $I = \{1,2,3,4\}$ 이다. 프라임 사각형 P_1은 모든 인덱스(즉 1, 2, 3, 4)를 포함하기 때문에 알고리즘 3은 최소 비용을 갖는 프라임 사각형으로 P_1을 선택한다.

2.4 분해 알고리즘

부울 분해식 산출 알고리즘은 되부름 형태

(recursive procedure)로 Fig. 6의 알고리즘 4와 같다. 알고리즘 4는 최소 비용을 갖는 프라임 사각형 하나를 선택하고, 이 프라임 사각형으로부터 분해식 $F=QD+R$을 산출한다. 여기서, Q와 D는 프라임 사각형의 행들과 열들에 상응하는 단순식이다. R은 $R=F-QD$로부터 산출한다. 다시, Q, D와 R에 대하여 각각 프라임 사각형을 산출하고 이로부터 분해식을 반복해서 산출한다.

Algorithm 4: Boolean factorization.
Input: A given expression F.
Output: A Boolean factored form of F.
Method:

```
FACTOR( F )
    if (there is not any common literal to some cubes in F) then write( F);
                                                        else return;
    Construct an extended co-kernel cube matrix M;
    P=PRIME_RECT( M );              /* Algorithm 1 */
    P=IRREDUNDANT_RECT( P );        /* Algorithm 2 */
    best_rect=SELECT( P );          /* Algorithm 3 */
    Q=expression corresponding to rows of best_rect ;
    FACTOR( Q );
    D=expression corresponding to columns of best_rect ;
    FACTOR( D );
    R= F- QD ;
    if ( R≠φ)   then write("+");
                else FACTOR( R );
```

Fig. 6. Algorithm 4.

예 12: Fig. 2의 확장된 코커널 큐브 행렬 M에 대하여 알고리즘 2와 3을 수행하면 프라임 사각형 ({ a, be } , { $b'c, b'e, c'd'$ })을 얻는다. 그러면, $Q=a+be$, $D=b'c+b'e+c'd'$이고 $R=\phi$가 된다. 다시, $D=b'c+b'e+c'd'$에 대하여 $b'(c+e)+c'd'$의 분해식이 산출된다. 그러면,최종적으로 $F=(a+be)(b'(c+e)+c'd')$인 부울 분해식이 산출되며, 이 때 리터럴 개수는 8이 된다. 만일 F에 대하여 Fig. 1의 코커널 큐브 B를 이용하였다면 사각형 커버는
({ a} , { $b'c, b'e, c'd'$ })와
({ $c'd'$ } , { be })가 된다. 따라서, 대수 분해식 $F=a(b'c+b'e+c'd')+bc'd'e$가 산출되고, 반복해서 $b'c+b'e+c'd'$에 대한 분해식을 산출하여,결국 $F=a(b'(c+e)+c'd')+bc'd'e$인 분해식을 산출한다. 이 경우 리터럴 개수는 10이 된다.

2.5 실험 결과

제안한 방법은 Sun Sparc 20 컴퓨터에서 C 언어로 프로그램하였다. 코커널/커널들을 산출하기 위해서 Brayton 등이 제시한 커널 산출 알고리즘[3]을 이용하였다. 본 논문에서 제안한 방법을 PLA 형의 여러 MCNC (Microelectronics Center of North Carolina) 벤치마크 회로(benchmark circuit)에 대하여 테스트하였고, 그 결과를 SIS 1.2[5]의 출력과 비교하였다. 널리 사용

되는 SIS 1.2는 커널 기반 대수 분해 방법으로 대수 분해식을 산출한다. 본 실험에서 분해식의 리터럴 개수만을 비교할 것이기 때문에, SIS 1.2에서는 분해식 산출 명령 "factor -g *"를 각 벤치마크 회로에 적용하였다.

제안한 방법과 SIS 1.2의 결과를 Table 3에 나열하였다. 본 실험 동안 수행 시간이 3분을 초과하면 수행을 종료하도록 하였다. Table 3에서 제외된 벤치마크 회로는 대수 분해식과 부울 분해식 산출에 3분 이상 수행 시간이 요구되는 것들이다.

Table 3에서 첫 번째 열은 벤치마크 회로의 이름이다. 입력수와 출력수로 표기된 행들은 각각 벤치마크 회로의 입력과 출력의 개수를 나타낸 것이다. SIS 1.2로 표시된 열은 SIS 1.2의 수행 결과로 리터럴 개수와 수행 시간을 초 단위로 표시한 것이다. 오른쪽의 두 열들은 본 논문에서 제안한 방법으로 실험한 결과들로써 리터럴 개수와 수행 시간을 나타낸 것이다.

제안한 방법은 예상한 바와 같이 리터럴 개수에 있어서는 실험한 모든 벤치마크 회로에 대하여 SIS 1.2보다 적거나 같은 결과를 산출하였다. 특히 squar5, tcon, cmb의 경우 $x'y+xz+yz$와 같은 형태의 단순식을 포함하기 때문에 SIS 1.2에 비해 30%에 달하는 리터럴 개수 감소 효과를 얻었다.

여기서, $x'y+xz+yz$와 같은 형태의 단순식은 $(x'+z)(x+y)$의 부울 분해식으로 표현된다. 실험 결과에서 대수 분해식과 부울 분해식의 리터럴 개수가 동일하게 산출되었는데, 이러한 현상은 주어진 회로 자체가 대수 분해식으로 최적화되었음을 의미하는 것으로 해석할 수 있다.

전체 벤치마크 회로에 대하여 제안한 방법은 SIS 1.2와 비교하면 평균적으로 3.6%의 리터럴 개수를 줄일 수 있었다. 수행 시간을 SIS 1.2와 비교하면 실험 대상 회로 중 60%에서 수행 시간이 증가된 반면, 40%의 회로에서 수행 시간이 감소된 결과를 얻었다.

이러한 결과를 Fig. 1과 2를 비교하여 설명하면 다음과 같다. Fig. 2에서 사각형 ($\{a, be\}$, $\{b'c, b'e, c'd'\}$)는 모든 인덱스를 포함하는 반면에, Fig. 1의 모든 인덱스를 포함하기 위해서는 2개의 사각형 ($\{a\}$, $\{b'c, b'e, c'd'\}$)와 ($\{c'd'\}$, $\{be\}$)이 요구된다. 즉 확장된 코커널 큐

Table 3. Comparison with SIS.

회로	입력수	출력수	SIS 1.2		제안 방법	
			리터럴 수	시간(초)	리터럴 수	시간(초)
5xp1	7	10	161	0.5	161	1.49
inc	7	9	246	1.0	243	8.27
bw	5	28	290	0.5	290	0.24
misex1	8	7	86	0.2	86	0.10
misex2	25	18	164	0.2	163	0.07
rd53	5	3	71	0.3	69	1.93
squar5	5	8	127	0.5	91	3.28
con1	7	2	19	0.1	19	0.01
o64	130	1	130	0.2	130	0.01
vg2	25	8	246	0.7	246	19.68
xor5	5	1	28	0.2	28	1.73
f51m	8	8	168	0.6	164	3.02
z4m1	7	4	69	0.4	67	5.47
mux	5	1	79	0.6	71	16.57
sct	19	15	138	0.2	137	0.14
tcon	17	16	48	0.1	40	0.05
majority	5	1	10	0.1	10	0.02
decod	5	16	80	0.1	80	0.01
cmb	16	4	98	0.2	82	0.85

브 행렬에서의 사각형은 확장 전의 코커널 큐브 행렬에서의 사각형을 커버하며, 사각형 개수가 줄어들기 때문에 대수 부울 분해식 산출 수행 시간이 주는 경우가 얻어진다.

Table 4는 최근에 발표된 Stanion[7]의 부울 분해식 결과와 제안한 방법으로 산출된 결과를 리터럴 개수 측면에서 비교한 것이다. Stanion 방법으로 산출된 부울 분해식은 때때로 대수 분해식 보다 리터럴 개수가 늘어난 결과를 초래한다. 그러나 제안한 방법으로 산출된 부울 분해식은 대수 분해식보다 리터럴 개수가 늘어나는 경우는 발생하지 않는다.

Table 4. Comparison with Stanion's

회로	Stanion 방법	제안 방법
bw	306	290
misex1	88	86
misex2	163	163
vg2	254	246
f51m	174	164
z4m1	67	67

3. 결 론

본 논문은 부울 분해식을 산출할 수 있도록 코커널 큐브 행렬을 확장하는 방법과 확장된 코커널 큐브 행렬의 커버 방법을 제안하였다.

확장된 코커널 큐브 행렬은 코커널/커널들과 커널/커널들을 동시에 이용하여 만든다. 확장된 코커널 큐브 행렬에서 코커널/커널들은 대수 분해식을 산출하는 데 이용되며, 커널/커널들은 부울 분해식을 산출하는 데 이용한다.

코커널/커널들과 커널/커널들이 동시에 분해식 산출에 고려되기 때문에 대수 분해식과 부울 분해식 중에서 보다 적은 리터럴 개수를 갖는 분해식을 산출하는 기반을 제공한다. 커널 기반 대수 분해 산출 방법과 비교했을 때 제안한 방법은 커널/커널들을 산출해야 하기 때문에 수행 시간이 상대적으로 길어지는 반면에 리터럴 개수를 줄일 수 있었다.

따라서, 제안한 방법은 분해식을 산출하는 데 수행 시간보다는 리터럴 개수를 줄이는 것이 중요할 때 적합하다. SIS 1.2의 결과와 비교했을 때, 제시한 방법으로 산출된 분해식을 그대로 CMOS 회로로 구현한다면 트랜지스터 개수를 줄이는 효과를 갖게 되며, 결국 회로의 칩 면적을 줄일 수 있게 된다.

실험 결과에서 보였듯이 대수 분해식과 부울 분해식의 리터럴 개수가 동일한 경우가 많았다.

즉, 이러한 경우 커널/커널이 불필요한 데, 이처럼 커널/커널 산출 여부를 결정할 수 있는 판단 방법이 향후 연구로 요구된다.

참고문헌

1) E. Lawler. An approach to multilevel Boolean minimization. *Journal of ACM*.11(3), 283-295 (1964)
2) R. K. Brayton and C. McMullen. The decomposition and factorization of Boolean expressions. *Proc. ISCAS*, 49-54 (1982)
3) R. K. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, and A. R. Wang. MIS: A multiple-level logic optimization system. *IEEE Trans. CAD*, 6(6), 1062-1081 (1987)
4) R. K. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, and A. Wang. Multi-level logic optimization and the rectangle covering problem. *Proc. ICCAD*, 66-69 (1987)
5) E. M. Sentovich, K. J. Singh, C. Moon, H. Savoj, R. K. Brayton, R. K., and A. Sangiovanni-Vincentelli. Sequential circuit design using synthesis and optimization. *Proc. ICCD*, 328-333 (1992)
6) S. Liao, S. Devadas, and A. Ghosh. Boolean factoring using multiple-valued minimization. *Proc. ICCAD*, 606-611 (1993)
7) T. Stanion and C. Sechen. Boolean division and factorization using binary decision diagrams. *IEEE Trans. CAD*, 13(9), 1179- 1184 (1994)

(1999년 12월20일 접수, 2000년 2월25일 채택)