Journal of the Korea Academia-Industrial cooperation Society
Vol. 15, No. 9 pp. 5744-5749, 2014

http://dx.doi.org/10.5762/KAIS.2014.15.9.5744
ISSN 1975-4701 / eISSN 2288-4688

배타 논리합 원리를 이용한 다출력 논리회로 간략화

권오형[1*]
[1]한서대학교 컴퓨터공학과

Multioutput Logic Simplication Using an Exclusive-OR Logic Synthesis Principle

Oh-Hyeong Kwon[1*]
[1]**Dept. of Computer Engineering, Hanseo University**

요 약 다출력 논리식에서 공통식을 추출하는 것은 매우 중요한 기술이다. 본 논문에서는 배타 논리합 식 산출 원리를 이용해서 공통식을 추출하는 새로운 방법을 제안하였다. 산출된 논리식은 AND, OR, NOT 연산자만을 이용해서 전체 논리식을 표현하도록 고안하였다. 공통식 산출의 수행 시간과 리터럴 개수를 줄이기 위해서 선험 방법을 제안하였다. 실험 결과 제안한 방법이 기존의 방법들보다 리터럴 개수를 줄일 수 있음을 보였다.

Abstract An extraction technique for a common logic expression is an extremely important part of multiple-output logic synthesis. This paper presents a new Boolean extraction technique using an exclusive-OR logic synthesis principle. The logic circuits produced only have AND, OR and NOT gates. Heuristic methods can also be applied to reduce the execution time and the number of literals. The experimental results showed improvements in the literal counts over the previous methods.

Key Words : extraction, support variable, exclusive-OR logic synthesis principle

1. 서론

여러 개의 출력단을 갖는 논리회로에는 동일한 회로가 반복 사용되는 경우가 종종 있다. 논리회로 최적화 방법 중의 하나가 반복 사용된 회로를 공유하도록 하여 전체 논리회로를 최적화하는 것이다. 이러한 최적화 방법을 공통식 추출 방법(extraction) 이라 한다. 공통식 추출 방법에 사용된 많은 방법들이 논리회로를 논리식으로 나타내고, 여러 논리식을 동시에 나누는 제수(divisor)를 찾는 것이다. 동시에 나누는 제수가 있는 경우, 제수에 해당하는 부분을 변수로 치환하여 전체 논리식들을 최적화한다.

나눗셈에 의한 기존의 공통식 산출 방법들을 정리하면 다음과 같다. Brayton 등은 커널(kernel) 개념을 도입하여 다항 큐브 제수를 찾는 방법과 커널 교집합을 적용하여 공통식을 찾는 방법을 제안하였다[1-3]. Rajski와 Vasudevamurthy는 단항 큐브와 보수(complement)를 함께 고려하여 공통식을 찾는 방법을 제안하였다[4]. Wu와 Zhu는 이진 결정 그래프 (Binary Decision Diagram, BDD)를 이용한 논리식 간략화 방법을 제안하였다[5]. Kwon은 커널-커널 쌍을 이용한 방법, 2-큐브 몫 행렬을 이용한 방법 및 서포트 변수를 추가해서 공통식을 산출하는 방법을 제시하였다[6-8]. 최근에는 양자컴퓨터의 기반이 되는 리버서블로직(reversible logic)을 이용한 회로설계자동화 연구가 진행되고 있다[9]. 이러한 방법들 모두 선험 방식(heuristic method)에 의한 기법으로 여전히 간략화를 하지 못하는 문제점을 갖고 있다. 따라서, 본 논문에서는 기존의 나눗셈에 의한 공통식 산출이 아니라, 각 출력단의 논리식을 변형하고, 변형된 후의 논리식에서 공통식이 있는지 찾는 방법을 제시하였다.

*Corresponding Author : Oh-Hyeong Kwon(Hanseo Univ.)
Tel: +82-41-660-1365 email: ohkwon@hanseo.ac.kr
Received August 7, 2014 Revised (1st August 29, 2014, 2nd September 10, 2014) Accepted September 11, 2014

2. 다출력 논리식에서 공통식 산출

본 장에서는 제안하는 방법을 기술하는데 필요한 용어 정의를 제시한 후, 본 논문에서 제안하는 방법에 대하여 기술한다. 다음 정의들은 [1-3]에서 기술한 내용을 옮긴 것이다.

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 서포트 변수 집합을 $sup(F)$로 표기하면, $sup(F) = \{x|$큐브 $C \in F$에 대하여, $x \in C$ 또는 $x' \in C\}$.

지금부터 서포트 변수를 간략히 서포트라 부른다.

예 2: 단순식 $F=a+bc'$의 서포트는 $sup(F)$ = { 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 $F=abfg+a'cdef+cdfg$ 에 대하여, F의 커널과 코커널의 예를 보이기 위해서 주어진 논리식을 다음과 같이 표현하자. 즉, $F=abfg+cdf(a'e+g)$로 표현된 경우, $a'e+g$는 커널이 되며, 이 때 코커널은 cdf이다.

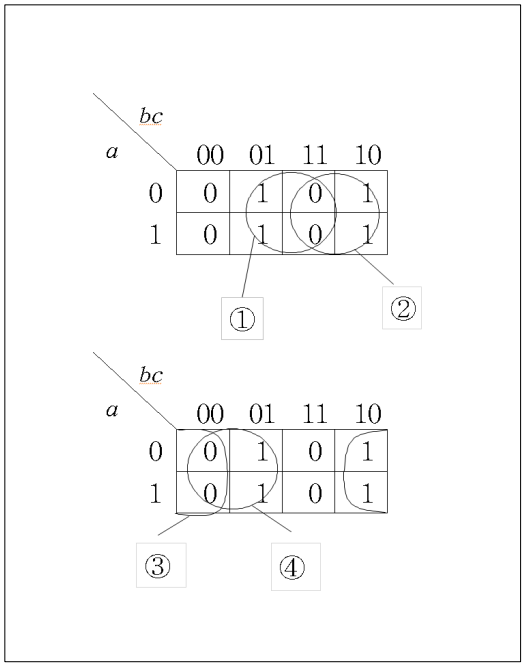
정의 4: 배타 논리합 식(Exclusive sum-of-products (ESOP) form)은 논리곱 항들을 배타 논리합 연산자(이하 XOR로 표기)로 표현한 식이다. 본 논문에서 논리식을 표현할 때, 배타 논리합 연산자에 대한 기호로는 $\oplus$를 사용한다.

예 4: $F=ab\oplus cd$ 은 논리곱 항 ab와 cd를 XOR 연산자로 표현한 배타 논리합 식이다.

2.1 배타 논리합 적용 논리식 산출

카르노 맵을 이용해서 배타 논리합 식을 산출하기 위해서는 논리값이 1인 부분은 홀수번 묶음에 속하고, 논리값이 0인 부분은 짝수번 묶음에 속하도록 묶음을 찾아 논리곱 항을 산출하고, 이 논리곱 항들을 XOR 연산자를 이용해서 표현하면 된다. 이 때, 배타 논리합 식은 다수 개가 산출될 수 있으나, 일반적으로 단순식에 비해 적은 수의 리터럴을 갖는 논리식이 산출될 수 있다.

예 5: Fig. 1과 같이 3개의 입력 a, b, c와 출력 F에 대한 카르노 맵이 주어졌다고 가정하자. Fig. 1의 위와 아래는 카르노 맵에서 서로 다른 배타 논리합 식이 산출되는 예를 보인 것이다. Fig. 1의 묶음 ①과 묶음 ②에 의해서 논리값 1에 해당하는 부분은 홀수번인 한 번씩 묶음에 속하고, 논리값 0에 해당하는 부분은 묶음에 속하지 않는 부분과 두 번의 묶음에 속하게 되어 모두 짝수번 묶음에 속하게 된다. 따라서 전체 논리식은 묶음 ①과 ②에 각각 대응되는 c와 b에 XOR 연산자를 적용하여 $F=b\oplus c$가 된다. 마찬가지로 Fig. 1의 아래 카르노 맵에서 보인 바와 같이 묶음 ③, ④가 산출되어 배타 논리합 식 $F=b'\oplus c'$이 산출될 수 있다.



[Fig. 1] Covers for ESOP Expressions

2.2 다출력 논리회로 간략화

이전 절에서 언급한 바와 같이 배타 논리합 식의 경우 단순식에 비해 리터럴의 개수를 줄일 수 있는 장점이 있지만, XOR 게이트를 논리회로에 사용해야 점과 다수 개의 배타 논리합 식이 산출될 수 있다는 단점이 있다. 본 논문에서는 논리회로에 XOR 게이트를 사용하지 않고, 단지 AND, OR, NOT 게이트만으로 구성된 논리회로를 산출하는 것을 목표로 하며, 제안하는 방법은 다음과 같다.

배타 논리합 식에서 얻어진 논리곱 항들은 그대로 이용하고, XOR 연산자 대신 OR 연산자를 사용한다. 그러면 논리곱 항은 논리값 0 부분을 포함하게 되어, 이에 대한 보정이 필요하다. 이 때, 이 보정 부분에 해당하는 논리회로 또는 논리식이 다른 출력 부분에서 중복 사용되는 경우, 중복 사용을 피하고 단 한 번만 사용하게 되면 전체 논리회로 또는 논리식이 간략화 될 수 있다. 이점이 본 논문의 핵심이며, 이를 다시 원리 1과 2로 표현하였다. 즉, 원리1과 원리2를 주어진 다출력 논리회로 또는 논리식에 적용하면 전체 논리식에 사용된 리터럴 개수를 줄일 수 있다.

원리1: AND와 XOR 연산자를 이용한 배타 논리합 식이 (식1)과 같을 때, AND와 OR 연산자만을 이용한 논리식은 (식2)로 표현 가능하다. 이 때, $(z_{c_1} z_{c_2} \cdots z_{c_l})'$을 본 논문에서는 보정식이라 부른다.

$$x_{a_1} x_{a_2} \cdots x_{a_m} \oplus y_{b_1} y_{b_2} \cdots y_{b_n} \quad \text{(식1)}$$

$$= (x_{a_1} x_{a_2} \cdots x_{a_m} + y_{b_1} y_{b_2} \cdots y_{b_n})(z_{c_1} z_{c_2} \cdots z_{c_l})' \quad \text{(식2)}$$

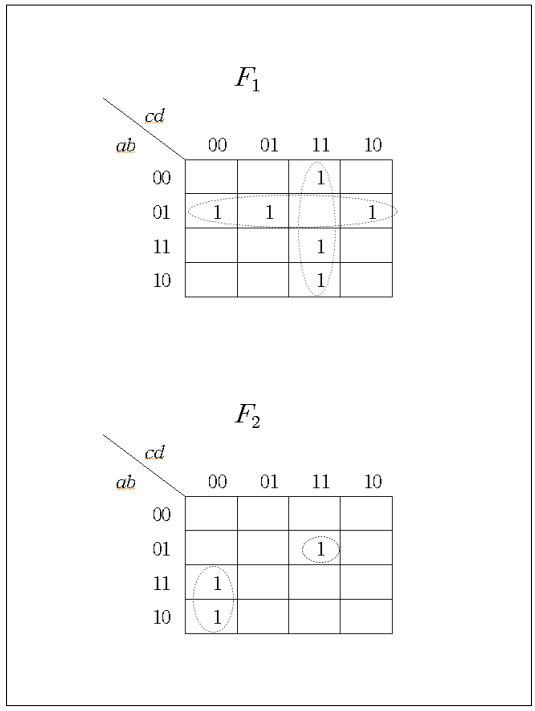
그리고,

$$z_{c_i} = \begin{cases} \varnothing & \text{if } x_{a_p} = y_{b_q} \\ x_{a_p} y_{b_q} & \text{if } sup(x_{a_p}) = sup(y_{b_q}) \end{cases}$$

여기서, $1 \le i \le l$, $1 \le p \le m$, $1 \le q \le n$.

원리2: 원리1의 (식2)가 어떤 출력, 예로 F_r에서 산출된 식이고, (식2)의 $z_{c_1} z_{c_2} \cdots z_{c_l}$가 또 다른 출력 F_s의 원소, 즉 $z_{c_1} z_{c_2} \cdots z_{c_l} \in F_s, (r \neq s)$가 된다면, (식2)는 (식3)과 같이 표현된다. (식3)에서 $F_s = F_s \setminus G$ 의 $\setminus$ 기호는 집합에서 원소를 제거하기 위한 연산자(difference)를 나타낸다.

$$\begin{cases} F_r = (x_{a_1} x_{a_2} \cdots x_{a_m} + y_{b_1} y_{b_2} \cdots y_{b_n})\, G' \\ F_s = F_s \setminus \{z_{c_1} z_{c_2} \cdots z_{c_l}\} \cup \{G\} \\ G = z_{c_1} z_{c_2} \cdots z_{c_l} \end{cases} \quad \text{(식3)}$$



[Fig. 2] Maps of F_1, F_2 with respect to Ex. 6

예 6: Fig. 2와 같이 F_1과 F_2에 대한 카르노 맵이 주어졌다고 가정하자. 그러면 F_1에 대한 배타 논리합 식은 $F_1 = a'b \oplus cd$ 가 된다. 이 논리식의 항들을 그대로 사용하고 XOR 연산자 대신 OR 연산자를 사용하면 $F_1 = (a'b + cd)(a'bcd)'$ 이 된다. 이 때, $a'bcd$가 보정식이 된다. 또 F_2에 대한 단순식은 $F_2 = ac'd' + a'bcd$ 가 된다. 그러면 F_1과 F_2에서 공통의 논리항 $a'bcd$가 산출된다. 이 공통항을 새로운 변수로 치환하면 간략화된 다음의 논리식들을 산출하게 된다.

$$F_1 = (a'b + cd)\,G'$$
$$F_2 = ac'd' + G$$
$$G = a'bcd$$

2.3 선험 방법과 알고리즘

최적화를 위한 입력 자료가 다출력, 다단 논리회로라고 하자. 그러면, 주어진 논리회로는 다수 개의 노드들로 구성되며, 각 노드는 단순식의 논리식으로 표현된다. 이 때, 이 전 절에서 언급한 방식을 적용하기 위해 즉, 배타 논리합 식 산출 원리를 적용한 논리식을 산출하고, 보정식이 다른 출력의 논리식과 공통 부분이 있는지 전수 조사를 하는 것은 장시간의 수행시간을 요구하기 때문에 실용성이 없다. 따라서, 본 논문에서는 배타 논리합 식 산출과 보정식의 공통 부분 조사를 위해 커널 집합을 이용한 선험 방법을 제안한다.

제안하는 방법은 동일한 서포트 입력을 갖는 2개의 노드를 선택한다. 즉, 2개의 노드를 N_i, N_j, $(i \neq j)$, 라고 하자. 그러면, N_i 노드에서 커널을 구하고, 이 N_i 노드의 커널과 N_j 노드의 보정식에 공통식이 있는지 조사해서, 공통식이 있는 경우 N_j 노드의 논리식을 간략화 한다. 이 때, 2개의 노드를 선택하는 기준은 그리디(greedy) 방식의 다음 선험기준1과 2를 적용하였다.

선험기준1: 노드들 중에서 동일한 서포트의 수가 가장 많은 2개를 선택한다.

선험기준2: 만일 선험기준1로부터 다수 개의 동일 후보 노드들이 있을 경우, 커널의 수가 많은 것을 선택한다.

주어진 논리회로 또는 논리식들에 선험기준 1과 2를 적용해서 논리회로 또는 논리식을 간략화하는 알고리즘은 Fig. 3 같다. Step 1에서는 주어진 다단 논리식의 각 노드에 대한 서포트를 산출한다. Step 2에서는 각 노드에 대하여 커널 집합을 산출하고, 각 커널에 대한 서포트를 산출한다. 다음 Step 3에서 서로 다른 2개의 노드를 선택해서, 하나의 노드에서는 Step 1에서 산출한 서포트를 다른 노드에서는 Step 2에서 산출한 커널의 서포트의 공통수가 많은 것부터 적은 개수 순서로 정렬을 한다. 즉, 선험기준1과 2를 적용한 정렬 순서를 산출한다. Step 4에서 Step 3에서 산출한 정렬 순서에 따라 2개의 노드를 선택하고, 이 2개의 노드에 원리 2, 즉 배타합 논리식 산출 원리를 적용하여 공통식을 산출한다. 예 7에 알고리즘의 수행과정에 따른 동작 예를 보인다.

Step 1: Calculate $sup(F_i)$ for each F_i with respect to node N_i;
Step 2: Calculate $sup(k_i^j)$ for each kernel $k_i^j \in K_i$, where K_i is the set of kernel of F_i;
Step 3: Sort (F_i, k_j^l) pairs in decending order of by $\|sup(F_i) \cap sup(k_j^l)\|$, $(i \neq j)$;
/* $\|G\|$ means the number of elements in G */
Step 4: Apply the synthesis principle according to the order of results in Step 3;

[Fig. 3] Proposed Algorithm

예 7: 4개의 노드에 대한 논리식이 다음과 같이 주어졌다고 가정하자. 즉, $F_1 = a'b'c' + a'bc$, $F_2 = a'b'c$, $F_3 = abd' + a'bd$, $F_4 = abd$에 대하여 위의 알고리즘을 적용해서 논리식을 간략화하자. Step 1에서 각 논리식의 서포트 집합을 다음과 같이 산출한다. $sup(F_1) = \{a, b, c\}$, $sup(F_2) = \{a, b, c\}$, $sup(F_3) = \{a, b, d\}$, $sup(F_4) = \{a, b, d\}$. 다음 Step 2에서 F_1의 커널 집합은 $K_1 = \{b'c' + bc\}$이고, 커널은 $k_1^1 = b'c' + bc$ 이며, 서포트는 $sup(k_1^1) = \{b, c\}$ 가 산출된다. 또 F_3의 커널 집합은 $K_3 = \{ad' + a'd\}$이고, 커널은 $k_3^1 = ad' + a'd$ 이며 이 커널의 서포트 $sup(k_3^1) = \{a, d\}$ 가 된다. 그리고, $K_2 = K_4 = \phi$ 이다. 다음, Step 3에서 (F_i, k_j^l)로 짝을 지어서 공통 서포트 개수가 많은 것부터 차례로 정렬한다. 그러면, 정렬

[Table 1] Experimental results

Circuit	# of inputs	# of outputs	SIS[3]		2-cube[11]		support[12]		proposed	
			# of literals	time(s)	# of literals	time(s)	# of literals	time(s)	# of literals	time(s)
b12	15	9	124	0.1	119	0.1	118	0.1	117	0.1
rd53	5	3	77	0.2	71	0.1	71	0.2	71	0.3
rd73	7	3	176	0.4	169	0.4	172	0.5	168	0.4
rd84	8	4	243	0.2	236	0.2	234	0.3	232	0.7
con1	7	2	23	0.1	23	0.1	23	0.1	22	0.2
z4ml	7	4	70	0.2	61	0.3	63	0.2	62	0.3
cmb	16	4	70	0.1	73	0.1	70	0.1	70	0.1
vg2	25	8	107	0.1	112	0.1	105	0.2	102	0.3
decod	5	16	64	0.1	51	0.1	54	0.2	51	0.4
misex1	8	7	80	0.1	80	0.1	80	0.2	80	0.2
alu4	14	8	1755	2.0	1557	1.8	1554	2.3	1550	2.1
sao2	10	4	203	0.3	192	0.4	192	0.5	190	0.4
e64	65	65	254	0.1	254	0.1	254	0.2	254	0.3
apex6	135	99	904	0.1	902	0.1	901	0.3	901	0.4

순서는 (F_2, k_1'), (F_4, k_2')가 된다. 마지막으로 Step 4에서 예 6에서 보인 것과 같이 F_1와 k_1에 대하여 배타 논리합 원리를 적용해서 간략화하고, 또 F_3와 k_2에 대하여 간략화를 한다. 그러면 최종 결과는 다음과 같게 되며, 전체 리터럴은 16개가 된다.

$$\begin{cases} F_1 = (a'b' + a'c)F_2' \\ F_2 = a'b'c \\ F_3 = (ab + bd)F_4' \\ F_4 = abd \end{cases}$$

여기에, F_1과 F_3식에 대하여 인수분해까지 하면 다음 식들이 산출되어 리터럴 개수는 14개가 된다.

$$\begin{cases} F_1 = a'(b' + c)F_2' \\ F_2 = a'b'c \\ F_3 = b(a + d)F_4' \\ F_4 = abd \end{cases}$$

3. 실험 결과

제시한 알고리즘은 Pentium IV 1.4GHz CPU PC의 Linux 환경에서 구현 및 실험하였다. 알고리즘이 선험 방법에 기반을 두고 있기 때문에 벤치마크 회로들에 대하여 타 방법들과 초 단위의 수행시간과 리터럴 산출 개수를 대상으로 성능을 비교하였다. 비교 대상이 되는 방법으로는 나눗셈 기반의 SIS1.2, 2-큐브에 의한 공통식 산출 방법, 그리고 서포트 변수들을 이용한 공통식 산출 방법이다. 특히, SIS1.2는 1990년대 발표되었지만 지금까지 SIS1.2의 기능들이 산업계나 학계에서 활용되고 있고 많은 논문들이 연구결과들을 비교할 때 비교 대상으로 이용하고 있다. 따라서, 제안한 방법의 타당성을 보이기 위해서 SIS1.2를 비교 대상으로 선정하였다. 나머지 방법들은 최근의 연구 결과이기 때문에 비교 대상으로 선정하였다. 실험 결과는 Table 1에 정리하였다. Table 1의 첫 번째 열은 벤치마크 회로의 이름이고, 다음 2개의 열은 벤치마크 회로의 입력 수와 출력 수를 나타낸 것이다. 다음 2개의 열은 SIS1.2가 수행한 결과이고, 그 다음 2개의 열은 2-큐브 방식에 의한 결과이며 다음은 서포트 변수를 이용한 방식의 수행 결과를 보인 것이다. 마지막 2개의 열은 본 논문에서 제안한 방법으로 수행한 결과를 보인 것이다. 실험 결과를 보면 제안한 방법이 비교 대상이 되는 타 방법보다 리터럴 개수를 줄이는 효과를 보이고 있다. 수행시간 측면에서도 제안하는 방법이 타 방법들과 동일한 수행시간을 보이거나, 0.1~0.4초 정도의 시간이 늘어나는 정도였다. 본 연구는 수행시간보다는 리터럴 개수를 줄이는 것이 목표라는 점을 다시 언급한다. 특히, 논리식에서 하나의 리터럴은 MOS(Metal Oxide Silicon) 기술로 반도체를 설계할 경우 2개의 트랜지스터를 필요로 한다. 또 CPU 등의 반도체를 설계할 경우 예로 디코더와 같은 부품들이 다수 개 사용되는데 이 때, 디코더와 같은 부품의 크기를 줄이게 되면 최종적으로 CPU와 같은 반도체 회로의 크기가 줄게 된다. 이런 측면에서 제안한 방법으로 반도체 부품의 리터럴 개수를 줄인 효과는 매우 의미 있다고 하겠다.

4. 결론

본 논문은 여러 출력에서 존재할 수 있는 공통 논리식을 산출하기 위한 방법을 제안하였다. 즉, 최적화 전의 전체 논리식들에서 전역(global)의 공통식을 찾는 방법이다. 제안하는 방법은 XOR 연산자를 이용한 베타 논리합 식 산출 방법을 이용하였지만, XOR 연산자는 사용하지 않고 OR 연사자와 보정식을 사용하도록 하였다. 배타 논리합 식은 주어진 카르노 맵을 이용할 경우 다수 개가 산출될 수 있어, 최적의 결과를 찾는 수행 시간이 증가할 수 있기 때문에 본 논문에서는 선험 방식을 도입한 간략화 방법을 제안하였다. 마지막으로 제안한 방법은 리터럴 개수를 줄이는 것이 목표로 실험 결과에서 보였듯이 타 방법들에 비해 리터럴 개수를 줄이는 효과를 보였다.

References

[1] R. K. Brayton and C. McMullen, “The Decomposition and Factorization of Boolean Epressions.” Proc. ISCAS, pp. 49-54, 1982.

[2] R. K. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, and A. R. Wang, “ MIS: A Multiple-Level Logic Optimization System.” *IEEE Trans. CAD*, Vol. 6, No. 6, pp. 1062-1081, 1987.
DOI: http://dx.doi.org/10.1109/TCAD.1987.1270347

[3] 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, pp. 328-333, 1992.

[4] J. Rajski and J. Vasudevamurthy, "The testability-preserving concurrent decomposition and factorization of Boolean expressions," *IEEE Trans. CAD*, Vol. 11, No. 6, pp. 778-79, 1992.
DOI: http://dx.doi.org/10.1109/43.137523

[5] D. Wu and J. Zhu, "BDD-based Two Variable Sharing Extraction," Proc. of Asia and South Pacific Design Automation Conference(ASPDAC), pp. 1031-1034, 2005.

[6] O.-H. Kwon,, “Common Expression Extraction Using Kernel-Kernel pairs,” *Journal of the Korea Academia-industrial cooperation Society*, Vol. 12, No. 7, pp. 3251-3257, 2011.
DOI: http://dx.doi.org/10.5762/KAIS.2011.12.7.3251

[7] O.-H. Kwon,, “Common Expression Extraction Using Two-cube Qoutient Matrices,” *Journal of the Korea Academia-industrial cooperation Society*, Vol. 12, No. 8, pp. 3715-3722, 2011.
DOI: http://dx.doi.org/10.5762/KAIS.2011.12.8.3715

[8] O.-H. Kwon,, “Common Expression Extraction Using Supports in Multiple-Output Logic,” *Journal of the Korea Institure of Information Technology*, Vol. 9, No. 11, pp. 17-25, 2011.

[9] M. Morrison and N. Ranganathan, "Synthesis of Dual-Rail Adiabatic Logic for Low Power Security Applications," *IEEE Trans. CAD*, Vol. 33, No. 7, pp. 975-988, 2014.
DOI: http://dx.doi.org/10.1109/TCAD.2014.2313454

권 오 형(Oh-Hyeong Kwon) **[정회원]**

- 1999년 2월 : 포항공과대학교 컴퓨터공학과 (컴퓨터공학박사)
- 1989년 7월 ~ 1993년 2월 : 전자통신연구원 연구원
- 1999년 3월 ~ 2003년 2월 : 위덕대학교 컴퓨터공학과 교수
- 2003년 3월 ~ 현재 : 한서대학교 컴퓨터공학과 교수

<관심분야>
회로설계자동화, 논리합성