Journal of the Korea Academia-Industrial cooperation Society
Vol. 18, No. 8 pp. 361-366, 2017

https://doi.org/10.5762/KAIS.2017.18.8.361
ISSN 1975-4701 / eISSN 2288-4688

항추가 및 보정을 적용한 대입에 의한 논리식 간략화

권오형
한서대학교 항공컴퓨터전공

Logic Substitution Using Addition and Revision of Terms

Oh-Hyeong Kwon
Dept. of Aeronautic Computer Engineering, Hanseo University

요 약 2개 논리식에 대해서 어떤 논리식 F의 일부가 다른 논리식 G 전체를 포함하고 있을 때, 논리식 F의 일부분을 논리식 G로 대치한 식을 대입식이라고 한다. 논리식 사이에 대입 관계가 성립되면 전체 논리식에 사용된 리터럴 개수를 대폭 줄일 수 있는 장점이 있으나, 대입 관계가 성립하지 않는 경우 대입식으로부터 얻을 수 있는 간략화 효과가 없게 되어 상대적으로 리터럴 개수를 줄이는 효과가 줄어들게 된다. 지금까지의 연구들이 주어진 논리식들 자체에 대해서 논리식들 사이의 대입 관계를 찾고, 대입이 가능하면 대입식을 산출하기 위한 방법을 제안하였는데, 본 논문에서는 논리식들 사이에 대입식이 만들어지도록 필요한 항을 추가하고, 다시 추가된 항들에 대한 보정을 통해 대입식을 산출하는 논리합성 방법을 제안한다. 최적화하고자 하는 2개의 논리식들로부터 항추가를 위한 행렬을 만들고, 행렬에서 항이 추가 가능한 묶음 찾고 추가된 항에 대해 보정을 하여 대입식이 완성된다. 실험결과 여러 벤치마크 회로에 대하여 제안한 방법이 기존 합성도구보다 리터럴 개수를 줄일 수 있음을 보였다.

Abstract For two given logical expressions and, when expression contains the same part of the logical expression as expression, substituting for thatpart of expression is called a substituted logic expression. If asubstituted relation is established between the logical expressions, there is an advantage in that the number of literals used in the whole logical expression can be greatly reduced. However, if the substituted relation is not established, there is no simplification effect obtained from the substituted expression. Previous methods proposed a way to find substituted relations between logical expressions for the given logical expressions themselves, and to calculate substituted expressions if only substitution is possible. In this paper, a new method for performing substitution with addition and revision of logic terms is proposed in order to perform substitution, even though there is no substituted relation between two logic expressions. The proposed method is efficiently implemented using a matrix that finds terms to be added. Then, by covering the matrix thathas added terms, substituted logic expressions are found. Experiment results show that the proposed method for several benchmark circuits can reduce the number of literals, compared to existing synthesis tools.

Keywords : Boolean Algebra; Cube; Logic Synthesis; Revision; Substitution; Term Addition;

1. 서론

회로 설계 자동화의 중요한 과정 중 하나가 논리합성 단계다. 보통 논리합성 단계는 최종 회로에 종속되지 않고 독립적으로 간략화된 논리식을 산출하는 것을 목표로 한다. 논리식을 실제 회로로 구현할 경우 논리식이 간략화 될수록 실제 회로의 크기도 줄어들게 된다. 실제 회로를 고려한 간략화된 논리식 산출의 어려움 때문에 연구들은 실제 구현할 회로를 고려하지 않고 간략화된 논리식 만을 산출하기 위한 방법을 찾고 있으며, 특히 다단

*Corresponding Author : Oh-Hyeong Kwon(Hanseo Univ.)
Tel: +82-41-660-1365 email: ohkwon@hansseo.ac.kr
Received July 18, 2017 Revised (1st August 7, 2017, 2nd August 16, 2017)
Accepted August 17, 2017 Published August 31, 2017

논리식이 보다 간략화된 결과를 산출하기 때문에 연구들도 다단 논리식 산출에 중점을 두고 있다. 일반적으로 다단 논리식을 산출하기 위한 방법은 다음과 같이 크게 4가지로 구분할 수 있다. 하나의 논리식 내에서 공통으로 사용되는 인수를 찾아 논리식을 다단의 형태로 만드는 인수분해 방법, 다수 개의 논리식들에서 공통으로 사용된 항들을 찾아 공통항들이 한 번만 사용하도록 해서 논리식들을 간략화하는 공통식 추출 방법, 논리식에 내재하는 무관항(don't care term)을 찾아 논리식을 간략화하는 무관항을 이용한 간략화 방법, 마지막으로 어떤 논리식이 다른 논리식의 일부를 포함하는 경우 즉, 논리식의 일부가 다른 논리식과 동일한 경우 포함되는 부분을 논리식 명으로 대치하는 대입 방법이 있다.

본 논문은 대입 방법에 의해 간략화된 논리식을 산출하기 위한 방법을 제안한 것이다. 따라서 기존의 대입 방법들을 먼저 소개하고, 다음 기타 논리합성 연구에 대하여 소개한다. Brayton 등은 커널(kernel)을 이용한 공통 다항 큐브 제수를 찾는 방법[1,2,3] 제시하였고, 이 방법을 이용해서 대입에 의한 논리식 산출 방법을 제안하였다. 이들은 커널 개념을 이용한 논리합성 도구 MIS와 SIS를 발표하였다. SIS는 MIS의 기능에 순차회로 최적화 기능이 추가된 것이다. Chang과 Cheng은 주어진 논리회로에 여분의 연결선이나 게이트를 추가하고 고착결함(stuck-at fault) 검사 기법을 이용해서 불필요한 부분을 제거하는 간략화 방법[4]을 제안하였다. Chang과 Cheng의 실험 결과에 따르면 실험에 사용한 상당수의 벤치마크회로에 대하여 SIS보다 좋은 결과를 산출하였으나, 일부 벤치마크회로에 대하여는 SIS가 보다 좋은 결과를 산출하였다. Kwon은 부울 공리인 등멱법칙($aa=a$)과 보수법칙($aa'=\phi$)을 적용해서 부울 나눗셈을 통한 논리식 사이의 대입 관계를 찾는 방법[5]을 제시하였으며, 실험 결과 SIS 및 Chang과 Cheng 방법보다 리터럴 개수를 줄이는 효과를 얻었으나, 여전히 일부 회로에 대해서는 타 방법이 보다 좋은 결과를 산출하였다. 대입 방법 외의 논리합성을 위한 최근 연구를 소개하면 다음과 같다. 주어진 논리식에서 2개의 큐브들 사이에 부울 공리를 적용해서 공통식을 찾는 방법[6]이 제시되었다. 또한 논리식을 원하는 회로로 대응 (technology mapping) 시키는 연구로 3개 입력을 갖는 majority 함수와 보수를 이용해서 논리식을 산출하고, 산출된 논리식을 FPGA 회로로 대응 시키는 방법이 제안되었다[7]. 또한 논리식을 바로 트랜지스터 회로로 대응시키는 방법[8]이 제시되었는데, 이 연구는 논리식을 간략화하기 위한 목표보다는 주어진 논리식을 최소 개수의 트랜지스터 회로로 자동 변환되는 점에 목표를 두고 있다.

이미 언급했듯이 본 논문은 대입에 의해 논리식들을 간략화하기 위한 방법을 제안한 것으로, [5]의 후속 연구 결과를 제시한 것이다. [5]의 경우 2개의 논리식 사이에 대입 관계를 찾는 것을 목표로 하였으나. 본 논문에서는 2개의 논리식 사이에 대입 관계가 성립하지 않는 경우에도 대입 관계가 성립되도록 제안한 것이다. 즉, 주어진 논리식들에 항을 추가하고 다시 추가된 부분을 보정하는 기능을 통해 논리식들을 간략화하는 방법을 제안한 것이다. 즉, 기존의 대입 방법이 단순히 주어진 논리식들 사이에 대입이 가능한지만을 조사하고 대입이 가능한 경우에 한해 간략화를 실현했다면, 제안하는 방법은 항을 추가해서 기존의 방법들로는 대입 관계를 찾지 못한 것을 대입 관계가 가능하도록 하여 논리식을 간략화하도록 하였다. 제안하는 핵심 기술은 주어진 논리식들에 어떤 항을 추가할 것인지, 그리고 어떻게 추가된 항들을 보정할 것인지를 제시한 것이다.

논문의 구성은 다음과 같다. 2장에서 본 논문에서 제시하는 새로운 대입식 산출 방법을 소개한다. 3장에서 실험결과를 보이고, 4장에서 결론을 제시한다. 본 논문과 관련된 용어와 대수 나눗셈에 의한 대입식 산출 방법은 [1-6]에 기술되어 있는 것을 활용한다.

2. 제안한 대입식 산출 방법

주어진 논리식에 항을 추가하고 보정하는 과정을 통해 대입식을 산출하는 과정을 제시한다.

2.1 대입식 산출 행렬

2개의 논리식으로부터 다음과 같은 행렬 T를 만들고, 이를 이용해서 대입식을 산출한다. 주어진 논리식이 F와 G라고 하고, 논리식 F가 논리식 G의 대입식으로 표현되는지 검사한다고 하자. 그러면, 행렬 T의 행은 G의 각 항에 대응되고, 열들은 F의 각 항들을 G의 각 항들로 나눌 경우 산출되는 몫에 대응하도록 만든다. 그리고, 행렬의 원소는 식 1에 따라 값이 산출된다. 식 1에서 *index*( )는 주어진 논리식 F의 각 항에 대응하는 양의 정

수 값을 산출하는 함수이며, *는 무관항을 나타낸다.

$$T(g_i,k_j)=\begin{cases} index(g_ik_j) & \text{if } g_i\cdot k_j \in F \\ * & \text{if } g_i\cap k_j=\phi \end{cases} \quad \text{(식 1)}$$

예 1: 다음 2개의 논리식 F와 G에 대하여 대입식 산출 행렬을 만들어 보자.

$F=acd+abc'd+adef+ace+bc'd+bdef$

$G=ad+bd+ae$

먼저 논리식 F의 각 항에 양의 정수를 배정한 인덱스(index)를 Table 1과 같이 만든다. Table 1에서 인덱스는 논리식의 왼쪽 항부터 시작해서 오른쪽으로 이동하면서 인덱스 값을 1씩 증가하는 것으로 결정하였다. 다음 항 추가를 위한 행렬을 Table 2와 같이 만든다. Table 2는 3개의 행과 6개의 열로 구성된다. 여기서 행은 논리식 G의 각 항에 대응한 것이며, 열은 G의 각 항으로 F의 각 항을 나누었을 때 산출되는 몫에 대응되도록 만든다. 즉, G의 첫 번째 항 ad로 F의 각 항을 나누면 몫이 $\{c, bc', ef\}$가 산출되고 각각의 몫이 Table 2의 열에 대응된다. 동일한 방법으로 나머지 열들이 산출된다. Table 2의 행렬 안에 기입된 숫자는 행과 열의 논리곱에 의해 산출된 결과가 논리식 F의 항이 되면 이 항의 인덱스를 Table 1에서 찾아 입력한 것이다. 행 1, 열 1의 경우 즉, 행 ad와 열 c의 논리곱은 acd가 되고 *index*(acd)=1이 Table 1에서 산출되어 Table 2의 행 1과 열 1에 해당하는 원소에는 1이 입력된다. 나머지도 같은 방법에 의해 인덱스 값이 기입된다. 빈칸의 경우 행과 열을 곱이 F의 항과 일치하지 않는 경우를 나타낸 것이다. 참고로, Table 2의 음영을 넣은 부분이나 묶음을 나타낸 것은 예 2에서 설명하기 위한 부분으로 예 1과는 무관하다.

Table 1. Indexes for F

Term, f_i, of F	acd	$abc'd$	$abde$	ace	$bc'd$	bde
index(f_i)	1	2	3	4	5	6

Table 2. Array for Substitution

Term, g_i,of G \ quotient, k_j	c	bc'	ef	ac'	c'	df
ad	1	2	3			
bd		5	6	2	5	
ae	4					3

2.2 항 추가를 위한 묶음 산출

항추가를 위한 대입식 산출 행렬이 완성되었으면 행렬에서 논리식 대입을 위한 묶음 찾는데 이 때, 그리디(greedy) 방법을 이용한다. 묶음은 숫자들만으로 채워진 가장 큰 것을 추출하는 것부터 시작한다. 여기서 묶음의 행과 열이 반드시 인접할 필요는 없다. 본 논문에서 묶음은 (R, C)와 같이 2차원으로 표시하며, R은 행들의 집합, C는 열들의 집합을 나타내도록 한다. 예로, 묶음이 행 1, 2와 열 2, 3으로 구성된 경우 ({1, 2}, {2, 3})으로 표시한다. ({1, 2}, {2, 3})은 숫자만으로 구성된 묶음이다. 이러한 묶음에 행과 열을 하나씩 추가하면서 항추가가 가능한지 검토한다. 이 때, 추가될 항은 대입식 산출 행렬을 만드는데 이용된 2개의 논리식이 아닌 다른 논리식에 있는 항인지 조사한다. 즉, 논리식 F의 일부 항들을 논리식 G로 대치하기 위해 행렬을 만들고, 항 추가를 고려한 묶음을 산출하였는데 이 때, 추가한 항이 또 다른 논리식 H에 포함된다면 항 추가가 가능하다.

예 2: 예 1의 2개 논리식 F, G에 논리식 H를 추가하여 항 추가에 의해 대입식 산출 예를 보인다. 3개의 논리식은 다음과 같다.

$F=acd+abc'd+adef+ace+bc'd+bdef$

$G=ad+bd+ae$

$H=bcd+abc'e+aef$

논리식 F와 G에 대해서는 Table 2에 있는 대입식 산출 행렬을 이용한다. Table 2의 행렬에서 먼저 숫자들만으로 구성된 묶음을 찾으면, 행 1, 2와 열 2, 3으로 구성된 묶음 ({1, 2}, {2, 3})이 얻어진다. 이 묶음을 Table 2에 음영을 넣어 표시하였다. 이 묶음은 Table 1의 2, 3, 5. 6에 해당하는 F의 일부 항만을 포함한다. 이 묶음에 ({2}, {1}), ({3}, {2}), ({3}, {3})을 추가해서 새로운 묶음 ({1, 2, 3}, {1, 2, 3})을 산출하게 되면 이 묶음은 논리식 F의 모든 항들과 행 2와 열 1에 해당하는 항 bcd, 행 3과 열 2에 해당하는 항 $abc'e$, 그리고 행 3과 열 3에 해당하는 항 aef가 추가된다. 이렇게 F의 항들과 추가된 항들로 나타낸 묶음을 Table 2에 이중선으로 표시하였다. 그런데 추가된 항 bcd, $abc'e$, aef가 논리식 H에 포함되기 때문에 예 5의 논리식은 다음과 같이 간략화된다. Table 3은 Table 2의 빈칸들에 대응하는 논리곱들과 이 논리곱을 포함하는 H의 항을 표현한 것이다.

$F=G(c+bc'+ef)H'$

$G = ad + bd + ae$

$H = bcd + abc'e + aef$

Table 3. Added Terms

Term to be added \ Expression	H
$ac'd$	
adf	
bcd	1
bdf	
$abc'e$	2
abe	
aef	3
$ac'e$	
$adef$	

2.3 확장된 보정 방법

세 개의 논리식 F, G, H가 있다고 하자. 그리고, F에 G가 대입된 논리식을 만들 때, F에 추가할 항이 t이고 $t = s \cup \{l\}$라고 하자. 또한 $s \in H$인 경우 식 2에 의해 s를 2개의 항으로 분리한다.

$$s = tl + tl' \quad \text{(식 2)}$$

예로, $t = abc$, $s = ab$인 경우 즉 $t = \{a,b,c\}$, $s = \{a,b\}$. 그러면 $t = s \cup \{c\}$가 되고, 식 2에 따라 s는 $s = abc + abc'$가 된다. 이렇게 항분리를 한 후, 3.2절에서 서술한 바와 같은 방법으로 항추가와 보정을 통해 대입식을 산출한다.

예 3: 다음 F, G, H이 주어졌을 때 대입에 의한 논리식 간략화 과정을 보면 다음과 같다.

$F = acd + abc'd + adef + ace + bc'd + bdef$

$G = ad + bd + ae$

$H = bd + abc'e + aef$

F와 G는 예 1과 동일하고, H의 첫 번째 항만 예 2의 것과 다르다. 그러면 논리식 F와 G로부터 대입식 산출 행렬인 Table 2가 산출된다. 여기서 추가될 항을 나타낸 것이 Table 4가 된다. Table 4에서 bcd에 대해서 1^x로 표시를 하였다. 1^x의 의미는 H의 첫 번째 항을 분리하면 산출될 수 있음을 표시한 것이다. 즉, 식 2에 따라 H의 첫 번째 항인 bd는 $bd = bcd + bc'd$ 으로 변형된다. 그러면 H는 다음과 같게 된다.

$H = bcd + bc'd + abc'e + aef$

여기서 $H_1 = bcd + abc'e + aef$ 로 나타내면 다시 $H = H_1 + bc'd$ 가 된다. 따라서 F, G, H로부터 다음과 같은 대입식을 구할 수 있다.

$F = G(c + bc' + ef)H_1'$

$G = ad + bd + ae$

$H = H_1 + bc'd$

여기서, 각 논리식을 인수분해까지 하면 16개의 리터럴로 구성된 논리식들이 산출된다.

$F = G(c + bc' + ef)H_1'$

$G = (d + e)a + bd$

$H = H_1 + bc'd$

Table 4. Added Terms for Cube Split

Term to be added \ Expression	H
$ac'd$	
adf	
bcd	1^x
bdf	
$abc'e$	2
abe	
aef	3
$ac'e$	
$adef$	

2.4 대입식 산출 알고리즘

알고리즘은 주어진 논리식들로부터 항추가 및 보정을 위한 대입식 산출 행렬과 행렬에서 묶음 찾는 것으로 구성된다. 주어진 논리식 F와 G에 대해서 각 논리식을 구성하는 리터럴 집합을 조사해서 F의 리터럴 집합이 G의 리터럴 집합을 포함하는 경우 F의 일부 항들을 G로 대치하는 대입식을 항추가 및 보정 방법을 적용해서 산출한다. 전체 알고리즘은 다음과 같다. 알고리즘에서 4번 라인인 "F, G에 대한 대입식 산출 행렬 산출" 단계는 2.1절에서 서술한 내용을 표현한 것이고, 5번 라인인 "확장된 보정 방법 적용" 단계는 2.3절에서 서술한 부분을 나타낸 것이다. 라인 6의 "묶음 산출" 단계는 2.2절의 내용을 나타낸 것이다. 라인 7의 "묶음을 논리식으로 변환" 단계는 예 2에서 서술한 것이다.

Algorithm : 항추가 및 보정에 의한 대입식
입력: 주어진 논리식 집합 ***L***
출력: 대입에 의한 간략화된 논리식 집합 ***L***

```
begin
1.   for each expression F∈ L do
2.       for each expression G∈ L \ {F} do
3.       begin
4..          F, G에 대한 대입식 산출 행렬 산출;
5.           확장된 보정 방법 적용;
6.           묶음 산출;
7.           묶음을 논리식으로 변환;
       end
end
```

본 논문에서 제안하는 방법은 [5]의 후속 연구이기 때문에 여기서 [5]와의 차이점을 기술한다. [5]에서 제시한 방법은 단지 논리식 F와 G에 대해서 만일 F의 일부 항들이 G와 동일한 경우, 즉 F의 일부 항들을 G로 대치하기 위해서 F의 어떤 항들이 G와 동일한지 찾기 위한 방법을 제시한 것이다. [5]에서 제안한 방법이 F의 항들에 부울공리를 적용해서 G와 동일 여부를 찾는 것이다. 예로, F=ad+$a'bc$+bd, G=a+b가 주어진 경우 F=$(a+b)(a'c+d)$가 되도록 변형하는 과정이 [5]의 핵심이며, 최종적으로 논리식 F는 F=$G(a'c+d)$가 되는 논리식을 찾는 방법을 제시한 것이다. 반면에 본 논문에서 제시한 방법은 3개의 논리식 F, G, H에 대해서 F에 항을 추가함으로써 F의 일부 항들이 G와 동일하게 되고, 추가된 항은 H의 항들과 동일한 경우 대입에 의한 논리식을 산출하기 위한 것이다. 예 2과 예 3에 동작 과정을 보였다.

3. 실험 결과

3.3GHz i3 CPU가 장착된 Linux 운영체제의 컴퓨터에서 C언어로 제안한 방법을 구현하였다. 참고로, 제안한 방법이 선험 알고리즘이기 때문에 수행 시간을 시간 복잡도로 표현하는 대신 벤치마크 회로를 입력으로 제안한 방법과 타 방법들이 산출한 결과를 비교하는 것으로 서로의 우수성을 평가하였다. 본 실험에서는 Microelectronics Center of North Carolina (MCNC) 벤치마크 회로[9]와 IWLS2005의 벤치마크 회로[10]를 이용해서 수행 시간과 리터럴 개수를 비교하였다. 결과 비교를 위해서 SIS가 산출한 대입식 결과와 Chang과 Cheng이 제시한 방법으로 산출된 결과, 그리고 가장 최근에 발표된 Kwon의 결과를 제시하였다. 실험 결과를 정리한 것이 Table 5다. Table 5의 첫 번째 열은 벤치마크회로의 이름이고, 두 번째와 세 번째 열은 SIS의 대입식 산출 결과를 보인 것이다. 네 번째와 다섯 번째 열은 Chang과 Cheng이 제시한 여분의 연결선 추가 방법에

Table 5. Experimental Results

Circuit Name	SIS		Chang & Cheng		Kwon		Proposed method	
	# of literals	time (sec)	# of literals	time (sec)	# of literals	time (sec)	# of literals	time (sec)
alu2	446	6.2	417	2.1	367	1.9	361	8.8
alu4	829	25.4	784	5.1	775	6.2	772	30.5
apex6	807	0.3	795	1.3	795	1.1	764	2.1
apex7	278	0.1	263	0.3	263	0.2	263	1.1
dalu	2003	6.2	1768	18.7	1748	9.6	1748	19.9
des	4048	32.2	5697	65.4	4023	59.2	4023	69.9
i8	1817	4.2	915	25.3	1012	23.5	915	32.5
i9	750	0.2	729	6.2	731	4.9	725	9.9
rot	751	0.4	698	0.3	691	0.3	691	1.2
t481	2608	33.1	1958	9.5	2004	17.1	1948	36.6
C880	416	0.2	413	0.2	412	0.2	412	1.2
C1355	562	0.3	558	0.7	559	0.6	551	1.5
C1908	631	0.8	597	0.4	597	1.5	597	1.9
C2670	843	0.8	829	1.2	825	1.2	825	2.3
C5315	1961	2.4	1914	3.4	1901	3.6	1901	4.1
C6288	4212	13.4	3747	7.4	3689	14.2	3685	15.4
C7552	2522	5.2	2410	5.4	2420	5.3	2402	7.3
sum of literals	25,484	-	24,492	-	22,812	-	22,583	-

의한 대입식 산출 결과를 보인 것이다. 여섯 번째와 일곱 번째 열은 Kwon이 제시한 부울 공리를 적용한 대입식 산출 결과를 보인 것이다. 마지막 2개의 열이 본 논문에서 제시한 방법으로 산출한 대입식 결과를 정리한 것이다. Fig. 1은 Table 5의 마지막 행인 전체 리터럴 개수의 합을 그래프로 표현한 것으로 리터럴 개수가 적을 수록 우수함을 보인다. 제안한 방법으로 표시된 부분이 그래프의 가장 오른쪽 부분으로 그래프의 y 좌표가 가장 낮은 위치를 보이고 있다. Table 5와 Fig. 1을 통해 제시한 방법이 타 방법들보다 수행시간은 더 소요되었지만 리터럴 개수를 줄이는 결과를 보이고 있다. 수행시간이 늘어난 이유는 항추가 과정과 보정 과정이 추가되었기 때문이다. 참고로, 보통 설계자동화 도구는 2~3일 정도의 수행시간을 허용할 정도로 간략화를 더 중요시 한다. 어떤 부품의 경우 제안한 방법으로 리터럴 개수를 몇 개만 줄인 경우라도, 이 부품이 반복 사용된다면 전체 회로의 크기를 줄이는 효과는 클 것이다. 이런 점에서 제안하는 방법이 실제 효율성을 갖고 있다고 판단한다.

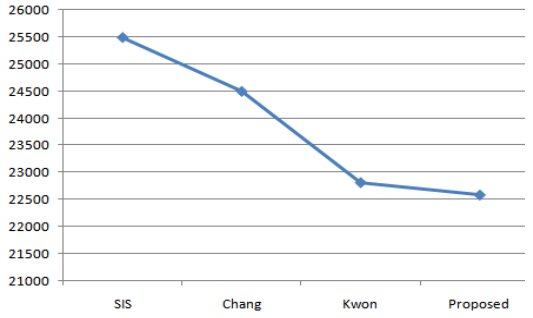

Fig. 1. Number of Literals

4. 결론

논리회로의 출력수가 3개 이상의 경우 항추가와 보정에 의해 논리회로를 최적화하는 방법을 제시하였다. 실험결과에서 보인 바와 같이 제안한 방법은 수행 시간보다는 리터럴 개수를 줄이는 것에 목표를 둔 경우에 적합한 방법이다. 수행 시간이 중요한 경우는 타 방법을 활용하고, 리터럴 개수를 줄이는 것이 중요한 경우는 제안한 방법을 사용함으로써 실용성을 높일 수 있을 것으로 본다.

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: https://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] S.-C. Chang and D. I. Cheng, "Efficient Boolean Division and Substitution Using Redundancy Addition and Removing," *IEEE Trans. CAD*, vol. 18, no. 8, pp. 1096-1106, 1999.
DOI: https://doi.org/10.1109/43.775630

[5] O.-H. Kwon, “Logic Optimization Using Boolean Resubstitution", *Journal of Korean Academia-industrial cooperation Society*, vol. 10, no. 11, pp. 3227-3233, 2009.

[6] O.-H. Kwon and B. T. Chun, “Boolean Factorization Using Two-cube Non-kernels", *Journal of Korean Academia-industrial cooperation Society*, vol. 11, no. 11, pp. 4597-4603, 2010.

[7] L Amaru, P.-E. Gaillardon, G. De Micheli., “Majority-Inverter Graph: A New Paradigm for Logic Optimization,” *IEEE Trans. CAD,* vol. 35, no. 5, pp. 806-819, 2016.
DOI: https://doi.org/10.1109/TCAD.2015.2488484

[8] D. Kagaris. "MOTO-X: A Multiple-Output Transistor-Level Synthesis CAD Tool," *IEEE Trans. CAD,* vol. 35, no. 1, pp. 114-127, 2016.
DOI: https://doi.org/10.1109/TCAD.2015.2448675

[9] S. Yang, "Logic Synthesis and Optimization Benchmarks User Guide Version 3.0," Technical Report, Microelectronics Center of North Carolina, 1991.

[10] IWLS 2005 Benchmarks, http://iwls.org/iwls2005/benchmarks.html.

권 오 형(Oh-Hyeong Kwon) **[정회원]**

- 1999년 2월 : 포항공과대학교 컴퓨터공학과 (한국학박사)
- 1990년 7월 ~ 1993년 2월 : 한국전자통신연구원 연구원
- 1999년 3월 ~ 2003년 2월 : 위덕대학교 컴퓨터공학과 교수
- 2003년 3월 ~ 현재 : 한서대학교 항공컴퓨터전공 교수

<관심분야>
설계자동화, 논리합성