2-큐브 제수와 보수에 의한 공통 논리식 산출

권 오 형[†] · 오 임 걸[††]

요 약

본 논문에서는 논리합성을 위한 공통식 추출 방법을 새롭게 제안한다. 제안하는 방법은 주어진 각 논리식들에서 2개의 큐브만으로 구성된 2-큐브 논리식 쌍을 추출한다. 2개의 큐브로 구성된 논리식 쌍들로부터 2-큐브 행렬을 만들고, 여기에 2-큐브 논리식의 보수를 추가하여 확장된 2-큐브 행렬과 압축 2-큐브 행렬을 만든다. 다음, 공통식 추출을 위해 압축 2-큐브 행렬을 분석한다. 그리디 방법(greedy method)에 의해 가장 많은 리터럴 개수를 줄일 수 있는 공통식을 선택한다. 실험결과 여러 벤치마크 회로에 대하여 제안한 방법을 논리회로 합성도구에 활용할 경우 기존 합성도구보다 리터럴 개수를 줄일 수 있음을 보였다.

키워드 : 논리합성, 전역최적화, 2-큐브 부울식

Boolean Extraction Technique Using Two-cube Divisors and Complements

Kwon, Oh-Hyeong[†] · Oh, Im-Geol[††]

ABSTRACT

This paper presents a new Boolean extraction technique for logic synthesis. This method extracts two-cube Boolean subexpression pairs from each logic expression. It begins by creating two-cube array, which is extended and compressed with complements of two-cube Boolean subexpressions. Next, the compressed two-cube array is analyzed to extract common subexpressions for several logic expressions. The method is greedy and extracts the best common subexpression. Experimental results show the improvements in the literal counts over well-known logic synthesis tools for some benchmark circuits.

Keyword : Logic Systhesis, Global Optimization, Two-cube Boolean Expression.

1. 서 론

논리합성은 상위 계층 합성 결과를 하위 계층으로 변환하기 위한 중간 단계로 논리 합성의 결과에 따라 최종 산출되는 회로 크기가 달라질 수 있다. 일반적으로 논리식이 간략화 될수록 칩의 면적도 작아지기 때문에 간략화된 논리식을 산출하기 위한 수많은 연구가 진행되고 있다. 논리식에 대한 최적화 방법으로 다단 논리회로(multi-level logic circuit) 합성 설계 방법이 널리 사용되고 있다. 다단 논리회로 최적화는 국부 최적화(local optimization)와 전역 최적화(global optimization)로 개발되어 왔다. 국부 최적화는 전체 부울 네트워크(Boolean network)의 형태에 영향을 주지 않고 단지 일부 논리식만의 최적화를 목적으로 한다. 반면에, 전역 최적화는 주어진 부울 네트워크의 변형까지 고려하면서 최적화를 수행하는 것을 목적으로 한다. 전역 최적화 방법 중의 하나인 공통식 추출(extraction) 기법은 주어진 여러 논리식 (multi-output Boolean expression)들을 리터럴 개수가 보다 적은 논리식들의 표현으로 변형하는 것이다. 이러한 공통식 추출 방법은 여러 논리식에서 공통으로 사용된 공통식을 찾고, 이 공통식을 새로운 변수로 대치하여 최적의 논리식들로 변형하는 것이다. Brayton과 McMullen은 커널(kernel)을 이용한 공통 다항 큐브 제수를 찾는 방법[1]을 제시하였고, 후에 Brayton, Rudell, Sangiovanni-Vincentelli와 Wang은 커널의 부분집합을 이용한 다항 큐브 제수를 찾는 알고리즘과 조합회로를 위한 논리합성 도구 MIS[2]를 발표하였다. Sentovich 등은 MIS에 순차회로 최적화를 위한 기능을 추가하여 SIS[3]를 만들었다. 이들의 알고리즘은 부울 공리인 등멱법칙 ($a\,a = a$)과 보수법칙($a\,a' = 0$)을 활용하지 않고, 단순히 대수적인 방법으로 공통식을 찾는다. 수행 속도는 부울 방법에 의한 공통식 산출보다 대체로 빠르나, 때때로 최적화된 결과를 산출할 수 없게 된다. Hsu와 Shen은 부울공리를 활용한 coalgebraic division이라는 나눗셈 방법[4]을 고안하여, 논리식들 사이에 coalgebraic division을 통해 전체 회로의 리터럴 개수를 줄이는데 활용하였다. 최근에는 주어진 논리회로를 Binary-Decision Diagram

† 정 회 원 : 한서대학교 인터넷공학과 부교수
†† 종신회원 : 한서대학교 인터넷공학과 부교수
논문접수 : 2007년 5월 3일, 심사완료 : 2008년 1월 10일

DOI: 10.3745/KIPSTA.2008.15-A.1.9

(BDD)으로 표현하고 BDD로부터 공통식을 찾고자 하는 연구들이 있었다. Yang과 Ciesielski는 XOR 게이트를 포함하는 논리식의 최적화를 위해 BDD를 이용한 BDS 방법[5]을 제안하였다. Wu와 Zhu는 BDS 방법에서 영감을 얻어 Folded BDD(FBDD)[6]를 발표하였다. 한편, BDD 자체에 대한 연구도 진행되고 있으며, 가장 최근에는 BDD를 확장한 Edge-Valued Multi-valued Decision Diagram (EVMDD)을 이용한 논리식 표현에 대한 연구[7]가 발표되었다. 발표된 논리합성 연구들 중에서 본 논문과 유사한 연구들을 정리하면 다음과 같다. Rajski와 Vasudevamurthy는 다변수 출력을 갖는 논리회로에서 단지 2개의 리터럴만을 갖는 단항 큐브와 2개의 큐브로 구성된 공통식을 추출하는 방법[8]을 제안하였다. 이들이 제시한 방법을 적용할 경우 일부 벤치마크회로 최적화에서 커널 기반 방법[1-3]보다 리터럴 개수를 줄이는 결과를 얻었다. 그러나, 부울 공리를 적용하지 않고 단지 대수 나눗셈에 의한 공통식 산출이라는 점에서 본 연구와 구별 된다. Wu와 Zhu는 단지 2개의 변수만으로 구성된 공통식들을 추출하는 방법[9]을 제시하였다. 이들은 BDD를 이용하여 논리합성 도구를 개발하였으며, 이 도구에 2개의 변수만으로 구성된 공통식 추출 알고리즘을 포함시켰다. 이들의 방법은 여러 논리식들에서 발견될 수 있는 공통식 개수를 줄여 수행 속도를 향상하는 효과를 얻었다. 이들이 제시한 방법은 2개의 변수로만 구성된 공통식을 추출하는 방법이나, 본 논문에서는 2개의 큐브로 구성된 여러 개의 2-큐브 논리식들을 찾고, 이 중에서 전체 리터럴 개수를 최소화 할 수 있는 공통식을 선택하는 방법이라는 점에서 구분된다. 한편, [10]은 2-큐브 논리식 쌍을 이용한 연구로 본 논문의 초기 연구 결과이다. [10]에서 제시한 방법은 단지 2-큐브 논리식 쌍만을 이용하여 공통식을 산출하였으나, 본 논문에서는 2-큐브 논리식 쌍과 함께 2-큐브 논리식의 보수를 추가하여 공통식을 찾도록 고안하였다.

지금까지의 논리합성 연구 결과는 일부 회로에서만 SIS보다 상대적으로 좋은 결과를 산출할 수 있었다. 이러한 관점에서 부울공리를 적용한 공통식(이하 부울 공통식 또는 부울식이라 부름) 산출 방법은 대수 나눗셈을 이용한 방법보다 리터럴 개수를 줄일 수 있음을 알 수 있으나, 여전히 어려운 문제로 남아있기 때문에 최근에는 이 분야에 대한 연구가 다소 주춤한 상황이다. 따라서, Field-Programmable Gate Array (FPGA)[11, 12]와 같이 논리 최적화 결과를 활용하는 분야에서는 주로 SIS의 결과를 그대로 활용하는 것으로 만족하고 있다.

본 논문에서는 선험방법(heuristic method)에 의한 부울 공통식 산출 방법을 제안한다. 제안하는 핵심 기술은 주어진 논리식들에서 2개의 큐브로 구성된 논리식 쌍(pair)을 산출하는 것이다. 실제 상황에서 논리식의 항수 또는 리터럴 개수가 매우 많기 때문에 공통식 추출을 위한 모든 경우를 고려할 수가 없다. 따라서, 본 논문에서는 2개의 큐브로만 표현된 공통식들과 그들의 보수를 고려하여 간략화된 논리식 산출을 얻도록 하였다.

논문의 구성은 다음과 같다. 2절에서는 본 논문 서술에 필요한 정의와 커널 기반 공통식 산출방법[2]에 대하여 서술하고, 3절에서 본 논문에서 제시하는 새로운 공통식 산출 방법을 소개한다. 4절에서 실험결과를 보이고, 5절에서 결론을 제시한다.

2. 배경 지식

본 논문의 공통식 산출 방법을 서술하는데 필요한 용어들과 본 논문의 비교 대상이 되는 커널 기반 공통식 산출 방법에 대하여 서술한다.

2.1 정의

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 변수이며, a와 a'은 리터럴이다. 리터럴 집합 $\{a,b\}$는 큐브이나 집합 $\{a,a'\}$은 큐브가 아니다. $\{\{a,b'\},\{b,c\}\}$는 단순식이다.

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

정의 2: 논리식 F의 서포트(support)는 논리식 F를 구성하는 변수들 집합으로 $sup(F)$로 표현한다. 논리식을 구성하는 모든 큐브들 간에 공통으로 사용되는 리터럴을 갖지 않은 경우 그 논리식은 큐브면제(cube-free) 되었다고 한다. 논리식이 어떤 큐브로부터 나누어졌을 때, 몫이 큐브면제라면 그 몫을 커널이라 한다. 이 때 커널을 산출한 큐브를 코커널(co-kernel)이라 한다.

예 2: 논리식 $F=a+bc'$의 경우, $sup(F)=\{a, b, c\}$. 논리식 $ab+c$ 는 큐브 면제된 경우이나, 논리식 $ab+ac$ 및 abc는 큐브 면제된 것이 아니다. $F=bc'd'e + ab'c + ab'e + ac'd'$은 다시 $F=bc'd'e+a(b'c+b'e+c'd')$으로 표현될 수 있으며, 이 때 $b'c+b'e+c'd'$은 코커널 a에 대한 커널이 된다.

정의 3: 논리회로는 비순환 방향 그래프(directed acyclic graph)로 표현되며, 각 노드 i는 변수 y_i 를 나타내고, 논리식 F_{y_i} 를 표현한다. 2개의 논리식 F와 G의 곱 FG는 $\{C_i \cup D_j | C_i \in F$와 $D_j \in G\}$를 의미한다. F와 G가 서로 독립 서포트(disjoint support)인 경우 FG는 대수 곱이고,

그 외의 경우 FG는 부울 곱이다. F/G는 가장 큰 큐브 집합인 몫 Q를 산출하여, 논리식 F를 $F=QG+R$로 표현할 수 있다. 여기서, R은 나머지를 나타낸다. QG가 대수 곱인 경우, F/G는 대수 몫을, 그 외의 경우 F/G는 부울 몫이 된다. $F/G=Q\neq\varnothing$ 이고 Q가 대수 나눗셈에 의해 산출된 경우, G는 대수 제수(algebraic divisor)라 하고, 그 외의 경우 부울 제수(Boolean divisor)라 한다.

예 3: $(a+b)(c+d)=ac+ad+bc+bd$는 대수 곱이며, $(a+b)(a+c)=a+ab+ac+bc$는 부울 곱이다. $F=ad+abc+bcd$ 이고 $G=a+bc$ 가 주어진 경우를 보자. F/G의 결과로는 대수 몫 d와 나머지 abc가 산출된다. 이 때, $a+bc$는 대수 제수가 된다. $F=abg+acg+adf+aef+afg+bd+be+cd+ce$와 $G=ag+d+e$가 주어진 경우, $F=(af+b+c)(ag+d+e)$로 표현될 수 있다. 이 때, $af+b+c$는 부울 몫이며, $ag+d+e$는 부울 제수가 된다.

2.2 커널 기반 공통식 산출

Brayton과 McMullen은 커널 집합을 이용해서 논리식들에서 공통식을 산출하는 과정을 2단계로 제안하였다[1]. 논리식들에서 커널들 산출이 첫 번째 단계이며, 커널 교집합으로부터 공통식을 선별하는 과정이 두 번째 단계다.

예 4: 다음과 같이 논리식 F_0와 F_1이 주어졌다고 하자. $F_0=ace+bce+de+g$, $F_1=ad+bd+cde+ge$. 첫 번째 단계에서, F_0의 커널 집합 $K(F_0)$은 $K(F_0)=\{(a+b),\ (ac+bc+d),\ (ace+bce+de+g)\}$이며, F_1의 커널 집합은 $K(F_1)=\{(a+b+ce),\ (cd+g),\ (ad+bd+cde+ge)\}$. 두 번째 단계에서, F_0와 F_1으로부터 다항 큐브인 공통식을 추출한다. $(a+b)\in K(F_0)$와 $(a+b+ce)\in K(F_1)$ 사이에 커널 교집합으로 $a+b$를 얻게 된다. 이로부터 주어진 논리식들은 다음과 같이 변형된다.

$$F_0=wce+de+g$$
$$F_1=wd+cde+ge$$
$$F_w=a+b.$$

3. 부울 공통식 산출

부울 공통식 산출에는 2개의 큐브로 구성된 논리식 쌍을 이용한다. 본 절에서는 이러한 부울 논리식 쌍을 찾는 방법을 제안하고, 또한 산출한 논리식 쌍을 이용해서 전역 최적화를 수행하는 방법을 제시한다.

3.1 2개의 큐브로 구성된 부울식 쌍

2개의 큐브로 구성된 부울식 쌍은 2개의 제수/몫 쌍들로부터 산출된다. 주어진 논리식에서 2개의 큐브를 선택하고 이 큐브들로부터 공통 큐브를 찾는다. 주어진 논리식을 F, C를 제수 집합, Q를 2개의 큐브로 구성된 몫 집합이라 하자. 표기상 제수/몫 쌍을 괄호를 이용하여 표현한다. $c_i\in C$, $c_j\in C$, $q_i\in Q$, $q_i\in Q$이고 $i\neq j$라 하자. 그러면, (c_i, q_i), (c_j, q_j)는 대수 나눗셈에 의한 제수/몫 쌍을 표현한 것이다. 만일 $c_i\in q_j$, $c_j\in q_i$ 이고 $q_i q_j$ 가 주어진 논리식 F에 포함되면, (q_i, q_j)는 2개의 큐브로 구성된 부울식 쌍이라 한다.

예 5: 다음 4개의 논리식들이 주어졌을 경우, 제수/몫 쌍과 몫/몫 쌍인 부울식 쌍을 찾아보자.

$$F_0=vux'y+wxz+yz$$
$$F_1=wxz+vx'y+yz$$
$$F_2=v'x+v'yz+w'x+w'yz$$
$$F_3=v'y+w'y$$

그러면, 제수/몫 쌍은 다음과 같이 산출된다. 논리식 F_0로부터 $\{(w,vx'y+xz),\ (y,vux'+z),\ (z,wx+y)\}$. F_1으로부터 $\{(z,wx+y),\ (y,vx'+z)\}$. F_2로부터 $\{(v',x+yz),\ (x,v'+w'),\ (yz,v'+w'),\ (w',x+yz)\}$. F_3로부터 $\{(y,v'+w')\}$. F_0의 제수/몫 쌍인 $(y,vux'+z)$와 $(z,wx+y)$를 보자. $(y,vux'+z)$에서 몫 $vux'+z$의 z는 $(z,wx+y)$의 제수 z와 동일하고, $(z,wx+y)$에서 몫 $wx+y$의 y는 $(y,vux'+z)$의 제수 y와 동일하다. 그리고, $(vux'+z)(wx+y)$는 F_0에 속한다. 따라서, $(vux'+z, wx+y)$는 F_0에 속하는 2개의 큐브로 구성된 부울식 쌍이 된다. 동일한 방법으로 F_1의 $(y,vx'+z)$와 $(z,wx+y)$로부터 $(vx'+z, wx+y)$를, F_2의 $(v',x+yz)$와 $(x,v'+w')$ 또는 $(v',x+yz)$와 $(yz,v'+w')$로부터 $(x+yz, v'+w')$을 얻는다.

3.2 2-큐브 행렬

다수 개의 논리식들이 주어진 경우, 제수들의 집합 C와 2개의 큐브로 구성된 몫들의 집합 Q를 3.1절에서 서술한 바와 같이 산출할 수 있다. 그러면 산출된 집합 Q의 원소 q_i에 0보다 큰 양의 정수 값을 배정한다. 이 때 값을 배정하는 함수를 $index(q_i)$로 표기한다.

부울 공통식을 산출하기 위해서 2개의 큐브로 구성된 몫들을 이용해서 2-큐브 행렬 T를 만든다. 행렬 T의 행들은 각 논리식을 나타내며, 열은 2개의 큐브로 구성된 몫 즉, 2-큐브식이다. 이 때, 행이 F_k이고 열이 q_i일 경우 행렬의 원소 $T(F_i,q_i)$에 대한 정의는 다음과 같다.

$$T(F_k,q_i)=\begin{cases} c_i & (c_i,q_i)\in F_k, c_i\in C, \text{ 이고} \\ & q_i\in Q \text{ 일 때} \\ c_i, index(q_j) & (c_i,q_i)\in F_k, (q_i,q_j)\in F_k, \\ & q_i\in Q, \text{ 이고} q_j\in Q \text{ 일 때} \end{cases}$$

예 6: 예 5의 4개 논리식들에 대하여 2-큐브 행렬을 산출하는 예를 보인다. 먼저 표 1에 보인 것처럼 2-큐브 몫들의 집합 Q의 각 원소에 양의 정수를 배당하고, 이 배정된 값을 이용해서 표 2에 2-큐브 행렬의 예를 보였다. 4개의 논리식과 6개의 2-큐브식이 산출되었기 때문에 행렬은 4개의 행과 6개의 열로 구성된다. 논리함수 F_0에서 제수/몫인 $(w, vx'y+xz)$의 경우를 보자. 행렬에서 F_0를 나타내는 첫 번째 행과 몫 $vx'y+xz$을 나타내는 첫 번째 열에 해당하는 원소에는 제수 w가 입력된다. 또, F_0의 $(y, vwx'+z)$와 $(z, wx+y)$로부터 부울식 쌍 $(vwx'+z, wx+y)$를 산출하였고, 표 1에서 $index(wx+y)=3$으로 값이 배정되었기 때문에, $T(F_0, vwx'+z)=y,3$이 된다. 그 외 행렬의 원소 내용도 같은 방법으로 원소의 내용이 결정된다.

〈표 1〉 2-큐브 몫에 대한 *index* 배정

2-큐브 q_i	$vx'y + xz$	$vwx' + z$	$wx + y$	$vx' + z$	$x + yz$	$v' + w'$
$index(q_i)$	1	2	3	4	5	6

〈표 2〉 2-큐브 행렬 T

F \ q_i	$vx'y + xz$	$vwx' + z$	$wx + y$	$vx' + z$	$x + yz$	$v' + w'$
F_0	w	y,3	z,2			
F_1			z,4	y,3		
F_2					v',6; w',6	x,5; yz,5
F_3						y

3.3 2-큐브 행렬의 확장과 압축 행렬

2-큐브 행렬 T를 산출한 후, 2-큐브식에서 각 항의 보수를 구하고, 다시 이 보수들을 공통인수로 포함하는 논리식들을 산출하기 위해서 2-큐브 행렬 T에 보수를 포함하는 행렬 XT로 확장한다. 확장된 행렬 XT는 2-큐브 행렬 T에 열을 추가해서 만든다. 열을 추가하는 방법은 다음과 같다. 행렬 T에서 각 열인 2-큐브식의 각 항 보수를 구하고, 이 보수에 대응하는 열을 2-큐브 행렬 T에 추가한다. 추가된 열과 각 함수에 대응하는 원소에는 다음 수식에 따라 값이 배정된다. 즉, 확장된 2-큐브 행렬 XT에서 추가된 열에 해당하는 논리식을 p_l이라고 할 경우 원소의 값은 다음과 같이 결정된다.

$$XT(F_k, p_l) = index(q_i) \quad p_l' \in q_i \text{ 이고 } q_i \in F_k \text{ 일 때}$$

예 7: 예 5의 4개 논리식들과 예 6의 2-큐브 행렬로부터 확장된 행렬을 산출한 결과가 표 3과 같다. 표 2의 행렬 T에 6개의 열을 추가한 것이 표 3의 확장된 행렬 XT 이다. 표 3에서 열 $v'+x+y'$ 은 행렬 T의 첫 번째 열인 $q_1 = vx'y + xz$ 에서 $vx'y$의 보수를 구한 것이다. 여기서, $p_l = v' + x + y'$ 라 하면, $p_l' = vx'y$ 이고 $p_l' \in q_1$ 이므로, $XT(F_0, p_l) = index(q_1)$. 즉, $XT(F_0, p_l) = 1$이 된다. 나머지 추가된 열과 원소의 내용은 위와 같은 방법으로 산출된다.

<표 3>과 같이 확장된 행렬 XT를 산출한 다음 이 확장된 행렬로부터 다시 압축 행렬 CT를 만든다. 이 때, 압축 행렬의 행과 열 수는 2-큐브 행렬 T와 같게 된다. 확장된 행렬 XT에서 각 행 단위로 확장 전의 2-큐브 식과 추가된 열에 해당하는 식을 비교하고, 추가된 열의 식이 2-큐브 식을 포함하면, 2-큐브 식에 해당하는 원소에 추가된 열의 원소 값 보수를 입력한다. 즉, 행렬 CT의 원소는 다음과 같이 결정된다.

$$CT(F_k, q_i) = \begin{cases} (index(q_j))' & XT(F_k, p_l) = index(q_j) \\ & \text{이고 } q_i \subset p_l \text{ 일 때} \\ T(F_k, q_i) & \text{그 외의 경우} \end{cases}$$

예 8: 예 7의 행렬 XT로부터 표 4의 압축 행렬 CT를 만든다. XT의 F_0 행을 보자. 이 행의 여섯 번째 열 $v'+w'$는 아홉 번째 열 $v'+w'+x$에 포함되고, $XT(F_0, v'+w'+x) = 2$ 이기 때문에 $CT(F_0, v'+w') = 2'$ 이 된다.

〈표 4〉 압축 2-큐브 행렬 CT

F \ q_i	$vx'y + xz$	$vwx' + z$	$wx + y$	$vx' + z$	$x + yz$	$v' + w'$
F_0	w	y,3	z,2			2′
F_1			z,4	y,3		
F_2					v',6; w',6	x,5; yz,5
F_3						y

〈표 3〉 확장된 2-큐브 행렬 *XT*

F \ q_i	$vx'y + xz$	$vwx' + z$	$wx + y$	$vx' + z$	$x + yz$	$v' + w'$	$v'+x + y'$	$x' + z'$	$v'+w'+x$	$w' + x'$	$v' + x$	$y' + z'$	p_l \ F
F_0	w	y,3	z,2				1	1	2	3			F_0
F_1			z,4	y,3						3	4		F_1
F_2					v',6; w',6	x,5; yz,5						5	F_2
F_3						y							F_3

3.4 가중치 계산

주어진 논리식들을 가장 적은 개수의 리터럴을 갖는 논리식들로 전환하기 위해서 여러 논리식들에서 가능한 많은 리터럴들을 줄일 수 있는 공통식들을 선택하는 것이 중요하다. 본 논문에서는 공통식 선택을 위해 가중치 부여 방법을 사용하였다. 2-큐브식 q_i에 다음 알고리즘에 따라 가중치를 부여하도록 하였다. 여기서, 2-큐브식 q_i의 가중치 값이 클수록 여러 논리식에서 공통으로 사용될 가능성이 높고, 결국은 전체 논리식의 리터럴 개수를 줄일 수 있게 된다. 다음 알고리즘에서 $NF(q_i)$는 2-큐브식 q_i를 포함하는 논리식들의 개수이고, $L(q_i)$는 q_i 자체의 리터럴 개수를 표현한 것이다. 제시한 방법은 순환 알고리즘으로 가중치 산정에 2-큐브식이 반복 사용될 수 있기 때문에 각 2-큐브식 마다 태그(tag)를 두어 반복 사용을 피하도록 하였다.

```
Algorithm : 가중치 산출(Weight calculation)
  입력 : 행렬 CT.
  출력 : 2-큐브식 q_i (q_i∈Q) 대한 가중치 weight(q_i)
begin
   for all q_i in Q do
      q_i.tag = not visited;
   for each q_i in Q do
      q_i.cost = weight(q_i);
end;

function weight(q_i)
   begin
      q_i.tag = visited;
      cost = (NF(q_i) - 1)(L(q_i) - 1) - 1;
      if (cost <= 0)
         return(0);
      for each F_k in F do
      begin
         if ((CT(F_k, q_i) == index(q_j)) and
              (q_j.tag == not visited))
         then cost = cost + weight(q_j);
      end
      return(cost);
   end;
```

예 9: <표 4>의 CT 행렬에서 $q_3 = wx + y$에 대한 가중치 weight(q_3)를 구해보자. 먼저, $q_3.tag$ = *visited*로 설정하여 다시 q_3가 가중치 산정에 이용되지 못하도록 한다. q_3는 F_0와 F_1 에서 공통으로 사용되었기 때문에 $NF(q_3)$ = 2가 되고, q_3 자체는 3개의 리터럴로 구성되었기 때문에 $L(q_3)$ = 3이다. 따라서, $cost = (NF(q_3) - 1)(L(q_3) - 1) - 1 = 1$로 산출된다. 다음 weight() 함수의 for-loop내 첫 번째 if문 조건 검사에서 $CT(F_0, q_3)$=2이므로 다시 weight(q_2)가 호출된다. 그러나, q_2는 논리식 F_0에만 포함되기 때문에 weight(q_2)는 가중치로 0을 리턴한다. 또, $CT(F_1, q_3)$ = 4에 대해서도 weight(q_4)가 가중치로 0을 리턴하기 때문에 $q_3.cost$ = 1이 된다. 나머지 2-큐브식에 대해서도 동일한 방법으로 가중치가 산출된다.

3.5 알고리즘

제안하는 공통식 추출 알고리즘은 다음과 같다. 공통식들을 찾기 위해 2-큐브 쌍들을 구하고, 이들로부터 압축 행렬 CT와 각 2-큐브식에 대한 가중치를 산출한다. 양수 값을 갖는 가중치 중에서 가장 큰 값을 갖는 2-큐브식을 선택하고, 이 2-큐브식을 새로운 변수로 대치한다. 그리고, 각 논리식을 조사해서, 이 2-큐브를 포함하는 논리식들에는 새로운 변수를 대입하여 간략화된 논리식으로 변환한다. 전체 알고리즘은 다음과 같다.

```
Algorithm : 부울 공통식 추출(Boolean extraction)
입력 : 주어진 논리식 집합 F
출력 : 공통식 추출에 의한 간략화된 논리식 집합 F
begin
   for each expression F∈ F do
   begin
      대수 2-큐브제수/몫 쌍 집합 DQ 산출;
      DQ로부터 2-큐브 부울식 쌍 집합 P 산출;
      2-큐브제수 집합 Q 추출;
   end
   DQ, P, Q를 이용해서 2-큐브 행렬 T 생성;
   T와 Q를 이용해서 확장된 2-큐브 행렬 XT 생성;
   행렬 XT를 압축 행렬 CT로 변형;
   각 2-큐브식 q_i (q_i∈Q) 에 대한 가중치 산출;
   k = 0;
   while (Q≠∅) do
   begin
      가장 큰 양의 가중치를 갖는
                2-큐브식 q_i∈Q 선택;
      Q = Q - q_i;
      G_k = q_i;
      for each expression F∈ F do
      begin
         if (q_i∈F )
         then 논리식 F에서 2-큐브식 q_i를
              새로운 변수 G_k로 치환;
      end
      F = F ∪{G_k};
      k = k + 1;
   end
   논리식 집합 F에서 단일항의 공통식 추출;
end
```

예 10: 예 5의 논리식들에 대하여 위 알고리즘을 적용하여 공통식을 추출하고, 간략화된 논리식으로 변환하는 결과를 보인다. 알고리즘은 예 8과 9에서 보인 바와 같이 CT 행렬과 가중치를 산출하고 while-loop을 수행한다. 이 예에서 가장 큰 가중치를 갖는 2-큐브식으로 $q_3 = wx + y$ 선택하면, $G_0 = wx + y$로 새로운 논리식이 만들어지고, q_3를 포함하는 논리식 F_0와 F_1을 다시 G_0을 포함하는 논리식으로 변환한다. 그러면, 주어진 논리식 F_0는 $F_0 = G_0(vwx' + z)$로 표현된다. 또 논리식 $F_1 = (wx + y)(vx' + z)$으로부터 $F_1 = G_0(vx' + z)$을 얻게 된다. 다시, 알고리즘은 공통식으로 $q_6 = v' + w'$ 를

선택한다. 그러면, F_2와 F_3는 모두 q_6를 포함하고, F_0은 $(q_6)' = vw$를 포함한다. 따라서, $G_1 = v' + w'$로 결정되어 $F_0 = G_0(vwx' + z)$는 $F_0 = G_0(G_1'x' + z)$로 변환되며, $F_2 = (v' + w')(x + yz)$는 $F_2 = G_1(x + yz)$로, $F_3 = (v' + w')y$는 $F_3 = G_1 y$로 변환된다. 따라서, 제안한 알고리즘이 산출한 논리식들은 다음과 같이 19개의 리터럴을 갖는 논리식들로 전환된다.

$$F_0 = G_0(G_1'x' + z)$$
$$F_1 = G_0(vx' + z)$$
$$F_2 = G_1(x + yz)$$
$$F_3 = G_1 y$$
$$G_0 = wx + y$$
$$G_1 = v' + w'$$

반면에, 대표적인 논리합성 도구인 SIS를 이용할 경우 22개의 리터럴을 갖는 다음 논리식들을 산출한다.

$$F_0 = [1]z + [3]w$$
$$F_1 = [1]z + [3]$$
$$F_2 = [2](v' + w')$$
$$F_2 = y(v' + w')$$
$$[1] = wx + y$$
$$[2] = x + yz$$
$$[3] = wx'y$$

4. 실험 결과

4.1 실험 방법

제안한 알고리즘을 Pentium IV 1.4GHz CPU PC의 Linux 환경에서 실험하였다. 실험은 ALU, 곱셈기(multiplier), 랜덤로직(random logic) 등 다양한 논리 회로로 구성된 Microelectronics Center of North Carolina(MCNC) 벤치마크 회로[13]에 대하여 리터럴 개수와 수행시간을 기준으로 비교하였다. 비교 실험은 2가지로 나누어 진행하였다. 첫째로, 이미 발표된 논리합성 도구들이 본 논문에서 다루는 공통인수 추출뿐만 아니라 분할(decomposition), 대입(substitution), 인수분해(factorization) 등의 모든 방법을 조합해서 논리식 간략화를 수행한다. 따라서, 동일한 조건에서 실험을 하기 위해서 제시한 알고리즘을 SIS에 삽입하여 수행 결과를 산출하고 다른 합성도구들과 성능을 비교하였다. SIS는 새로운 기능 추가가 매우 용이하도록 설계되어 있기 때문에 SIS를 선택하였다. 둘째로, 제시한 공통식 추출 방법만을 비교하기 위해 커널기반 공통식 산출 방법[2,3]에 의한 결과와 비교하였다.

첫째로, SIS의 script.algebraic와 script.rugged를 이용한 출력 결과 및 BDD를 이용한 논리합성 도구인 FBDD의 출력 결과와 비교하였다. 참고로, 스크립트는 SIS가 수행할 여러 논리합성 명령을 차례로 나타낸 파일이다. 표 5는 본 실험에 사용된 스크립트 파일들이다. 본 논문에서는 스크립트 명령에 사용되는 선택사항(option)에 대한 설명은 생략하고 각 명령에 대한 설명만을 간략히 소개한다. 표 5에서 sweep 명령은 논리회로에서 버퍼(buffer)와 인버터(inverter) 게이트를 제거하는 역할을 한다. 논리회로를 그래프로 나타내면 그래프의 노드는 논리식으로 표현되는데, 이 때 이 노드를 나타내는 논리식의 리터럴 개수가 사용자가 원하는 값보다 작은 경우 다른 노드들과 묶어서 하나의 노드로 만드는 명령이 eliminate 명령의 역할이다. Resub 명령은 논리식들끼리 나눗셈을 수행하기 위한 것이며, gkx와 gcx는 논리식에서 공통식을 추출하는 명령이다. Simplify와 full_simplify 명령은 무관항(don't-care term)을 이용한 노드 간략화를 수행하는 명령이다. Script.rugged에서 사용된 fx 명령에 대해서는 다음에 설명한다. Script.algebraic와 script.rugged는 SIS에서 기본적으로 제공하는 스크립트 파일이며, 특히 script.rugged는 Rajski 등[8]이 제안한 방법이 활용된 스크립트로 대체로 script.algebriac를 이용한 경우보다 더 좋은 결과를 산출한다. 실험을 위해 본 논문에서 제시한 알고리즘을 SIS에 추가하였으며 ex2cube라는 이름의 스크립트 명령으로 실행하도록 하였다. <표 5>에서 오른쪽 스크립트가 제안한 방법을 활용한 스크립트 파일이다. 이 스크립트는 중앙의 script.rugged와 대부분 동일하고 단지 fx 명령을 ex2cube로 바꾸었다. Fx 명령이 바로 Rajski 등이 제시한

〈표 5〉 Script 파일 내용

script.algebraic	script.rugged	제안한 script
sweep eliminate 5 simplify -m nocomp -d resub -a gkx -abt 30 resub -a; sweep gcx -bt 30 resub -a; sweep gkx -abt 10 resub -a; sweep gcx -bt 10 resub -a; sweep gkx -ab resub -a; sweep gcx -b resub -a; sweep eliminate 0 decomp -g *	sweep; eliminate -1 simplify -m nocomp eliminate -1 sweep; eliminate 5 simplify -m nocomp resub -a fx resub -a; sweep eliminate -1; sweep full_simplify -m nocomp	sweep; eliminate -1 simplify -m nocomp eliminate -1 sweep; eliminate 5 simplify -m nocomp resub -a **ex2cube** resub -a; sweep eliminate -1; sweep full_simplify -m nocomp

〈표 6〉 기존 논리합성 도구들과 논리회로 산출 비교 결과

회로	입력 수	출력 수	script.algebraic		script.rugged		FBDD		제안방법활용	
			리터럴 수	시간 (초)	리터럴수	시간 (초)	리터럴수	시간 (초)	리터럴 수	시간 (초)
Example1	5	4	23	0.2	20	0.2	45	0.2	19	0.2
b12	15	9	99	0.2	96	0.4	163	0.4	93	0.5
rd53	5	3	65	0.3	42	0.2	80	0.2	37	0.3
rd73	7	3	148	0.5	74	0.4	234	0.3	69	0.5
rd84	8	4	181	2.9	194	1.0	292	0.6	176	2.2
con1	7	2	23	0.2	21	0.2	48	0.1	22	0.3
z4ml	7	4	42	0.2	46	0.2	53	0.1	41	0.3
cmb	16	4	37	0.2	37	0.2	70	0.2	37	0.3
vg2	25	8	97	0.2	106	0.4	930	0.2	92	0.9
decod	5	16	52	0.2	58	0.4	57	0.3	58	0.8
misex1	8	7	67	0.2	58	0.2	167	0.1	52	0.3
alu4	14	8	1099	65.1	283	23.2	5206	5.2	255	35.8
sao2	10	4	185	0.2	176	0.5	426	0.3	157	1.2
e64	65	65	253	0.7	253	1.9	320	0.5	253	2.3
apex6	135	99	854	0.3	819	0.6	1441	0.2	799	0.7
C880	60	26	473	0.2	467	0.5	634	0.1	465	1.2
C1355	41	32	670	0.2	560	1.8	610	0.1	554	2.3
C1908	33	25	564	0.3	557	2.1	605	0.2	552	3.0
C2670	233	140	840	0.5	910	1.6	1352	0.4	902	3.5
C5315	178	123	2008	1.8	1837	2.8	2688	1.0	1824	5.1
C6288	32	32	3787	2.7	3348	8.2	4800	0.3	3350	8.9
C7552	207	108	2584	3.4	3255	14.1	3213	2.4	2450	20.9

〈표 7〉 커널기반 방법과 제시한 방법인 공통식 추출 방법만의 비교

회로	입력 수	출력 수	커널기반방법		제안방법	
			리터럴 수	시간 (초)	리터럴 수	시간 (초)
Example1	5	4	22	0.1	19	0.1
b12	15	9	124	0.1	119	0.1
rd53	5	3	77	0.2	71	0.1
rd73	7	3	176	0.4	169	0.4
rd84	8	4	243	0.2	236	0.2
con1	7	2	23	0.1	23	0.1
z4ml	7	4	70	0.2	61	0.3
cmb	16	4	70	0.1	73	0.1
vg2	25	8	107	0.1	112	0.1
decod	5	16	64	0.1	51	0.1
misex1	8	7	80	0.1	80	0.1
alu4	14	8	1755	2.0	1557	1.8
sao2	10	4	203	0.3	192	0.4
e64	65	65	254	0.1	254	0.1
apex6	135	99	904	0.1	902	0.1
C880	60	26	702	0.1	628	0.2
C1355	41	32	1032	0.1	989	0.3
C1908	33	25	1469	0.1	982	0.3
C2670	233	140	1995	0.2	1509	0.4
C5315	178	123	4355	0.2	3268	0.5
C6288	32	32	4800	0.1	4705	0.7
C7552	207	108	5968	0.1	4510	0.8

double-cube를 이용한 공통논리식 추출 방법으로, 본 논문에서 제시한 방법과 유사하다. 따라서, script.rugged의 fx 명령을 제외한 다른 부분은 그대로 활용하고 fx 명령만 교체하여 <표 5>의 오른쪽과 같은 스크립트를 만들어 실험하였다. <표 6>은 <표 5>의 스크립트들과 FBDD 도구를 이용하여 산출한 결과를 비교한 것이다. 표 6의 첫 번째 열은 벤치마크 회로의 이름이고, 다음 2개의 열은 입력 수와 출력 수를 나타낸 것이다. 그 다음 열들은 논리합성 도구가 수행한 결과들이다. 마지막 2개열은 본 논문에서 제안한 방법을 활용해서 산출된 결과를 나타낸 것이다.

둘째로, 공통식 추출 방법만을 비교하기 위해 제시한 방법을 대수 방법인 커널 기반 공통식 추출 방법과 비교하였다. 비교 결과는 <표 7>에 제시하였다. 커널 기반 공통식 추출을 하기 위해서 SIS의 명령 gkx와 gcx를 이용하였다.

4.2 결과 분석

<표 6>의 script.algebraic과 script.rugged 스크립트에 의한 합성 결과와 비교해서 제안한 방법을 사용한 스크립트로 합성을 한 경우 대부분의 벤치마크 회로에 대하여 리터럴 개수를 줄일 수 있었다. 특히, script.rugged와 비교하면 본 논문에서 제시한 방법인 ex2cube가 Rajski 등의 방법인 fx보다 더 좋은 결과를 산출하는 것으로 판단할 수 있다. FBDD의 경우는 빠른 수행 시간을 보였으나 리터럴 개수를 줄이는 데는 우수하지 않음을 보이고 있다. 또, 표 7로부터 커널기반 방법은 일부 회로에 대하여 제안한 방법보다 적은 수의 리터럴을 갖는 논리식을 산출하였다. 반면에 제안한 방법은 대부분의 경우 리터럴 개수를 줄이는 데 효과적임을 보이고 있다. 특히, 대칭 논리식(symmetric expression)인 rd53, rd73, z4ml의 경우 제안한 방법으로 보다 리터럴 개수를 줄이는 효과를 얻을 수 있었다. <표 6>과 <표 7>의 실험 결과에서 수행 시간이 증가한 원인은 본 논문의 3.1절에

서 다룬 2개의 큐브로 구성된 부울식 쌍을 산출하는 과정이 다른 방법들과 달리 추가되었기 때문이다. 반면에 이러한 부울식 쌍들을 이용함으로써 리터럴 개수를 줄이는 효과를 얻었다.

5. 결 론

논리회로의 출력수가 2개 이상의 경우 최적화된 논리회로를 얻기 위해서는 각 출력에 동일하게 사용되는 공통회로 또는 공통 논리식을 찾아 회로를 최적화하는 것이 중요한 일이다. 보통 부울 공통식을 산출하는데 장시간의 수행 시간이 요구되기 때문에 합성 도구들이 대수 방법으로 전체 회로를 간략화하기 위한 작업을 수행하나, 항상 최적의 결과를 얻을 수 없게 된다. 이러한 문제를 해결하기 위해서, 본 논문에서는 2개의 큐브로만 구성된 부울식 쌍들에 등멱법칙과 보수법칙을 고려하여 리터럴 개수를 줄일 수 있는 공통식과 그의 보수를 찾도록 하였다. <표 6>의 실험결과에 보였듯이 제시한 방법을 활용할 경우 실용성이 매우 높을 것으로 판단된다.

참 고 문 헌

[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.

[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] W.-J. Hsu and W.-Z. Shen, "Coalgebraic Division for Multilevel Logic Synthesis," *Proc. of DAC*, pp.438-442, 1992.

[5] C. Yang and M. Ciesielski, "BDS: A Boolean BDD-Based Logic Optimization System," *IEEE Trans. CAD,* Vol. 21, No. 7, pp.866-876, 2002.

[6] D. Wu and J. Zhu, "FBDD: A Folded Logic Synthesis System," Technical Report TR-07-01-05, University of Toronto, July, 2005.

[7] S. Nagayama and T. Sasao, "Representation of Elementary Functions Using Edge-Valued MDDs," *Proc. of the 37th International Symposium on Multiple-Valued Logic(ISMVL '07)*, pp.5-11, 2007.

[8] 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.

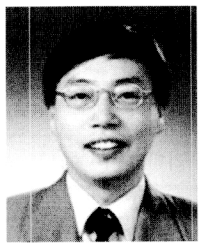
[9] D. Wu, and J. Zhu, "BDD-based Two Variable Sharing Extraction," *Proc. ASPDAC*, pp.1031-1034, 2005.

[10] O.-H. Kwon, "Boolean Extraction Technique for Multiple-level Logic Optimization," *Proc. ISCAS*, Vol. 4, pp.684-687, 2003.

[11] J. Cong and K. Minkovich, "Optimality Study of Logic Synthesis for LUT-Based FPGAs," *IEEE Trans. CAD,* Vol. 26, No. 2, pp.230-239, 2007.

[12] A. C. Ling, P. Singh, and S. D. Brown, "FPGA PLB Architecture Evaluation and Area Optimization Techniques Using Boolean Satisfiablity," *IEEE Trans. CAD,* Vol. 26, No. 7, pp.1196-1210, 2007.

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



권 오 형

e-mail : ohkwon@hanseo.ac.kr

1984년 인하대학교 전자계산학과(학사)

1988년 The University of Alabama in Hunsville Computer Science (석사)

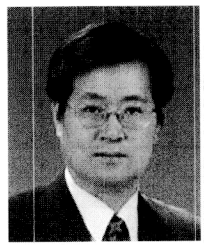
1999년 포스텍 대학원 컴퓨터공학과 (공학박사)

1990년~1993년 한국전자통신연구원

1999년~2003년 위덕대학교 컴퓨터공학과 조교수

2003년~현 재 한서대학교 인터넷공학과 부교수

관심분야 : 설계자동화



오 임 걸

e-mail : oig@hanseo.ac.kr

1983년 인하대학교 수학과(학사)

1986년 인하대학교 대학원 수학과(이학석사)

1993년 인하대학교 대학원 통계학과 (이학박사)

2000년 8월~현 재 인하대학교 컴퓨터공학과 박사과정

1995년 3월~현재 한서대학교 인터넷공학과 부교수

관심분야 : 컴퓨터 보안, 암호학, 컴퓨터 통신