論文95-32A-10-11

멀티플렉서에 기초한 논리모듈의 Library 생성 방법

(A Library Generation Method for Multiplexor-based Logic Module)

曺 漢 鎭 *, 裵 英 煥 *, 朴 仁 學 *

(Hanjin Cho, Young-Hwan Bae, and Inhag Park)

요 약

본 논문은 멀티플렉서에 기초한 논리모듈을 최적화 하는데 있어서, 논리모듈의 능력을 평가하고 technology mapping을 위한 library를 생성할 수 있는 방법에 대하여 논하였다. 즉 멀티플렉서에 기초한 논리모듈에 임의의 제어 게이트와 입력 게이트를 배치했을 때 첨가되는 제어 게이트의 실리콘 면적과 주어진 모듈이 구현할 수 있는 출력 함수의 능력을 최적화 하는 방법에 대하여 설명한다. 이렇게 생성된 library가 만족스럽지 못할 경우는 모듈의 구조를 변경하여 새로운 논리모듈이 구현할 수 있는 출력 함수를 알아내고 새로운 library를 생성하여 mapping에 사용할 수 있다. 멀티플렉서에 기초한 논리모듈이 구현할 수 있는 모든 논리를 library화 하기에는 현실적으로 불가능하나, 매핑이 가능한 최대 입력 변수의 수와 최대 product term의 수를 제한하고, 출력된 함수 중에서 입력의 조합이 다르면서도 같은 논리를 구현하는 함수, 즉 입력 변수의 순서를 바꿈으로 같은 논리가 되는 함수들을 제거하면 library화할 출력 함수의 수를 현저하게 줄일 수 있다.

Abstract

The evaluation of the logic capability and the library generation method of the multiplexor-based logic module is described. Optimizing logic module for silicon area and logic capability is essential to build a efficient FPGAs(Field-Programmable Gate Arrays). Because the multiplexor-based logic module can implement a large number of functions, it presents difficulties for library-based approaches. However, the logic functions of the logic module can be significantly reduced by limiting the number of variables and sum-of-products and by removing same functions with different variable ordering using algorithm presented in this paper.

I. 서 론

FPGA(Field Programmable Gate Array)의 논리모듈은 크게 Look Up Table(LUT) 방식과 멀티플렉서를 기본으로 한 방식으로 나누어진다. LUT 방식은 주어진 입력 변수의 구현 가능한 모든 함수를 구현할 수 있다. LUT의 하드웨어 소자로는 SRAM (Static RAM)을 사용하는데 n 입력의 LUT를 구현하는데 필요한 메모리는 2^n 개이다. 또 원하는 함수를 구현하기 위해 n 입력의 디코더와 2^n 입력의 멀티플렉서가 필요하다. 따라서 하나의 LUT를 구현하기 위해서는 많은 실리콘 면적이 소모된다. 실리콘 면적을 좀 더 효율적으로 쓰기 위해 그림 1과 같이 2:1 멀티플렉서 3개가 2단으로 구성되어 있는 멀티플렉서를 기초로

* 正會員, 韓國電子通信硏究所

(Elec. and Telecom. Research Institute)

接受日字: 1994年12月20日, 수정완료일: 1995年10月4日

한 논리모듈이 사용되고있다 [1,2]. 이 모듈은 LUT에 비해 적은 면적을 차지하기 때문에 제한된 칩 안에 많은 모듈들을 배치할 수 있어 집적도를 높일 수 있다. 멀티플렉서에 기초한 모듈의 효과적인 이용을 위한 알고리즘으로는 BDD (Binary Decision Diagram)를 이용한 것[3,4,5]과 DAG(Directed Acyclic Graph)를 이용한 것 [6] 등이 있다. 그러나 기존의 알고리즘들은 멀티플렉서의 제어단자에 연결된 제어게이트를 활용하지 못하므로 좀더 효율적인 새로운 알고리즘이 필요하다.

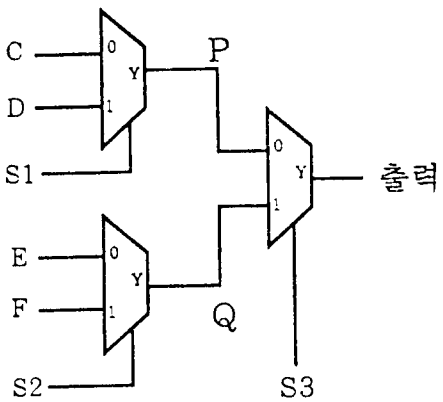

그림 1. 멀티플렉서에 기초한 논리모듈
Fig. 1. The multiplexor-based logic module.

멀티플렉서에 기초한 모듈은 입력에 따라 다양한 함수를 출력하나 LUT와는 달리 입력이 수에 따른 모든 논리함수를 구현할 수는 없다. 이 모듈이 구현할 수 있는 논리를 다양화시키기 위해 그림 1의 멀티플렉서의 제어신호 S1, S2와 S3에 적절한 제어게이트를 첨가함으로써 모듈의 논리구현 능력을 향상시킬 수 있고, 또 멀티플렉서의 입력 C, D, E및 F에 보통신호와 반전된 신호를 입력할 수 있게 함으로써 더욱 다양한 함수를 구현할 수 있다. 그러나 첨가되는 제어게이트의 실리콘 면적과 논리모듈의 논리구현 능력을 최적화 하기 위해서는 주어진 모듈이 구현할 수 있는 출력함수의 능력을 알아야 하고, 만족스럽지 못할 경우는 모듈을 변경하여 새로운 모듈이 구현할 수 있는 출력함수를 알아내는 알고리즘이 필요하다.

본 논문은 멀티플렉서에 기초한 논리모듈을 최적화하는데 있어서 주어진 논리모듈의 능력을 평가하고 mapping을 위한 library를 생성할 수 있는 방법에 대한 것이다. 즉 논리모듈에 임의의 제어게이트와 입력 게이트를 배치했을 때 모듈이 구현할 수 있는 모든 논리를 추출함으로써 논리능력의 검증과 최종적으로 library를 생성하는 방법이다. 논리모듈의 데이터 입력과 제어 입력에는 임의의 변수와 '0' 과 '1'이 가해질 수 있으므로 입력 경우의 수는 입력 단자 수의 증가에 따라 기하급수적으로 증가한다. n개의 데이터 입력과 m개의 제어 입력을 갖는 경우 각 입력에는 (n+m+2)개의 변수의 입력이 가능하므로 가능한 입력 조합의 수는 $(n+m+2)^{(n+m)}$이다. 예를 들어 입력 n=4, 제어입력 m=2인 경우 가능한 입력 조합의 경우의 수는 262,144 이다. 이 모든 경우를 library하기에는 현실적으로 불가능하다. 그러나 매핑이 가능한 최대 변수의 수와 최대 product term의 수를 제한하고, 입력의 조합이 다르면서도 같은 논리를 구현하는 함수, 입력의 변수 순서를 바꿈으로 같은 논리가 되는 함수를 제거하면 출력 함수의 수가 현저하게 줄어들 수 있다.

본 논문의 구성은 다음과 같다. Ⅱ장에서는 멀티플렉서를 기본으로 한 논리모듈에서 최대 변수와 최대 product term의 수를 제한함에 따라 입력의 조합을 추출하는 알고리즘을 설명하고, Ⅲ장에서는 출력된 함수의 논리 최적화와 입력 변수의 순서가 바뀐 동일 함수를 제거하여 최종적인 library를 구성하는 방법에 대해 설명한다. Ⅳ장은 생성된 library를 이용한 매핑 결과를 설명하고 결론을 맺는다.

Ⅱ. 입력의 경우 수

1. 멀티플렉서에 기초한 논리모듈의 논리 능력

그림 2의 2:1 멀티플렉서의 출력함수 F=AS′ + BS 이다. 제어신호 S에 논리게이트가 사용되면 출력함수의 변수의 수와 구현논리의 수가 증가한다. 예를 들면 C와 D를 갖는 2입력 AND 게이트가 제어게이트로 사용될 경우 출력함수는 식(1)과 같다.

$$F = A(CD)' + BCD = AC' + AD' + BCD \qquad (1)$$

즉 변수는 4개로 증가하고 product term은 3개로 증가하였다. 멀티플렉서는 '0' 과 '1'의 입력 단자가 있는데 위의 예에서 보면 '0'단자 A를 갖는 product term이 2개로 증가하였고 '1'단자 B를 갖은 product term은 1개 그대로 이다. 반면 AND 게이

트 대신 OR 게이트가 쓰일 경우 출력함수 F는 식(2)와 같다.

$$F = A(C+D)' + B(C+D) = AC'D' + BC + BD \quad (2)$$

이 경우도 AND 제어게이트인 경우와 마찬가지로 product term이 3개로 증가하였으나 '1'단자 B를 갖는 product term이 2개로 증가하였다. 같은 방법으로 제어게이트가 XOR 게이트인 경우는 '0' 와 '1' 입력 단자의 변수를 포함하는 product term은 각각 2개씩 증가하여 product term이 4개가 된다. 즉 product term의 증가는 제어게이트의 종류에 따라 다르게 된다.

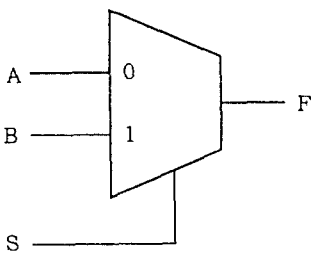

그림 2. 2-입력 멀티플렉서
Fig. 2. Two-input multiplexor.

입력 변수가 제어게이트에 사용된 변수와 다를 경우(이때의 입력 변수를 독립변수라고 칭한다)는 식 (1)과 (2)에 따르나, 입력 변수가 제어게이트에 사용된 변수와 같을 경우(이때의 입력 변수를 제어변수라고 칭한다)는 변수의 증가와 product term의 증가가 달라진다. 표 1은 이러한 관계를 정리한 표로서 '0'과 '1'은 멀티플렉서의 입력 단자를 의미하며, n은 제어게이트의 입력 갯수이다. 독립변수는 제어게이트에 사용되지 않은 논리변수로서 논리 '1'도 포함한다. 제어변수는 제어게이트에 사용된 논리변수를 의미한다. 독립변수와 제어변수가 혼용된 경우는 혼합변수라 부른다. 표 1에서와 같이 제어게이트가 n-입력 AND 게이트 일때, AND 게이트는 항상 하나의 product term만을 생성시키므로 멀티플렉서의 '0' 입력 단자에는 n개의 product term이, '1' 입력 단자에는 1개의 product term이 증가한다. 이때 입력변수가 독립변수이면 같은 수의 product term이 증가하고, 멀티플렉서의 '0' 입력 단자로 제어변수가 입력된 경우는 하나의 term이 제거되므로 n-1개의 product term이 증가한다. 제어게이트가 OR인 경우는 입력변수가 독립변수인 경우는 제어게이트가 AND인 경우와 멀티플렉서의 입력단자가 서로 바뀐 경우이다. 그러나 입력변수가 제어변수인 경우 제어게이트가 AND인 경우와 변수가 반전되는 효과가 있으므로 반전된 입력의 경우와 같다. N-입력 EX-OR의 출력의 product term의 수는 $2^{(n-1)}$개 이므로, 입력변수가 독립 변수의 경우 멀티플렉서의 입력단자에 관계없이 같은 수 만큼 증가하나, 입력변수가 제어변수인 경우 EX-OR의 출력이 임의의 변수의 반전된 경우의 군과 반전되지 않은 군으로 나누어 지므로, 증가되는 product term은 $2^{(n-2)}$가 된다.

표 1. 제어게이트에 따른 product term의 증가
Table 1. The increase of product terms according to the control gates.

Multiplexor 입력	변수의 종류	AND	OR	XOR
'0' 입력단자	독립변수	n	1	$2^{(n-1)}$
	제어변수	n-1	0	$2^{(n-2)}$
	반전된 제어변수	1	1	$2^{(n-2)}$
'1' 입력단자	독립변수	1	n	$2^{(n-1)}$
	제어변수	1	1	$2^{(n-2)}$
	반전된 제어변수	0	n-1	$2^{(n-2)}$

그림 1과 같이 2:1 멀티플렉서가 2단으로 되어있는 경우 제어게이트의 종류와 멀티플렉서의 입력변수에 따른 product term의 증가는 다음과 같이 설명된다. 예를 들어 제어게이트 조합이 n input AND-m input OR (1단 제어게이트는 AND 게이트이고, 2단 제어게이트는 OR 게이트)이고 멀티플렉서의 모든 입력 단자에 독립변수가 입력될 경우 그림 1의 중간 노드 P에는 n+1의 product term이, Q에는 n+1의 product term이 생긴다. 이때 입력 단자 C에 의해 생기는 product term이 n개, D에 의한 것이 1개, E에 의한 것이 n개 그리고 F에 의한 것이 1개이다. 두번째 단의 제어게이트가 OR게이트인 경우는 표 1에서와 같이 '0'단자에는 1개와 '1'단자에는 m개이므로 최종출력 O에는 (n+1)*(m+1)의 product term이 생기게 된다.

이와 같은 방법으로 모든 제어게이트의 조합의 경우의 생성되는 product term의 수를 표 2에 정리하였다. 표 2는 2단으로 구성된 2-입력 멀티플렉서들의 모

든 입력에 정해진 변수가 인가됐을 때 발생되는 product term의 증가를 나타낸 것이다. 모든 입력에 독립변수가 가해질 경우 (즉 입력 변수가 1단과 2단의 제어게이트에 사용된 변수와 다를 경우) 표 1의 독립변수에 따른 product term의 증가 항을 곱한 값만큼 증가한다. 제 1단 제어변수의 경우 (입력 변수가 1단의 제어게이트에 사용된 변수와 같을 경우)는 1단에서는 제어변수이지만 2단에서는 독립변수이므로 표 2의 1단의 제어변수 증가 항과 2단의 독립변수 증가항의 곱으로 나타나고 제 2단 제어변수의 경우(입력 변수가 2단의 제어게이트에 사용된 변수와 같을 경우)는 제1단 제어변수의 경우의 반대가 된다.

표 2. 제어게이트 조합에 의한 product term 증가

Table 2. The increase of product terms according to the combination of control gates.

제어게이트	제1단 / 제2단	독립변수 / 독립변수	제어변수 / 독립변수	반전된 제어변수 / 독립변수	독립변수 / 제어변수	독립변수 / 반전된 제어변수
AND-OR		$(n+1)*(m+1)$	$n*(m+1)$	$(m+1)$	$n+1$	$m*(n+1)$
AND-XOR		$(n+1)*2^m$	$n*2^m$	2^m	$(n+1)*2^{m-1}$	$(n+1)*2^{m-1}$
OR-XOR		$(n+1)*2^m$	2^m	$n*2^m$	$(n+1)*2^{m-1}$	$(n+1)*2^{m-1}$
AND-AND		$(n+1)*(m+1)$	$n*(m+1)$	$(m+1)$	$m*(n+1)$	$(n+1)$
XOR-XOR		2^{n+m}	$2^{(n+m)}$	2^{n+m-1}	2^{n+m}	2^{n+m-1}
OR-OR		$(n+1)*(m+1)$	$m+1$	$n*(m+1)$	$n+1$	$m*(n+1)$

n ~ 제 1단 제어게이트 입력수, m ~ 제 2단 제어게이트 입력수

제어게이트의 종류와 입력 단자 수가 결정되면 표 2로부터 product term의 총 수를 알 수 있다. 예를 들어 제어게이트가 AND-OR 이고, n이 2, m이 3, 입력변수가 모두 독립변수이면 총 12개의 product term이 생긴다. 그러나 제 1단의 제어변수가 사용되면 8개의 product term이, 제 2단의 제어변수가 사용되면 3개의 product term만이 발생된다. 역으로 추론하여 product term의 수를 제한하면 입력의 변수도 제한된다. 제 1단과 2단의 제어 게이트가 뒤바뀐 경우 제 1단 제어변수와 제 2단 제어변수 항이 바뀌게된다. 표 3은 입력 단자의 위치와 변수의 종류에 따른 2단 2-입력 멀티플렉서의 출력의 product term의 증가를 보여준다. 표 3은 제어게이트가 AND-OR인 경우 각 입력 단자의 변수입력이 출력의 product term 증가에 미치는 영향을 나타낸 것이다.

표 3. AND-OR 게이트 경우 product term 갯수

Table 3. The number of product terms in case of AND-OR control gate combination.

입력단자		변수종류	제1단 출력 P	제1단 출력 Q	제2단 출력 O	Minterm의 총 수: 독립변수	제1단 제어변수	제2단 제어변수
C	'0'	독립변수	n	-	1	n	n-1	0
		제어변수	n-1	-	0			
D	'1'	독립변수	1	-	1	1	1	0
		제어변수	1	-	0			
E	'0'	독립변수	-	n	m	n*m	m*(n-1)	n
		제어변수	-	n-1	1			
F	'1'	독립변수	-	1	m	m	m	1
		제어변수	-	1	1			

표 4. AND-OR 게이트 경우 반전된 입력 변수에 따른 product term 갯수

Table 4. The number of product terms of inversion inputs in case of AND-OR control gates combination.

입력단자		변수종류	제1단 출력 P	제1단 출력 Q	제2단 출력 O	Minterm의 총 수: 독립변수	제1단 제어변수	제2단 제어변수
C	'0'	독립변수	n	-	1	n	1	n
		제어변수	1	-	1			
D	'1'	독립변수	1	-	1	1	0	1
		제어변수	0	-	1			
E	'0'	독립변수	-	n	m	n*m	m	n* (m-1)
		제어변수	-	1	m-1			
F	'1'	독립변수	-	1	m	m	0	m-1
		제어변수	-	0	m-1			

1단 출력 P, Q는 표 1의 AND 제어게이트에 따라 product term이 증가하고, P는 2단 멀티플렉서 '0' 단자 입력에, Q는 2단 멀티플렉서 '1' 단자입력에 인가되므로 2단의 제어게이트로 인한 product term의 증가는, 입력 단자 C와 D의 경우는 표 3의 '0', E와 F는 '1' 입력 단자의 product term 증가를 따르게된다. C 단자에 독립변수가 인가되면 제 1단의 독립변수 증가 항과 제2단의 독립변수 증가항의 곱이 출력함수의 독립변수 증가량이 되고, 제1단의 제어변수가 인가되면 제1단의 제어변수 증가 항과 제2단의 독립변수 증가항의 곱이 출력함수 제1단 제어변수 증가량이 된다. 여기에 제2단의 제어변수가 인가되면 제1단의 독립변수 증가 항과 제2단의 제어변수 증가항의 곱이 출력함수의 제2단 제어변수 증가량이 된다. 나머지 입력 단

자도 위와 같은 방법으로 계산된다. 멀티플렉서의 입력 단자에는 반전된 제어게이트의 입력도 가능한데 표 4는 반전된 제어변수가 각 입력에 인가됐을 때의 product term의 증가를 보여준다.

2. 입력의 경우수 추출 알고리즘

입력 경우 수를 추출하기 위해서는 제어게이트의 종류에 따라 표 3과 표 4와 같이 각 입력 단자에 입력되는 변수의 종류에 따른 product term의 증가표를 작성하고, 그 표에서 product term의 갯수가 정해진 수를 넘지 않는 모든 경우의 수를 추출한다.

표 5. 2-입력 AND와 3-입력 OR의 제어게이트인 경우 입력단자에 인가된 변수의 종류에 따른product term의 증가

Table 5. The increase of product term according to the input variables in case of 2-input AND and 3-input OR control gates.

	I	C1	CB1	C2	CB2
C	2	1	2	0	2
D	1	1	0	0	1
E	6	3	6	2	4
F	3	3	0	1	2

표 5는 제1단 제어게이트가 2입력 AND이고 제2단 제어게이트가 3입력 OR인 경우에 C, D, E와 F 입력 단자에 입력되는 입력변수의 종류에 따라 product term의 증가를 나타낸 것이다. 표 5에서 독립변수는 I, 제1단 제어변수와 반전된 제어변수는 각각 C1과 CB1, 제2단 제어변수와 반전된 제어변수는 각각 C2와 CB2로 표시된다.

표 5에서 최대 product term의 갯수를 8개로 제한하면 가능한 입력 경우는 다음과 같은 논리로 추출된다. 서로 다른 입력 패턴을 발생시키기 위해 입력 단자들에 논리값 '0'의 입력 수에 따라 구분하였다. 다음의 입력 패턴에서

- I, C1, CB1, C2, CB2 앞에 붙는 숫자는 product term의 증가 갯수
- '(' 와 ')' 괄호 안은 C, D, E, F 입력 단자에 입력될 변수조합
- '{' 와 '}' 괄호 안은 각 입력 단자에 입력가능한 변수의 종류를 나열 등을 나타낸다.

1. 논리값 '0' 입력이 없는 경우 :

(1C1, {1I, 1C1, 1CB2}, 2C2, {3I, 3C1, 1C2, 2CB2})
(1C1, {1I, 1C1, 1CB2}, 4CB2, {1C2, 2CB2})
({2I, 2CB1, 2CB2}, {1I, 1C1, 1CB2}, {3I, 3C1, 2C2}, {1C2, 2CB2})
({2I, 2CB1, 2CB2}, {1I, 1C1, 1CB2}, 4CB2, 1C2)

2. 논리값 '0'입력이 1개 있는 경우:

('0', {1I, 1C1, 1CB2 }, {6I, 3C1, 3CB1, 2C2, 4CB2}, 1C2)
('0', {1I, 1C1, 1CB2 }, {3C1, 3CB1, 2C2, 4CB2}, 2CB2)
('0', {1I, 1C1}, {3C1, 3CB1, 2C2, 4CB2}, {3I, 3C1})
({2I, 2CB1, 2CB2}, '0', {3C1, 3CB1, 2C2, 4CB2}, {1C2, 2CB2})
({2I, 2CB1, 2CB2}, '0', {3C1, 3CB1, 2C2}, {3I, 3C1})
(1C1, '0', {6I, 3C1, 3CB1, 2C2, 4CB2}, 1C2)
(1C1, '0', {3C1, 3CB1, 2C2, 4CB2}, {3I, 3C1, 2CB2})
({2I, 1C1, 2CB1, 2CB2}, {1I, 1C1, 1CB2},'0', {3I, 3C1, 1C2, 2CB2})
({2I, 2CB1, 2CB2}, {1I, 1C1, 1CB2}, {3C1, 3CB1, 2C2, 4CB2}, '0')
(1C1, {1I, 1C1, 1CB2}, {6I, 3C1, 3CB1, 2C2, 4CB2}, '0')

3. 논리값 '0' 입력이 2개 있는 경우

('0', '0', {6I, 3C1, 3CB1, 2C2, 4CB2}, {1C2, 2CB2})
('0', '0', {3C1, 3CB1, 2C2, 4CB2}, {3I, 3C1})
('0', {1I, 1C1, 1CB2}, '0', {3I, 3C1, 1C2, 2CB2})
('0', {1I, 1C1, 1CB2}, {6I, 3C1, 3CB1, 2C2, 4CB2}, '0')
({2I, 1C1, 2CB1, 2CB2}, '0', '0', {3I, 3C1, 1C2, 2CB2})
({2I, 1C1, 2CB1, 2CB2}, '0', {6I, 3C1, 3CB1, 2C2, 4CB2}, '0')
({2I, 1C1, 2CB1, 2CB2}, {1I, 1C1, 1CB2}, '0', '0')

4. 논리값 '0' 입력이 3개 있는 경우

('0', '0', '0', {3I, 3C1, 1C2, 2CB2})
('0', '0', {6I, 3C1, 3CB1, 2C2, 4CB2}, '0')
('0', {1I, 1C1, 1CB2}, '0', '0')
({2I, 1C1, 2CB1, 2CB2}, '0', '0', '0')

위의 조합에서 {}안에 있는 입력의 조합은 모두 가능하고 ()안의 독립변수 I를 배치하는 방법은 독립변수간의 순열의 경우를 발생시키고 입력 단자에 반전된 입력이 가능한 경우는 각 독립변수의 반전되는 경우도 발생시킨다. 제어변수인 경우는 C1과 C2 위치에 제어변수를 배치하고 입력 단자에 반전된 입력이 가능한 경우는 CB1과 CB2 위치에 반전된 제어변수를 배치한다.

제어 게이트에 반전 입력이 가능한 경우도 있는데 이런 경우에는 제어 게이트 변수군을 반전이 된 제어변수(C1B, C2B)와 반전이 아닌 제어변수(C1, C2) 2개의 군으로 확장된다. 즉 제1단 제어변수는 [{C1},{C1B}]로 제2단 제어변수는 [{C2}, {C2B}]로 나눌 수 있으므로 각 단자의 product term 증가는 [{C1}, {C1BB}, {CB1}, {C1B}, {C2},

{C2BB}, {CB2}, {C2B}]의 변수 군으로 확장된다. C1BB와 C2BB는 각각 1단과 2단의 반전된 제어변수의 반전된 변수를 나타낸다. 확장된 변수군에서 [{C1}, {C2}, {CB1}, {CB2}]들에 의한 product term의 증가와 [{C1BB}, {C2BB}, {C1B}, {C2B}]들과 순서대로 같으므로 {C1}은 [{C1}, {C1BB}]로, {C2}는 [{C2}, {C2BB}]로, {CB1}은 [{CB1}, {C1B}]로, {CB2}는 [{CB2}, {C2B}]로 확장된다. 멀티플렉서 입력단자에 반전된 입력이 가능하지 않은 단자는 위의 확장된 변수군으로 부터 [{CB1}, {CB2}, {C1BB}, {C2BB}]의 변수군을 소거하고, 제어 게이트 입력에 반전된 단자가 없으면 [{C1B}, {C2B}]의 변수 군을 소거한다.

Ⅲ. 출력함수의 동일 함수 제거

II장의 알고리즘으로 생성된 출력 함수들은 논리적으로 같은 함수를 포함하는데 이 장에서는 그러한 함수들을 제거하는 방법에 관한 것이다. 동일 함수 제거는 먼저 내부에서 개발된 논리 최적화 프로그램을 사용하여 출력함수의 논리 최적화를 수행한 후 동일 함수를 제거한다. 멀티플렉서에 기초한 논리모듈은 입력단의 변수 순서에 따라 논리적으로는 같으나 표현이 다른 함수를 출력한다. 따라서 논리 최적화에 따른 동일 함수 제거 후에도 입력 변수의 치환으로 동일 함수가 되는 것들을 제거해야한다. 예를 들면 함수 F1=abc+bd와 F2=abc+ae는 함수 F1의 변수 a를 b로, b를 a로, 또 d를 e로 치환하면 함수 F1은 F2와 동일 함수가 된다. 변수 치환을 통해서 동일 함수를 제거하는 방법은 다음과 같다.

1. 논리 최적화를 거친 출력 함수들을 같은 변수의 수와 같은 sum-of-product의 색인을 갖는 것들끼리 분리한다. 즉 색인은 (Vn, i1, i2, ...)인데 Vn은 변수의 수이고, (i1, i2, ...)는 각 product term의 변수 갯수이다.
2. 같은 색인을 갖는 함수들 중에서 같은 inverter 조합을 갖는 함수끼리 분리한다. 예를 들면 F1=abc+a′b′c′의 inverter 조합은 (0,3)이고, F2=abc′+a′b′c는 (1,2)이므로 서로 다른 함수이다.
3. 순서 1 과 2를 거쳐 분류된 함수 중에서 치환으로 동일함수가 되는 함수들을 제거하기 위해서는 변수 빈도 색인을 이용한다. 변수 빈도 색인은 함수의 literal들의 빈도 수를 literal 별로 표시한 것이다. 즉 함수 F1=abc+a′cd+abd의 변수 빈도 색인은 {(2,2,2) (1,2,2) (2,2,2)}이다. 반전된 변수는 새로운 빈도 변수로 간주한다. ()안의 색인의 조합이 같고 {}안의 블록의 조합이 같으면 동일 함수이다. 예를 들면 F2=abd+acd′+bcd의 변수 빈도 색인은 {(2,2,2) (2,2,1) (2,2,2)}인데 함수 F1의 변수 빈도 색인과 비교하면 가운데 블록의 색인의 순서만 바뀌 있음을 알 수 있다. 함수 F2에서 빈도 수 1에 해당하는 빈도 변수 d′를 함수 F1의 a′로 치환하면 함수 F2는 함수 F1과 동일 함수가 됨을 볼 수가 있다.

이렇게 출력된 최종함수들을 mis-II[7]상에서 각 함수들 서로간에 치환(resubstitution)을 수행해본 결과 각각이 독립 함수임을 알 수 있었다.

Ⅳ. library 생성 및 mapping 결과

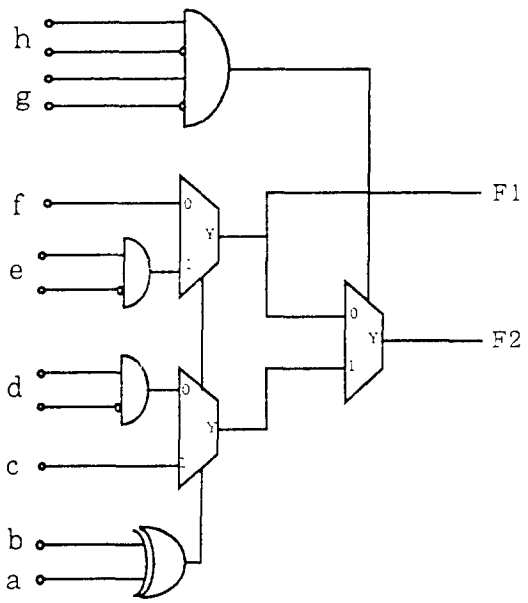

그림 3. 제어게이트가 XOR-AND인 멀티플렉서에 기초한 논리모듈
Fig. 3. The multiplexor-based logic module with XOR-AND control gates.

위에서 설명한 library 생성 알고리즘으로 그림 3의 멀티플렉서에 기초한 논리모듈에 대해 실험을 하였다.

```
.ios
INORDER=A B C D E F G H;
OUTORDER=FF1 FF2 FF3 FF4 FF5 FF6 FF7 FF8 FF9 FFA FFB FFC FFD FFE FFF     FFG FFH FFI;
.debug
.functions
FF1=(E&(!A&B|A&!B)|F&!(!A&B|A&!B))&(!G|!H)|(C&(!A&B|A&!B)|D&!(!A&B|A&!B))&(G&H);
FF2=(E&(!A&B|A&!B)|F&!(!A&B|A&!B))&(G|!H)|(C&(!A&B|A&!B)|D&!(!A&B|A&!B))&(!G&H);
FF3=(E&(!A&B|A&!B)|F&!(!A&B|A&!B))&(!G|H)|(C&(!A&B|A&!B)|D&!(!A&B|A&!B))&(G&!H);
FF4=(E&(!A&B|A&!B)|F&!(!A&B|A&!B))&(G|H)|(C&(!A&B|A&!B)|D&!(!A&B|A&!B))&(!G&!H);
FF5=(E&A|F&!A)&(!G|!H)|(C&A|D&!A)&(G&H);
FF6=(E&A|F&!A)&(!G|H)|(C&A|D&!A)&(G&!H);
FF7=(E&A|F&!A)&(G|!H)|(C&A|D&!A)&(!G&H);
FF8=(E&A|F&!A)&(G|H)|(C&A|D&!A)&(!G&!H);
FF9=(E&!A|F&A)&(!G|!H)|(C&!A|D&A)&(G&H);
FFA=(E&!A|F&A)&(!G|H)|(C&!A|D&A)&(G&!H);
FFB=(E&!A|F&A)&(G|!H)|(C&!A|D&A)&(!G&H);
FFC=(E&!A|F&A)&(G|H)|(C&!A|D&A)&(!G&!H);
FFD=(E&A|F&!A)&(!G)|(C&A|D&!A)&(G);
FFE=(E&A|F&!A)&(G)|(C&A|D&!A)&(!G);
FFF=(E&!A|F&A)&(!G)|(C&!A|D&A)&(G);
FFG=(E&!A|F&A)&(G)|(C&!A|D&A)&(!G);
FFH=(E&(!A&B|A&!B)|F&!(!A&B|A&!B))&(!G)|(C&(!A&B|A&!B)|D&!(!A&B|A&!B))&(G);
FFI=(E&(!A&B|A&!B)|F&!(!A&B|A&!B))&(G)|(C&(!A&B|A&!B)|D&!(!A&B|A&!B))&(!G);

.outputs
FF1 FF2 FF3 FF4 FF5 FF6 FF7 FF8 FF9 FFA FFB FFC FFD FFE FFF FFG FFH FFI

.names
A B C D E F G H

.values
a b a a g g g h
a b a !a g g g h
a b a !a !g g g h
a b a a !g g g h
a b a g a g g h
a b a g !a g g h
a b a g g a g h
a b a !g g a g h
a b a !g !g a g h
a b a g !g a g h
a b g a a g g h
a b g !a a g g h
a b g !a !a g g h
a b g a !a g g h
.end
```

그림 4. 출력 함수를 얻기 위한 LFGEP의 입력 파일
Fig. 4. The input file of LFGEP to get the boolean output fuction.

그림 3의 논리모듈은 첫 단의 제어게이트가 2-입력 XOR이고, 두 번째 단의 제어게이트는 4-입력 AND이다. 4개의 멀티플렉서 입력 중 2개의 입력에 반전된 입력을 가능하게 하기 위해 한쪽 입력이 반전된 2-입력 AND 게이트가 연결되어 있다.

제어게이트의 입력 상태에 따라 다른 논리의 함수가 출력된다. 그림 4는 논리 최적 프로그램의 입력 파일을 보여 주는데 함수 FF1에서 FFI는 제어게이트의 입력 변화에 따른 함수들이다. 하나의 입력 패턴이 FF1에서 FFI까지의 함수를 출력한다. 이렇게 출력된 함수들은 3장의 알고리즘으로 동일함수는 제거되고 최종적으로 library가 생성된다. 그림 3의 논리모듈에 최대 product term의 수를 8개로, 최대 변수의 수를 5개로 제한하여 얻어진 총 함수의 수는 4,000개가 넘었으나 Ⅲ장의 알고리즘으로 걸러져서 최종적으로 library로 등록된 함수는 734개였다. 그림 5는 변수

별로 생성되어진 library 함수들의 숫자이다. 그림에서 볼 수 있듯이 거의 모든 4 변수 함수들을 구현할 수 있다.

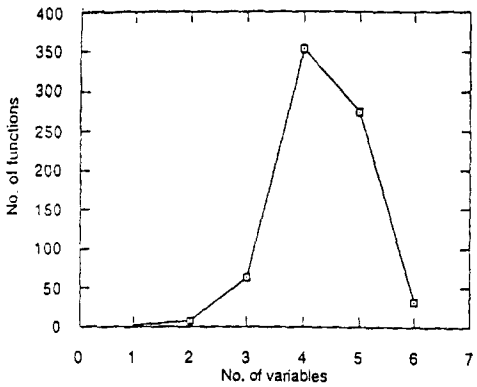

그림 5. 그림 3의 논리모듈로부터 생성된 library의 변수별 함수분포도

Fig. 5. The number of logic functions gererated from the logic module of FIG. 3.

표 6. MCNC benchmark 결과

Table 6. The results of MCNC benchmark.

CIRCUIT	로직 모듈의 수				CIRCUIT	로직 모듈의 수			
	MIS_PGA	Amap	MIM	ours		MIS_PGA	Amap	MIM	ours
5xp1	39	42	37	30	misex1	27	25	18	22
9sym	27	106	90	18	misex2	47	47	37	40
alu4	156		-	113	rd53	10			8
apex1	590			439	sqrt8	30			19
apex2	91			66	squar5	30			24
apex3	617			551	xor5	5			2
apex4	1027			1047	vg2	49	44	30	34
apex5	434			98	sao2	81	56	59	56
b12	42			30	seq	509			383
bw	84	83	68	69	ex1010	1391			1048
clip	51	60	43	39	spla	180			142
con1	14			9	rd73	17	32	25	12
cordic	29			15	rd84	23			18
cps	416			334	table3	498			374
duke2	194	175	149	145	table5	427			324
e64	95	105	95	95	misex3	247			189
ex1p	281	-	-	201	misex3c	271	-	-	193
ex5p	156			127	t481	17			11
inc	59			47	pdc	184			137

실제 회로의 mapping은 생성된 library를 SYNOPSYS에 등록시켜서 실행하였는데 표 6은 MCNC benchmark[8]를 기존의 BDD를 이용한 mapping방식인 MIS_PGA[9]의 act_map과 MIM[10]의 수행 결과중 가장 좋은 결과와 비교하였고, DAG를 이용한 mapping 방식인 Amap[6]과도 비교하였다. 표 6의 각 항목의 숫자는 각 회로를 구현하기에 필요한 논리모듈의 갯수를 나타낸 것으로 본 논문이 제안한 방법이 논리모듈을 훨씬 효율적으로 이용함을 알 수 있다.

V. 결 론

본 논문의 알고리즘은 현재 FPGA의 논리모듈의 기본이 되는 2:1 멀티플렉서 3개가 두 단으로 연결되어 있는 구조의 논리능력을 알기 위해 각 입력 단자에 입력변수를 배치하는 방법에 대한 것이다. 이 구조에서는 멀티플렉서를 제어하는 제어게이트의 종류와 크기, 또 멀티플렉서 입력 단자의 반전의 가능 유무에 따라 구현할 수 있는 논리능력이 크게 달라지는데, 본 논문의 알고리즘은 주어진 구조에 대한 모든 가능한 입력변수를 발생시켜 사용자가 구조를 변경시키면서 논리능력의 변화를 볼 수 있으므로 원하는 형태의 구조를 얻을 수 있게 한다. 즉 사용자가 논리능력과 실리콘 면적 간의 최적점을 찾을 수 있게 한다.

참 고 문 헌

[1] Khaled A. El-Ayat, et al., "A CMOS Electrically Configurable Gate Array," IEEE J. Solid-State Circuits, vol. 24, No. 3, pp. 752-762, June 1989.

[2] Very High Speed FPGAs Data Book, Quicklogic Corp., Santa Clara, CA, pp. 1-10, 1994.

[3] R. Murgai, R. K. Brayton, A. Sangiovanni-Vincentelli, "An Improved Synthesis Algorithm for Multiplexor-based PGAs," ACM/SIGDA First International Workshop on Field-Programmable Gate Arrays, Berkeley, CA, pp. 97-102, Feb. 12-15, 1992.

[4] S. Ercolani and G. De Micheli, "Technology Mapping for Electrically Programmable Gate Arrays," Proc. 28th

Design Automation Conference, San Francisco, CA, pp. 234-239, June 17-21, 1991.

[5] A. Bedarida, S. Ercolani, G. De Micheli, "A New Technology Mapping Algorithm for the Design and Evaluation of Fuse/Antifuse-based Field-Programmable Gate Arrays," ACM/SIGDA First International Workshop on Field-Programmable Gate Arrays, Berkeley, CA, pp. 103-108, Feb. 12-15, 1992.

[6] K. Karplus, "Amap: a Technology Mapper for Selector-based Field-Programmable Gate Arrays," Proc. 28th Design Automation Conference, San Francisco, CA, pp. 169-172, June 17-21, 1991.

[7] R. Brayton et al, "MIS: A Multiple-Level Logic Optimization System," IEEE Transactions on Computer Aided Design, Vol. CAD-6, No. 6, November 1987, pp 1062-1081.

[8] S. Yang, "Logic Synthesis and Optimization Benchmarks User Guide - Version 3.0," Microelectronic Center of North Carolina, Durham, NC, Jan. 1991.

[9] Ewald Detjen et al., "Technology Mapping in MIS," ICCAD-87, November 1987, pp 116 - 119.

[10] Mahesh Mehendale, "MIM : Logic Module Independent Technology Mapping for Design and Evaluation of Antifuse-based FPGAs," Proc. 30th Design Automation Conference, Dallas, Texas, pp 219 - 223.

저 자 소 개

曺 漢 鎭(正會員)

1960년 7월 8일생. 1982년 2월 한양대학교 전자공학과 졸업. 1987년 New Jersey Institute of Technology 전자공학과 석사학위 취득. 1992년 University of Florida 전자공학과 박사학위 취득 (고속 소자 및 회로 모델링 전공). 1992년 11월~현재 한국전자통신연구소 선임연구원.

朴 仁 學(正會員)

1958년 3월 6일생. 1980년 2월 고려대학교 전자공학과 졸업. 1983년 9월 고려대학교 대학원 전자공학과 졸업(공학석사). 1992년 7월 프랑스 국립폴리테크닉 연구소 (INPG) 졸업(공학박사). 1982년 2월~현재 한국전자통신연구소 근무. 현재 자동설계연구실 실장. 주관심분야는 컴퓨터 그래픽스, 상위수준합성, VLSI CAD 시스템 등임.

裵 英 煥(正會員)

1962년 10월 29日生. 1985년 2월 한양대학교 전자공학과 졸업. 1987년 2월 한양대학교 대학원 전자공학과 졸업(공학석사). 1987년 2월~현재 한국전자통신연구소 자동설계연구실 선임연구원. 주관심 분야는 VLSI CAD, 컴퓨터 그래픽스 및 컴퓨터 아키텍춰 등임.