論文98-35C-7-9

코드할당에의한 다치논리함수의 모듈러 함수분해에 관한 연구

(A Modular function decomposition of Multiple-Valued Logic functions using code assignment)

崔 在 碩 *, 朴 春 明 **, 成 賢 慶 ***, 朴 承 用 ****, 金 興 壽 *

(Jai-Sok Choi, Chun-Myoung Park, Hyeung-Kyung Sung, Seung-Yong Park, and Heung-Soo Kim)

요 약

본 논문은 다치 논리 함수의 함수 분해 방법과 입력변수 처리 방법에 관한 모듈러 설계기법을 제시하였다. 부분 함수 분해방법으로서 행 분할 변수 중 1-열 변수의 입력 값에 따라 함수를 분할하여 column multiplicity를 구한다. 이 경우 기존의 방식보다 동일 부분 함수가 증가하므로 간략화 과정을 통해 더욱 간단한 회로를 얻을 수 있다. 또한, 제시된 부분 함수는 단일 T-gate를 사용하므로 모듈 라이브러리 방식에서의 제어함수를 생략할 수 있다. 입력 변수를 처리방법은 입력 변수에 코드를 할당하는 look up 테이블을 제안하여 입력 변수의 복잡 도를 감소시켰다. 이 경우, 코드 길이와 코드 할당 테이블 수는 부분 함수의 수가 증가함에 따라 증가하게 되는데 각 부분 함수 사이에는 입력 변수를 서로 공유하는 경우가 있으므로 이를 간략화할 수 있다. 함수에 적용한 결과 주어진 함수가 비대칭, 불규칙인 경우 기존의 방식에 비해 내부결선이 약 12%정도, T-gate수는 16% 정도 감소되어 더욱 간단한 회로설계가 이루지는 장점이 있음을 보였다.

Abstract

This paper presents modular design techniques of Multiple-Valued Logic functions about the function decomposition method and input variable management method. The function decomposition method takes advantage of the property of the column multiplicity in a single-column variable partitioning. Due to the increased number of identical modules, we can achieve a simpler circuit design by using a single T-gate, which can eliminate some of the control functions in the module library types. The input variable management method is to reduce the complexity of the input variables by proposing the look up table which assign input variables to a code. In this case as the number of sub-functions increase the code-length and the size of the code-assignment table grow. We identify some situations where shared input variables among sub-functions can be further reduced by a simplification technique. Accordign to the result of adapting this method to a function, We have demonstrated the superiority of the proposed methods which is being decreased to about 12% of interconnection and about 16% of T-gate numbers compare with the existing for the non-symmetric and irregular function realization.

* 正會員, 仁荷大學校 電子工學科
(Dept. of Electronic Eng. In-Ha Univ.)
** 正會員, 忠州大學校 컴퓨터工學科
(Dept. of Computer Eng. Chung-Ju National Univ.)
*** 正會員, 尙志大學校 電子計算學科
(Dept. of Computer Science Sangji Univ.)
**** 正會員, 才能大學 電子計算科
(Dept. of Computer Science Jea Neung Junier College)
接受日字:1998年3月18日, 수정완료일:1998年6月29日

Ⅰ. 서 론

VLSI 회로 설계분야는 지난 수 십년간 빠른 기술적 발전으로 소자의 집적능력을 비롯하여 기능 당 가격 비율이 크게 향상되었다. 그러나, 회로가 지나치게 집적됨으로 인해 소자가 차지하는 면적보다 소자를 연결하는 배선 및 내부 결선에 의한 면적이 전체 집적능력을 결정하는 요소로 작용하게 되었다. 이러한 배선과 내부결선 증가로 인한 문제점으로는 신호선 사이의 혼선잡음과 배선 길이에 의한 신호지연 등을 들 수 있다. 이러한 문제점을 해결하기 위해 VLSI칩 내부에 다치 논리회로를 사용하는 경우, 신호전송을 위한 배선과 모듈간의 내부결선을 줄일 수 있으므로 가장 큰 문제점인 혼선 잡음을 줄일 수 있다. 따라서, 다치 논리 회로를 사용함으로써 칩의 처리능력 및 신뢰성을 향상시킬 수 있다.[1]

함수 분해(Function Decomposition)는 복잡한 함수관계를 보다 규모가 작고, 독립된 관계를 갖는 부분함수로 나누어서 문제를 해결하려는 것으로, 시스템 분석과 설계 과정에서 문제의 복잡 도를 감소시키기 위한 동기에서 시작되었다. 즉, 주어진 함수를 설계와 분석이 용이한 부분함수로 분해한다. 분해된 부분함수를 사용하여 회로를 설계하는 경우 더욱 작은 실리콘 면적이 소요되고 입력으로부터 출력까지 보다 짧은 신호경로를 갖게 되므로 신호지연 측면에서 장점이 있다[1].

2-치 논리 함수 분해는 Ashenhurst[2]가 처음 테이블 기법(Tabular method)을 이용하여 함수 분해 가능성을 연구하여 발표하였다. 그의 방법은 n-변수 함수에 대해 $2^n-(n+2)$개의 테이블이 필요하고 회로 구현 시 실행 시간이 변수 증가에 대해 지수 함수적으로 증가하였다.[3,4] 또한, 스펙트랄 기법에 의한 함수 분해 기법은 실행 시간을 감소 시켰지만 변수가 증가함에 따라 여전히 실행 시간이 지수 함수적으로 증가하는 단점이 있다.[5] 변수 증가에 따른 실행 시간 증가의 문제점을 보완하기 위한 고속 알고리듬이 Shen[6] 등에 의해 제안되었다.

한편, 다치 논리 분야의 함수 분해는 초기에는 주로 2-치 논리 시스템의 Ashenhurst 알고리듬을 확장하는 방법을 연구하였다. 그러나 이러한 방법은 첫째, 주어진 함수의 테이블을 검색하여 결과를 얻는 것이 지극히 복잡하고 효율적이지 못하며, 둘째, 함수가 적절하게 분해되는 경우가 매우 적고, 셋째, 함수 분해가 이루어 지지않는 경우 효과적인 설계가 되지 못하는 문제점이 있으므로 새로운 방향의 연구가 요구되었다[7]. 특히, 다치 논리 함수의 경우, 부분함수 검색과정이 2치 논리 함수의 경우와 비교하여 지수 함수적으로 증가하므로 기존의 2치 논리 함수분해 방법을 확장하여 적용하는 것만으로는 한계가 있다. 따라서, 다치 논리 함수에 적합한 새로운 개념의 함수 분해 방법이 필요하게 되었다.

최근, VLSI가 실용화되고 주류를 이루게 됨에 따라, 개별 소자에 의한 회로 구현방식보다 경제적인 모듈 형태의 회로 설계 기법이 등장하게 되었고[2,4] 다치 논리 분야에서는 Post Algebra와 같은 다양한 형태의 연산자를 연구하여 이를 회로 설계에 도입하게 되었다.[8] 또한, ULM(Universal Logic Module)을 사용하여 입출력 단자 수를 줄이는 방법이 연구되었으며[9,10] Thelliez, kameyama등에 의해 T-gate와 같은 트리 구조-ULM에 관한 연구도 이루어졌으나 체계적인 회로 설계는 되지 못했다.[11-13] 테이블 기법을 이용한 단순한 형태의 함수 분해기법과 분해된 부분함수에 직접 모듈을 적용할 수 있는 새로운 개념의 모듈 설계 방법은 Fang[14]에 의해 처음 제안되었다. Fang은 다치 논리 함수를 2-변수 부분함수로 분류하여 모듈 라이브러리를 정의하고, 주어진 임의의 다치 논리 함수를 2-행 변수, 3-열 변수로 분류하여 부분 함수를 검색하는 매우 체계적이고 단순한 방법을 제시하였다. 그러나, 검색된 부분 함수와 정의된 모듈과의 매칭의 문제점으로 부분 함수와 일치하는 모듈이 없는 경우 모듈의 입력 변수를 변화시키기 위한 제어 함수를 추가해야 한다. 또한, 부분 함수의 규모가 너무 크기 때문에 최적의 회로를 설계하기가 어려운 문제점을 나타내었다.

본 논문에서는 2-행 변수로 테이블을 검색하는 방법에 있어서는 Fang의 방법과 동일하나 테이블로부터 부분 함수를 검색하는 과정과 입력 변수를 처리하는 부분에서 새로운 기법을 제시하여 기존의 문제점을 해결하고자 한다. 먼저, 부분 함수의 분해 방법에 있어서 기존의 경우, 전적으로 행 변수에 대한 CM(Column Multiplicity)만을 사용하고 있으나, 본 논문에서는 행 변수를 세분하여 CM을 검색하는 방법을 사용한다. 이 경우 동일 부분 함수를 검색할 수 있는 확률이 증가하게 된다. 모듈 구성 면에서는 기존의 모듈 라이브러리

방식을 사용하지 않고 ULM개념의 단일 T-gate를 기본 블록으로 사용하고 있으므로 부분 함수와 모듈의 매칭 문제를 해결할 수 있다.

열 분할변수를 처리하기 위해 본 논문에서는 입력 변수에 코드를 할당하는 LUT(Look Up Table)방식을 제안하여 입력 변수를 처리하기 위한 복잡 도를 감소시켰다. 이러한 방법은 Sasao[15]의 경우 고전적인 함수 분해 기법[2]을 PLA 설계에 응용하였다. Sasao의 방법은 LUT를 적용하는 대상과 응용방법에 있어서 본 논문의 방법과 차이가 있으나 코드 할당에 의한 설계 단순화 측면에서 유사하다. 그러나, 이러한 LUT는 입력 변수 처리를 단순화할 수 있는 반면, 참조해야 하는 테이블 수가 증가되는 단점이 생기게 된다.

본 연구는 모듈러 함수 분해 방법으로서 VLSI나 ULSI회로를 설계함에 있어서 지금까지의 함수 분해방법이 함수 값들 사이의 관계를 분석하여 처리하려는 것과 달리 함수 테이블에서의 기하학적인 관계를 고려하였다. 이 경우 부분함수를 얻기 위한 검색 과정이 매우 단순하고, 모든 함수에 대해서 기저에 관계없이 부분함수를 얻을 수 있다. 이러한 모듈러 설계 방식은 설계 과정을 단순화시킬 수 있고, 설계된 회로의 검사에도 매우 유리하다.

본 논문의 구성은 Ⅱ에서 기본적인 함수분해이론과 기존의 모듈러 함수분해 방법 및 새로 제안한 함수분해 방법이 나타나 있고 필요한 정의 및 정리를 제시하였다.Ⅲ에서는 제시된 함수분해 방법을 실제 회로에 적용하는 방법을 제시하고 효율적인 회로설계가 되도록 코드할당방법을 제시하였다.Ⅳ에서는 함수분해 방법에 대하여 검색횟수 및 부분함수의 개수등을 비교하였고 장단점을 논하였다. Ⅴ에서 결론을 제시하였다.

Ⅱ. 함수분해 이론

변수의 규모가 큰 함수를 회로로 구현하는 경우 직접 설계가 어렵고 대부분 동일한 함수가 포함되어 있으므로 다수의 부분함수를 조합하여 표현하고자하는 연구를 해왔다. 본 장에서는 함수를 부분함수로 표현하는 일반 이론을 간단히 나타내고, 본 논문에 적용하기 위한 함수 분해 법을 설명한다.

함수분해는 원래의 함수보다 적은 변수의 함수로 분해하여 각 함수를 독립적으로 설계할 수 있으므로 비교적 설계하기가 용이하다.

예를 들어 함수 f가 $f(x,y,z)=\exp(\cos x+\sqrt{y^2+z^2})$과 같이 주어진 경우

$v(y,z)=\sqrt{y^2+z^2}$이고 $u(x)=\cos x$, $F(u,v)=\exp(u+v)$라 하면, 그림 1과 같이 나타낼 수 있다.

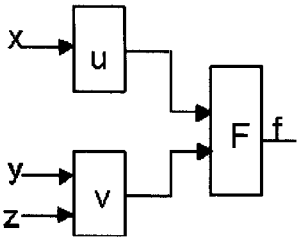


그림 1. 부분 함수에 의한 표현
Fig. 1. Represented by subfunctions.

여기서, 함수 f는 부분함수 u,v,F 로 분해(decomposed)되었다고 말한다. 그러나 대부분의 함수가 그림 1과 같이 분해되는 경우는 드물고 입력변수가 증가하는 경우 부분함수로 분해하기가 더욱 어렵다. 정의 1은 함수 테이블로부터 동일한 함수 값의 개수를 나타내는 CM을 정의한다. CM에 대한 개념은 함수분해에서 매우 중요한데, 부분 함수의 개수가 CM와 밀접한 관련이 있기 때문이다.

[정의 1[16]] 함수 분할 표(partition matrix)에서 서로 다른 Column 패턴을 CM이라 한다. 입력 변수가 $X_n=\{x_1, x_2, \ldots, x_n\}$이고, 분할이 $\lambda_A=\{A, X_n-A\}$으로 표시될 때 (X_n-A)를 입력 변수로 하는 함수 f_A를 식 (1)과 같이 나타낸다.

$$
\begin{aligned}
&f_{A1}(X_n-A)=(b_1, b_2, b_3, \ldots, b_{2^A})\\
&f_{A2}(X_n-A)=(b_1, b_2, b_3, \ldots, b_{2^A})\\
&\quad\vdots \qquad\qquad\qquad \vdots\\
&f_{A2^{(X_n-A)}}(X_n-A)=(b_1, b_2, b_3, \ldots, b_{2^A})
\end{aligned}
\tag{1}
$$

여기서, CM은 서로 다른 함수 값을 갖는 f_A의 개수를 말한다. 따라서, 함수 값이 모두 동일한 경우 CM은 1이고 모두 다른 함수 값을 갖는 경우 CM은 $2^{(X_n-A)}$를 갖는다.

[예제 1] 다음과 같이 주어진 함수 테이블에서 CM=5 이다.

		X_1 X_2		
		0 0 0	1 1 1	2 2 2
		0 1 2	0 1 2	0 1 2
X_3	0	0 1 0	1 1 2	1 2 1
	1	2 1 2	0 0 2	1 2 1
	2	1 1 1	1 1 2	1 0 1
		①②	③④	⑤

본 논문에서는 부분 함수를 분해하기 위해 주어진 함수를 열 변수와 행 변수로 분할하고 열 변수와 행 변수의 조합에 의해 함수 분할 표는 서로 다른 함수구성을 갖게 된다. 여기서, 변수 분할에 따른 함수 분할 표의 개수는 n 입력 변수에 대해 $2^{n-1}-1$ 로 주어진다. 이러한 개념을 바탕으로 본 논문에서 제안하는 모듈함수분해를 설명하고자하며 고전적인 함수분해이론에 관한 자세한 내용은 참고문헌 [21] 에 나타나 있다.

1. 모듈러 함수분해

현대의 회로설계 기술은 단일 칩 상에 수만 게이트 이상의 회로를 가지므로 그 설계 방식에 있어서도 단일 게이트 단위의 설계 방식보다 모듈 형태의 기본 블록을 사용하는 것이 더욱 경제적이다. 본 논문에서는 모듈 설계 방식에 중점을 둔 일반화된 회로 설계 방식을 제안하고자 한다.

기존의 함수 분해방법[2,5,7]이 단순분리형, 복합 분리형 혹은 비 분리형 인지의 여부를 검색하기가 매우 어려운 반면 Fang[14]의 방법은 미리 정의된 모듈 라이브러리를 갖고, 함수테이블 전체가 부분함수로 표현가능하며, 주어진 함수에 정의된 모듈을 적용할 수 있도록 부분함수로 분해한 뒤 적절한 모듈을 적용하여 단일 출력으로 간단히 회로가 구성되는 형태이다. Choi[17]의 방법은 모듈 라이브러리를 이용하는 방법에서 Fang과 유사하나, 확장된 형태의 모듈 라이브러리 사용을 제안하였다는 점에서 차이가 있다.

Kameyama[18]의 경우, 2-변수 입력을 갖는 ULM을 제안하였고 간략화 기법을 아울러 제시하였다. 그러나, 고전적 함수 분해기법을 확장한 방식으로 효율적인 설계는 되지 못 한다. 본 논문은 테이블 검색 방법에서 Fang과 동일하고, 부분 함수의 분해 방법에서는 다음과 같은 점에서 차이가 있다.

첫째, 모듈 라이브러리를 사용하지 않는다.

둘째, T-gate 단일 모듈을 사용한다.

셋째, 열 변수 입력 부분에 코드 할당 기법(LUT)을 사용한다.

네째, 모듈의 입력 값을 제어하기 위한 제어 함수를 생략하였다.

이러한 기본블록을 이용한 형태의 설계 기법은 Dietmeyer[20]의 방법에 기초를 두고 있으나, Dietmeyer의 경우 기존의 고전적 함수 분해 기법을 사용하여 함수가 적절히 분해되지 않을 경우 효율적인 회로설계가 이루어지지 못하는 단점이 있다.

2. 기존의 부분 함수 분해 법 호 역

기존 방법에서의 함수 표현을 보면 주어진 함수가

$f(x_1, x_2, x_3, x_4) = x_1 x_3 + x_2 x_3 + x_1 x_4 + x_2 x_4$ 인 경우

$f(x_1, x_2, x_3, x_4) = F(\varphi_1(x_1, x_2), \varphi_2(x_3, x_4))$ 로 표현된다.

여기서, $F(A,B) = \varphi_1 \cdot \varphi_2$

$\varphi_1(x_1, x_2) = x_1 + x_2$

$\varphi_2(x_3, x_4) = x_3 + x_4$

로 분해되고 이를 회로로 나타내면 그림 2와 같다.

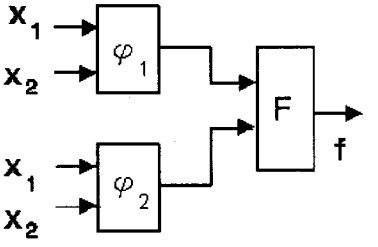


그림 2. 복합 분리형 함수 분해
Fig. 2. Compound disjunctive decomposition.

함수 분해는 다수의 입력 변수를 갖는 함수를 보다 적은 입력 변수를 갖는 함수로 분해하는 것으로 이미 정의한 바 있다. 그러나, 입력 변수의 수가 15 - 50, 출력이 10 - 100 정도인 경우 기존의 함수 분해 이론을 적용하여 회로를 설계하는 데에는 두 가지 문제점이 있다.[15] 첫째, 데이터 처리 시간과 메모리 용량인데 서로 다른 함수로 분해되는 경우의 수가 n-변수인 경우 2^n이고 함수 분해 테이블의 크기 또한 2^n이다. 따라서, 입력 변수가 많은 경우 기존의 함수 분해 방법을 직접 적용하기는 거의 불가능하다.

두 번째로는 함수 분해 기법을 적절하게 사용하는 문제로서, 지금까지의 함수 분해가 단순 분리형 함수분해(simple disjunctive decomposition)인 경우 10-변수 함수로 구성 가능한 경우의 수는 2^{95}이고 단순

함수 분해를 포함한 모든 가능한 함수 분해의 수는 2^{1024}이므로 기존의 함수 분해 법은 실제로 특정한 함수에 대해서만 유용하다고 할 수 있다. 이러한 문제점으로 인하여 새로운 함수 분해 법이 필요하고 좀 더 경제적인 회로 설계가 이루어져야 한다.

한편, Fang[14]은 고전적인 함수 분해 방법을 배제하고 2-변수 다치 논리 함수의 성질을 이용하여 함수의 표현 형태를 분류하고 분류된 각각의 함수를 모듈로 만들어 라이브러리 화하였다. Fang의 방법에서는 모든 함수가 매우 간단하게 부분 함수로 분해되고 모듈로 쉽게 전환된다. 그러나, 부분함수와 모듈을 대응시키는 과정에서 제어함수를 도입 해야하는 문제점과 부분 함수들 사이에 함수 관계가 형성되어 회로 테스트면 에서 단점이 있다. 본 연구에서는 Fang의 테이블 검색 방식을 그대로 적용하였으나 모듈구성과 함수 분해 법을 비롯한 설계 방식에서 차이가 있다. Fang의 경우 함수 표현은 식 (2)와 같다.

$$F(x_3, x_4)=\varphi(x_1, x_2)\cdot x_3+\varphi(x_1, x_2)\cdot x_4 \qquad (2)$$
$$=\varphi(x_1, x_2)(x_3+x_4)$$

여기서, $\varphi(x_1, x_2)$가 부분 함수이고 그림 3과 같이 나타낼 수 있다. 그러나, 입력 변수가 5-변수 이상인 경우 부분 함수 φ는 다시 다른 부분 함수로 표현되어야 하는 단점을 갖는다.

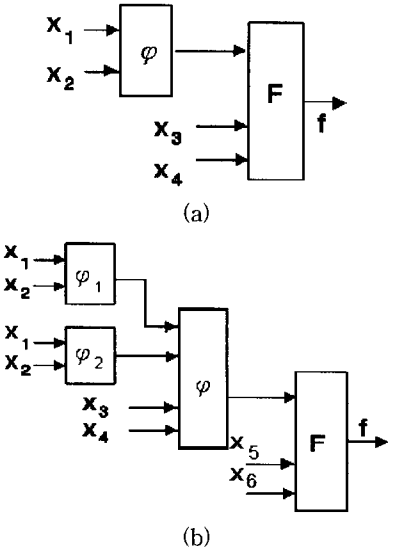


그림 3. Fang의 방법에 의한 함수 분해
(a) 4-변수 (b) 6-변수
Fig. 3. Decomposition of Fang's method.
(a) 4-variables (b) 6-variables

3. 제안된 부분 함수 분해 법

기존의 회로 설계 방식의 단점을 보완하기 위해 Fang의 방법에서 부분 함수와 모듈이 일치하지 않는 경우 발생하는 제어 함수 문제를 개선하고 부분 함수의 크기를 줄여 동일 모듈 발생 확률을 높이기 위해 정리 1과 같이 주어진 함수의 행 분할을 2-변수 함수로 분할한다.

[정리 1] 임의의 m-치 n-변수 함수 f를 $f: M^n \rightarrow M$라 하면 행 변수를 x_1, x_2 의 2-변수로 분할할 때 부분함수 φ는 항상 m개의 부분 함수 $\varphi_1, \varphi_2, \ldots, \varphi_m$ 으로 분해되어 다음과 같은 식 (3)의 부분 함수 형태를 갖는다.

$$\begin{aligned} F(\varphi(x_1, x_2), X_n-2) = \{ & \varphi_1(x_1, x_2(0)), \varphi_2(x_1, x_2(1)) \\ & \ldots, \varphi_m(x_1, x_2(m-1))\}\cdot x_3 \\ +\{ & \varphi_2(x_1, x_2(0)), \varphi_2(x_1, x_2(1)), \ldots, \varphi_m(x_1, x_2(m-1))\}\cdot x_4 \\ & \vdots \qquad (3) \\ +\{ & \varphi_2(x_1, x_2(0)), \varphi_2(x_1, x_2(1)), \ldots, \varphi_m(x_1, x_2(m-1))\}\cdot x_n \end{aligned}$$

따라서, 함수 f는 부분함수 φ_i와 입력변수 X_n-2로 다음 식 (4) 와 같이 표현된다.

$$f(X_n)=F(\varphi_1, \ldots, \varphi_m, X_n-2) \qquad (4)$$

[증명] m-치 논리시스템의 행 변수가 2-변수이므로 총 m^2개의 행 변수 분할이 생기고, 2-변수 중 1-변수가 상수 입력이므로 x_2 -변수의 경우 $x_2(0)$, $x_2(1), \ldots, x_2(m-1)$가 되어, 부분함수의 입력이 $(x_1, x_2(0)), (x_1, x_2(1)), \ldots, (x_1, x_2(m-1))$이 되는 부분함수 $\varphi_1, \varphi_2, \ldots, \varphi_m$을 얻을 수 있다. 따라서, 행 변수가 2-변수인 경우 항상 m개의 부분함수를 얻을 수 있다. ■

본 논문의 경우 Fang의 부분 함수 분해와 달리 변수의 수가 증가하는 경우에도 다시 부분 함수로 분해되지 않는다. 3-치 논리 시스템인 경우 즉, m=3 일 때 3개의 부분함수 $\varphi_1, \varphi_2, \varphi_3$를 얻을 수 있고 각 부분 함수 사이에는 다음 식 (5)의 관계를 만족한다.

$$\varphi_i(x_1, x_2)=\{\varphi_1(x_1, x_2(0)), \varphi_2(x_1, x_2(1)), \varphi_3(x_1, x_2(2))\}$$

① $\varphi_1(x_1, x_2(0))=\varphi_2(x_1, x_2(1))\neq\varphi_3(x_1, x_2(3))$

② $\varphi_1(x_1, x_2(0))\neq\varphi_2(x_1, x_2(1))=\varphi_3(x_1, x_2(2))$

③ $\varphi_1(x_1, x_2(0)) = \varphi_3(x_1, x_2(2)) \neq \varphi_2(x_1, x_2(1))$ (5)

④ $\varphi_1(x_1, x_2(0)) = \varphi_2(x_1, x_2(1)) = \varphi_3(x_1, x_2(2))$

⑤ $\varphi_1(x_1, x_2(0)) \neq \varphi_2(x_1, x_2(1)) \neq \varphi_3(x_1, x_2(2))$

따라서, 정리 1은 그림 4와 같은 복합 분리형 함수분해 형태를 갖는다.

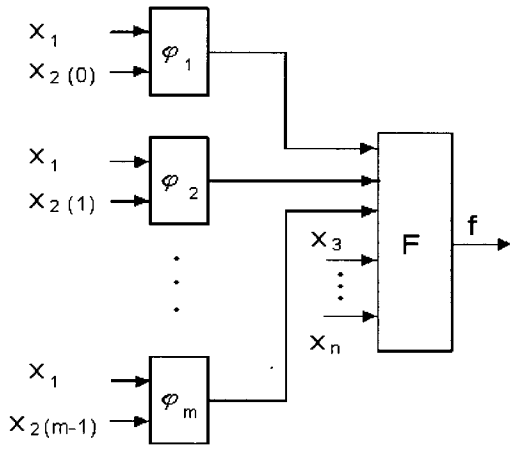


그림 4. 제안된 부분 함수분해
Fig. 4. Proposed functional decomposition.

이러한 부분 함수들은 단일 T-gate 모듈만을 사용하고 이들이 조합된 모듈 라이브러리는 사용하지 않는다. 그러나, 회로 구현 시 T-gate 이외의 PLA혹은 기타 ULM으로도 회로 구현이 가능하다.

4. 기본 모듈 구성

제안된 부분 함수분해 법을 적용하여 논문에서 사용하는 모듈을 정의하기 위해 m-치 n-입력변수의 열 분할과 행 분할 변수를 표 1과 같이 일반화하여 나타낸다.

표 1. m-치, n-변수 함수 테이블
Table 1. Table of m-valued, n-variable functions.

	π_1	π_2	...	$\pi_{m^{n-3}}$	$\pi_{m^{n-2}}$
σ_1	$f(\pi_1, \sigma_1)$	$f(\pi_2, \sigma_1)$	...	$f(\pi_{m^{n-3}}, \sigma_1)$	$f(\pi_{m^{n-2}}, \sigma_1)$
σ_2	$f(\pi_1, \sigma_2)$	$f(\pi_2, \sigma_2)$	...	$f(\pi_{m^{n-3}}, \sigma_2)$	$f(\pi_{m^{n-2}}, \sigma_2)$
σ_3	$f(\pi_1, \sigma_3)$	$f(\pi_2, \sigma_3)$	...	$f(\pi_{m^{n-3}}, \sigma_3)$	$f(\pi_{m^{n-2}}, \sigma_3)$
⋮	⋮	⋮	...	⋮	⋮
σ_{m^2}	$f(\pi_1, \sigma_{m^2})$	$f(\pi_2, \sigma_{m^2})$	...	$f(\pi_{m^{n-3}}, \sigma_9)$	$f(\pi_{m^{n-2}}, \sigma_9)$

입력이 m-치 n-변수 논리 함수의 열 분할의 개수는 열 분할 $\pi = \{\pi_1, \pi_1, ..., \pi_{m^{n-2}}\}$ 이므로 $|\pi| = m^{n-2}$이고, 행 분할의 개수는 행 분할 $\sigma = \{\sigma_1, \sigma_2, \sigma_3, ..., \sigma_{m^2}\}$ 이므로 m-치 논리 시스템의 열 변수가 2-변수인 경우 $|\sigma| = m^2$ 이다. 표 1은 다음의 관계를 만족한다.

① $\pi_1 \cup \pi_2 \cup \cdots \cup \pi_{m^{n-2}} = \pi$, $\pi = \{X_n - 2\}$

② $\pi_1 \cap \pi_2 \cap \cdots \cap \pi_{m^{n-2}} = \varnothing$

③ $\sigma_1 \cup \sigma_2 \cup \cdots \cup \sigma_{m^2} = \sigma$

④ $\sigma_1 \cap \sigma_2 \cap \cdots \cup \sigma_{m^2} = \varnothing$

본 논문에서는 표 1에서와 같이 행 변수는 항상 2-변수이고, 열 변수는 전체 입력에서 행 변수를 제외한 나머지 변수가 배열된다. 3-치 5-변수 함수인 경우 행 분할의 수는 27, 열 분할의 수는 9가 된다. 이때, 정리 2의 부분함수는 정의 2의 모듈과 일치하게 된다.

[정의 2] 임의의 m-치 n-변수 함수 테이블에서 모듈 M을 다음과 같이 정의한다.

$$M_i = f(\sigma_1\sigma_2...\sigma_m : \pi_i) \qquad 1 \le i \le m^{n-2}$$

$$M_j = f(\sigma_{m+1}\sigma_{m+2}...\sigma_{2m} : \pi_j) \qquad 1 \le j \le m^{n-2} \quad (6)$$

$$\vdots$$

$$M_k = f(\sigma_{m^2-(m-1)}\,\sigma_{m^2-(m-1)}...\,\sigma_{m^2} : \pi_k) \quad 1 \le k \le m^{n-2}$$

여기서, σ : 행 변수, π : 열 변수

이때, 식 (6)의 모듈들 사이에는 식 (7)의 관계를 만족한다.

$$M_i \cup M_j \cup ... \cup M_k = U \quad (U \text{ ; 전체 집합}) \qquad (7)$$

$$M_i \cap M_j \cap ... \cap M_k \neq \varnothing \quad (\varnothing \text{ ; 공 집합})$$

3-치 5-변수의 경우 가능한 총 모듈 수는 표 3.1에서 열 분할 변수가 π_1 에서 π_{27}까지, 행 분할 변수는 σ_1 에서 σ_9까지 이므로 $27 \times 3 = 81$ 개이다. 다음 예제 2는 함수 테이블로부터 정의된 모듈을 생성한다.

[예제 2] 주어진 함수 테이블(표 2)로부터 부분 함수를 분해하여 모듈로 나타내면 다음과 같다.

열 분할이 3-변수이므로 표 1과 같이 $\pi_1, \cdots, \pi_{3^3}$ 이 되고 이 중 표 2의 함수가 π_9까지 정의되어 있으므로 식 8과 같다.

$$\pi_1 = x_3 x_4 x_5 = 000$$

$$\pi_2 = x_3 x_4 x_5 = 001 \qquad (8)$$

$$\pi_3 = x_3 x_4 x_5 = 002$$
$$\vdots$$
$$\pi_9 = x_3 x_4 x_5 = 022$$

표 2. 3-치 5-변수 함수
Table 2. Ternary five-variable functions.

$x_1 x_2$	$x_3 x_4 x_5$ = 000	001	002	010	011	012	020	021	022
00	100	122	200	001	111	202	110	212	020
01	122	200	001	111	202	010	212	020	121
02	200	001	111	202	010	121	020	121	212
10	200	001	111	202	010	121	020	121	212
11	100	122	200	001	111	202	110	212	020
12	122	200	001	111	202	010	212	020	121
20	122	200	001	111	202	010	212	020	121
21	200	001	111	202	010	121	020	121	212
22	100	122	200	001	111	202	110	212	020

행 분할은 2-변수이므로 표 1과 같이 $\sigma_1, \ldots, \sigma_{3^2}$ 이 되고 주어진 함수가 σ_9-행까지 정의되어 있으므로 식 (9) 와 같이 나타낸다.

$$\sigma_1 = x_1 x_2 = 00$$
$$\sigma_2 = x_1 x_2 = 01 \quad (9)$$
$$\sigma_3 = x_1 x_2 = 02$$
$$\vdots$$
$$\sigma_9 = x_1 x_2 = 22$$

표 2의 함수로부터 행 변수가 $\sigma_1, \sigma_2, \sigma_3$ 일 때, CM=9 이므로 9개의 모듈을 구할 수 있으며, 이 중 동일 모듈을 제외한 3개의 모듈 M_1, M_2, M_3 를 구할 수 있다. 행 변수가 $\sigma_4, \sigma_5, \sigma_6$ 일 때와 $\sigma_7, \sigma_8, \sigma_9$인 경우에도 각 각 9개의 모듈을 얻을 수 있으나, 동일 모듈을 제외한 3개의 모듈 M_4, M_5, M_6과 M_7, M_8, M_9 를 각 각 얻을 수 있다. 이들 모듈을 나타내면 식 (10)과 같다.

$$M_1 = f(\sigma_1\sigma_2\sigma_3 : \pi_1) = 112$$
$$M_2 = f(\sigma_1\sigma_2\sigma_3 : \pi_2) = 020 \quad (10)$$
$$M_3 = f(\sigma_1\sigma_2\sigma_3 : \pi_7) = 101$$
$$M_4 = f(\sigma_4\sigma_5\sigma_6 : \pi_1) = 211$$
$$M_5 = f(\sigma_4\sigma_5\sigma_6 : \pi_2) = 002$$
$$M_6 = f(\sigma_4\sigma_5\sigma_6 : \pi_7) = 110$$
$$M_7 = f(\sigma_7\sigma_8\sigma_9 : \pi_1) = 112$$
$$M_8 = f(\sigma_7\sigma_8\sigma_9 : \pi_2) = 200$$
$$M_9 = f(\sigma_7\sigma_8\sigma_9 : \pi_7) = 011$$

여기서, M_1, M_7은 함수 값(112) 를 갖는 동일 모듈임을 알 수 있다. 이러한 동일 모듈들은 적절한 간략화 기법을 이용하면 최소화가 가능하다.

Ⅲ. 회로 설계

1. 입력 변수 코드할당

본 논문에서는 T-gate를 기본 모듈로 사용한다. 다치 논리 분야에서 T-gate 는 지금까지 널리 사용되어 온 매우 유용한 소자로서 향후 다치 논리-FPGA 등의 기본 블록으로 널리 사용될 전망이다.[19] T-gate 는 그림 5와 같이 m개의 입력과 1개의 제어 변수로 구성되어 있고 정의 3과 같이 동작한다.

[정의 3[20]] m-치 논리시스템의 출력이 p_i 인 T-gate의 동작은, 제어 입력이 $S=i$ 일 때 출력은 $f(p_0, p_1, \ldots, p_{m-1} : S) = P_i$이다. 여기서, $0 \le p_i, S \le m-1$.

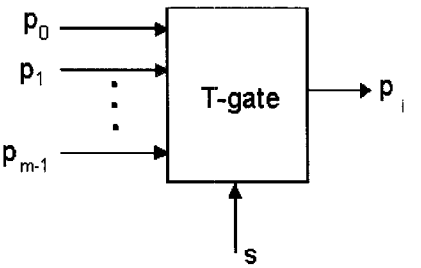


그림 5. m-치 T-gate 기호
Fig. 5. Symbol of m-valued T-gate.

본 절에서는 제안한 함수 분해 법을 도입하여 회로를 설계하는 과정에서 입력변수 처리를 단순화하기 위해 코드를 할당하는 방법을 제안한다.

각 모듈의 입력 변수를 간략화하기 위해 구해진 모듈의 입력 변수를 새로운 집합 λ_i 를 도입하여 표시하고 정리 3에서 코드 길이 β 를 결정한다.

[정리 3] 각 모듈의 입력 변수에 할당되는 코드의 길이를 β 라고 하고 얻어진 모듈 수를 k 라 하면 r-치 논리 함수인 경우 $k \le r^{\beta}$ 가 되는 최소 β 값을 코드 길이로 정한다.

[증명] 회로 구현에 필요한 모듈 수는 CM의 수

k와 일치한다. 따라서, 각각의 모듈 M_k에 대해 열 변수 분할 π_i 로 이루어진 분할 λ_k를 얻을 수 있고, 각각의 λ_k에 대해 코드를 할당하므로 m-치 논리시스템의 경우, 코드 길이 β에 대해서는 m^β 만큼 표현이 가능하므로 $k \le m^\beta$ 인 관계를 만족한다. ∎

또한, 정의 4는 모듈에 적절한 코드 값을 할당하기 위한 치환관계를 나타낸다.

[정의 4] m-치 논리 시스템에서 모듈 수를 k라 하고, 코드 길이를 β라 하면, 각 모듈에 코드 값을 할당시키기 위한 치환관계 $\prod_i$는 식 (11)과 같이 나타낸다.

$$\prod_i : \lambda_i \longrightarrow R_{\beta-1}\, R_{\beta-2} \ldots R_1\, R_0 : C_{i-1} \quad (11)$$

여기서, $0 \le R \le m-1$, $i=1,\ldots,n$.

[예제 3] 3-치 논리시스템에서 m=3, k=9 인 경우 $9 \le 3^\beta$를 만족하는 β의 최소 값은 β = 2 이므로 코드 할당은 2 자리의 코드 값으로 할당된다.

$$\begin{aligned}
&\prod_1 : \lambda_1 \longrightarrow R_1\, R_0 (=0\ 0) : C_0 \\
&\prod_2 : \lambda_2 \longrightarrow R_1\, R_0 (=0\ 1) : C_1 \\
&\prod_3 : \lambda_3 \longrightarrow R_1\, R_0 (=0\ 2) : C_2 \qquad (12) \\
&\qquad\vdots \\
&\prod_9 : \lambda_9 \longrightarrow R_1\, R_0 (=2\ 2) : C_9
\end{aligned}$$

[적용예 1] 본 논문에서 제안한 함수 분해 형태에 따라 3-치 5-변수 함수를 표 3[11]과 같이 행 변수 분할σ에 따라 σ_A, σ_B, σ_C 로 분류하고 부분함수를 분해한다.

표 3. 3-치 5-변수 함수- I .
Table 3. Ternary five-variables function table- I .

	X_3 X_4 X_5																								
	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	2	2	2	2	2	2	2	2	2	2
X_1X_2	0	0	0	1	1	1	2	2	2	0	0	0	1	1	1	2	2	2	0	0	0	1	1	1	2	2	2
	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2
0 0	1	0	0	0	0	1	1	1	0	0	1	1	0	2	0	1	1	0	2	0	1	1	0	0	0	1	1	1	0
0 1	1	2	2	2	2	1	0	1	2	0	0	0	1	1	1	2	2	0	0	0	2	0	0	1	2	2	2	2	1	0	1	2
0 2	2	0	0	0	0	2	1	2	0	0	2	2	0	0	0	2	2	2	0	0	0	2	0	0	0	2	1	2	0	0	2	0
1 0	1	2	2	2	2	1	0	1	2	0	0	0	1	1	1	2	2	0	0	0	2	0	0	1	2	2	2	2	1	0	1	2
1 1	1	0	0	0	0	1	1	1	0	0	1	1	0	2	0	1	1	0	2	0	1	1	0	0	0	1	1	1	0
1 2	2	0	0	0	0	2	1	2	0	0	2	2	0	0	0	2	2	2	0	0	0	2	0	0	0	2	1	2	0	0	2	0
2 0	1	2	2	2	2	1	0	1	2	0	0	0	1	1	1	2	2	0	0	0	2	0	0	1	2	2	2	2	1	0	1	2
2 1	1	0	0	0	0	1	1	1	0	0	1	1	0	2	0	1	1	0	2	0	1	1	0	0	0	1	1	1	0
2 2	2	0	0	0	0	2	1	2	0	0	2	2	0	0	0	2	2	2	0	0	0	2	0	0	0	2	1	2	0	0	2	0

σ_1, σ_2, σ_3 인 경우에 대하여 부분 함수를 분해하면 $x_1\, x_2$:(00,01,02) 인 경우로 표 4와 같다.

표 4. $f(\sigma_1\sigma_2\sigma_3 : \pi_i)$ 인 경우 함수 테이블
Table 4. Function table of $f(\sigma_1\sigma_2\sigma_3 : \pi_i)$.

	π_0	π_1	π_2	π_3	π_4	π_5	π_6	π_7	π_8	π_9	π_{10}	π_{11}	π_{12}	π_{13}	π_{14}	π_{15}	π_{16}	π_{17}	π_{18}	π_{19}	π_{20}	π_{21}	π_{22}	π_{23}	π_{24}	π_{25}	π_{26}	π_{27}
σ_1	1	0	0	0	0	1	1	1	0	0	1	1	1	0	2	0	1	1	0	0	0	0	1	1	1	0	0	1	1	0
σ_2	1	2	2	2	2	1	0	1	2	0	0	0	1	1	1	2	2	0	0	0	2	0	0	1	2	2	2	2	1	0
σ_3	2	0	0	0	0	2	1	2	0	0	2	2	0	0	0	2	2	2	0	0	0	2	0	0	0	2	1	2	0	0
	①	②		③		④	⑤													⑥									

표 4에서 CM=6이므로 총 6개의 모듈을 얻을 수 있다. 각각의 모듈을 나열하고 입력 변수분할 π_i 를 함께 나타내면 표 5와 같다.

표 5. 모듈과 입력 변수분할
Table 5. Modules and input variable partition.

모듈	함수 값	입력 변수분할
M_1	1 1 2	π_1, π_6, π_8, π_{18}, π_{24}, π_{26}
M_2	0 2 0	π_2, π_3, π_4, π_5, π_9, π_{20}, π_{21}, π_{22}, π_{23}, π_{27}
M_3	1 0 1	π_7, π_{25}
M_4	0 0 0	π_{10}, π_{15}, π_{17}
M_5	1 0 2	π_{11}, π_{12}, π_{13}, π_{14}, π_{18}
M_6	2 2 0	π_{16}

표 5의 입력 변수분할을 분할 λ_i 를 적용하여 다시 나타내면 식 (13)과 같다.

$$\begin{aligned}
\lambda_1 &= \{\pi_1, \pi_6, \pi_8, \pi_{19}, \pi_{24}, \pi_{26}\} \\
\lambda_2 &= \{\pi_2, \pi_3, \pi_4, \pi_5, \pi_9, \pi_{20}, \pi_{21}, \pi_{22}, \pi_{23}, \pi_{27}\} \\
\lambda_3 &= \{\pi_7, \pi_{25}\} \\
\lambda_4 &= \{\pi_{10}, \pi_{15}, \pi_{17}\} \\
\lambda_5 &= \{\pi_{11}, \pi_{12}, \pi_{13}, \pi_{14}, \pi_{18}\} \\
\lambda_6 &= \{\pi_{16}\}
\end{aligned} \quad (13)$$

여기서, $\lambda_1 \cap \lambda_2 \cap \lambda_3 \cap \lambda_4 \cap \lambda_5 \cap \lambda_6 = \emptyset$

분할 $\lambda_1 \sim \lambda_6$ 는 각 각 동일한 모듈을 나타내기 위한 입력 변수분할의 집합이므로 집합 λ_i에 대해 서로 다른 코드를 할당하여 간략화 하면 입력 변수처리를 단순화 할 수 있으므로 보다 간단한 회로를 설계할 수 있다. 임의의 m-치 다치 논리 함수에서 모듈 수를 k라 하면 정리 3에 의해 코드의 자리 수 β가 얻어지고 각 모듈에 대해 정의 4와 같이 코드가 할당되므로 k개의 코드 할당 테이블이 필요하다. 이러한 방법

에 의해 표 6을 얻을 수 있다.

표 6. 모듈의 입력 변수에 대한 코드 할당
Table 6. Code assignment for input variables of modules.

π	λ_1
0 0 0	0 0
0 1 2	
0 2 1	C_0
2 0 0	
2 1 2	
2 2 1	

π	λ_2
0 0 1	0 1
0 0 2	
0 1 0	C_1
0 1 1	
0 2 2	
2 0 1	
2 0 2	
2 1 0	
2 1 1	
2 2 2	

π	λ_3
0 2 0	0 2
2 2 0	C_2

π	λ_4
1 0 0	1 0
1 1 2	
1 2 1	C_3

π	λ_5
1 0 1	1 1
1 0 2	
1 1 0	C_4
1 1 1	
1 2 2	

π	λ_6
1 2 0	1 2
	C_5

따라서, 주어진 함수(표 4)는 표 7과 같이 모듈과 입력 변수분할 λ로 나타낼 수 있다.

표 7. 모듈에 의한 함수 테이블
Table 7. Function table of modules.

	λ_1	λ_2	λ_3	λ_4	λ_5	λ_6
$\sigma_1\ \sigma_2\ \sigma_3 = \sigma_A$	M_1	M_2	M_3	M_4	M_5	M_6

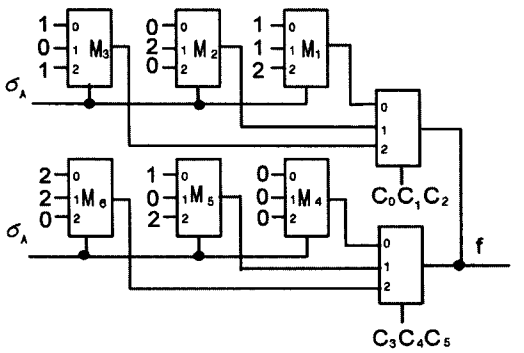
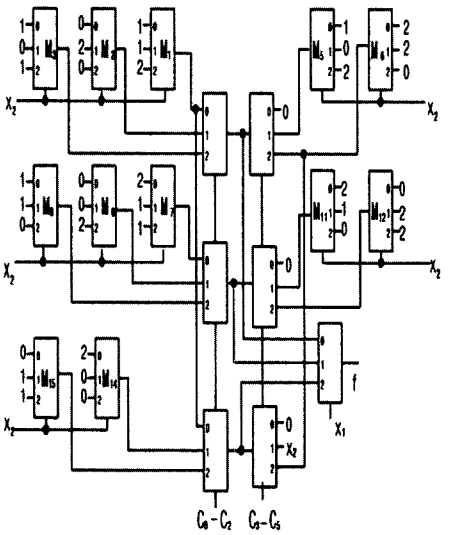


그림 6. 모듈을 이용한 표 4의 회로 구성
Fig. 6. Circuit design of table 4 using modules.

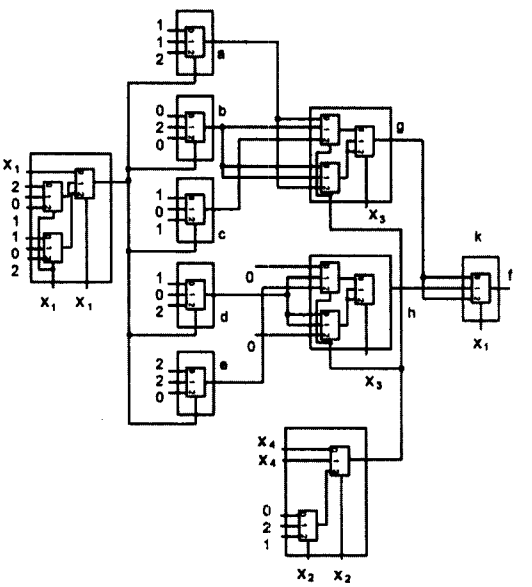
각각의 모듈은 변수 분할 λ_k에 의해 선택되고 그 때의 함수 값은 σ 변수에 의해 출력된다. 출력된 값은 회로 테스트 등의 목적에 따라 단일 출력 혹은 다중 출력이 가능하다. 이때, λ_k의 코드 값은 표 6에서 정의한 바와 같다. 여기서, 각 모듈은 정의 4의 T-gate를 기본 구성소자로 하면 그림 6과 같이 회로를 구성할 수 있다.

표 8. 부분함수로 표현된 다치 논리함수
Table 8. MVL function represented by sub-functions.

	λ_1	λ_2	λ_3	λ_4	λ_5	λ_6
σ_A	M_1	M_2	M_3	M_4	M_5	M_6
σ_B	M_7	M_8	M_9	M_{10}	M_{11}	M_{12}
σ_C	M_{13}	M_{14}	M_{15}	M_{16}	M_{17}	M_{18}



(a)



(b)

그림 7. 표 3의 회로 구현 (a) 본 논문의 회로 (b) Fang의 회로[14]
Fig. 7. Circuit implementation of table 3. (a)Circuit of this paper (b) Circuit of Fang[14]

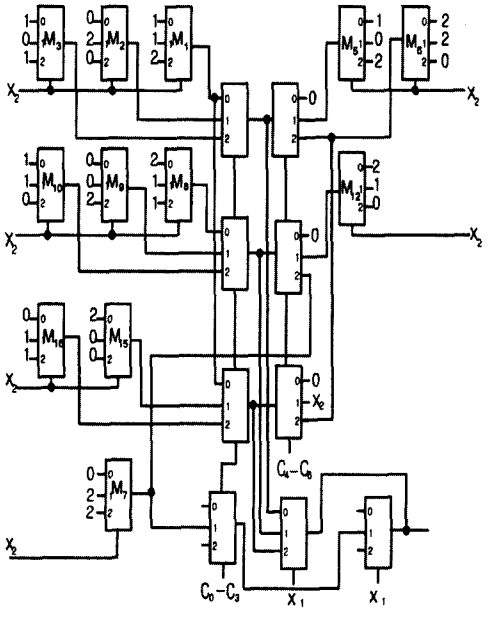
행 변수 분할이 $\sigma_B : \sigma_4\sigma_5\sigma_6$ 와 $\sigma_C : \sigma_7\sigma_8\sigma_9$의 경우는 $\sigma_A : \sigma_1\sigma_2\sigma_3$에서의 과정을 반복하여 얻을 수 있다. 표 3으로부터 총 18개의 모듈을 얻을 수 있으며 표 8과 같다. (자세한 과정 및 함수 값은 참고문헌 [22] 에 나타나 있다.)

이를 회로로 나타내면 그림 7과 같고 회로 비교는 Ⅳ. 비교 및 검토에 나타낸다.

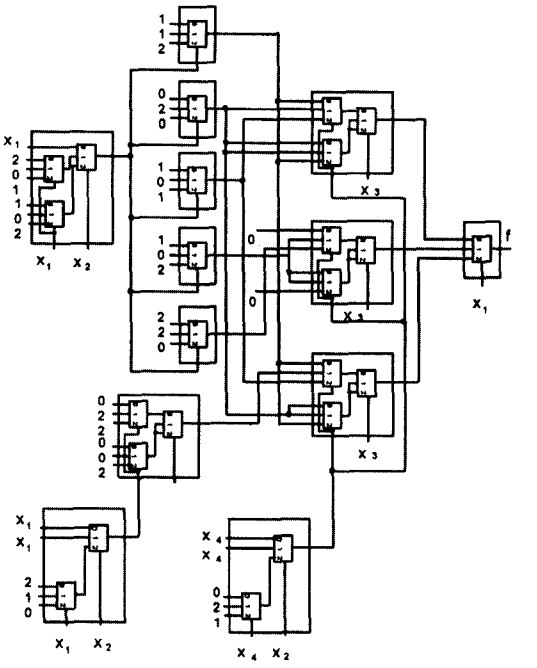
[적용예 2] 표 9[23] 는 표 3의 함수 값이 $f(0\,2\,2\,1\,0) = 0 \rightarrow 2$ 로 변환된 경우를 나타낸다. 이는 함수의 규칙성이 감소되는 형태라 할 수 있으며 Fang의 함수 분해 법을 적용할 경우 모듈의 급격한 증가를 볼 수 있다. 반대로 Fang의 방법이 적용되기 쉬운 형태인 경우, 본 논문에서 제안된 부분함수 분해 법 또한 적용이 용이하다.

표 9. 3-치 5-변수 함수-Ⅱ
Table 9. Ternary five-variable functions-Ⅱ.

x_1x_2 \ $x_3x_4x_5$	000	001	002	010	011	012	020	021	022	100	101	102	110	111	112	120	121	122	200	201	202	210	211	212	220	221	222
00	1	0	0	0	1	1	0	1	1	2	0	1	1	0	1	1	0	1	1	0	1	1	0	1	1	0	1
01	1	2	2	2	1	0	1	2	1	0	0	0	0	0	0	2	0	0	1	2	2	1	0	1	2	1	0
02	2	0	0	0	0	2	1	2	0	0	2	2	0	2	2	0	2	2	0	0	2	1	2	2	0	2	1
10	2	0	0	0	0	2	1	2	0	0	2	2	0	2	2	0	2	2	0	0	2	1	2	2	0	2	1
11	1	0	0	0	0	1	1	1	0	0	2	0	1	1	0	1	1	0	1	2	0	1	1	0	1	1	0
12	1	2	2	2	1	0	1	2	1	0	0	0	0	0	0	2	0	0	1	2	2	1	0	1	2	1	0
20	1	2	2	2	1	0	1	2	1	0	0	0	0	0	0	2	0	0	1	2	2	1	0	1	2	1	0
21	1	0	0	0	0	1	1	1	0	0	2	0	1	1	0	1	1	0	1	2	0	1	1	0	1	1	0
22	2	0	0	0	0	2	1	2	0	0	2	2	0	2	2	0	2	2	0	0	2	1	2	2	0	2	1



(a)



(b)

그림 8. 표 9함수의 회로 구현 (a) 본 논문의 회로 (b) Fang의 회로[14]
Fig. 8. Circuit implementation of table 9 (a)Circuit of this paper (b) Circuit of Fang[14]

Ⅳ. 비교 및 검토

본 논문에서는 모듈 설계 기법에 중점을 두고, 기존의 함수 분해 방법과 다른 새로운 형태의 함수 분해 법을 제안하였다. 제안된 방법은 고전적 방법으로 함수 분해가 불가능한 경우에도 적용 가능하므로 매우 효과적이고, 함수 분해 법이 매우 단순하므로 쉽게 적용이 가능하다. 입력 변수에 대해 코드 변환을 하여 처리 과정을 단순화하였다. 비교 대상에 있어서 구성 모듈의 복잡 도가 서로 다른 모듈을 직접 비교하여 수치화 한다는 것은 모순이며 고전적인 회로 설계 방법과 비교하는 것도 어렵다 따라서, Fang과 본 논문의 방식을 고전적 함수 분해방법과 비교 할 수 없으므로 단지, Fang이 제안한 방법과 비교할 수 있다. 적용예에서 사용한 함수는 Fang의 방법에서 사용한 함수이며 본 논문에서도 부분함수 분해과정을 기술하고 결과를 Fang의 방법과 비교하기 위해 사용하였다.

1. 부분 함수분해

표 10에서와 같이 Fang이 제안한 부분 함수는 함

수 분할 표에서 CM과 일치하게 된다. 따라서, 3-치 5-변수 함수인 경우 행 번수는 3^3 = 27개의 패턴을 갖게 된다. 제안된 알고리듬의 경우 3^4=81 개의 패턴을 갖게 된다. 그러나, 소요되는 T-gate수는 모듈 라이브러리의 가장 복잡한 형태인 경우, 1 모듈 당 4개의 T-gate가 소요되므로 27×4=108개가 필요한 반면, 본 논문의 경우 총 81개의 T-gate만이 소요되므로 복잡 도 면에서 오히려 간단해진다. 부분 함수분해 과정에서는 기존의 방식이 제안된 방식보다 테이블 검색 면에서 더욱 단순하다. 즉, 부분 함수 분해 과정이 기존의 경우가 1회인 반면 제안된 알고리듬의 경우 3회가 소요된다. 표 10에서 두 가지 방식의 테이블 검색 횟수 및 모듈 수 등을 비교하였다.

표 10. 변수 증가에 따른 검색 횟수 및 모듈 수 비교

Table 10. Number of search and modules to the increment of variables.

변수	테이블 검색		부분 함수		모듈 수(T-gate)			부분 함수의 원소수	
	본 논문	Fang	본 논문	Fang	본 논문	Fang min.	Fang max.	본 논문	Fang
3	3	3	9	3	9	3	12	3	9
5	13	13	81	27	81	27	108	3	9
7	26	26	729	243	729	243	972	3	9
9	63	63	6561	2187	6561	2187	8748	3	9
11	118	118	59×10^3	20×10^3	59×10^3	20×10^3	79×10^3	3	9
13	196	196	53×10^4	18×10^4	53×10^4	18×10^4	71×10^4	3	9
15	301	301	48×10^5	16×10^5	48×10^5	16×10^5	64×10^5	3	9
17	437	437	43×10^6	14×10^6	43×10^6	14×10^6	57×10^6	3	9
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
n	n!/[2!·(n-2)!] +N(n-2)*		3^{n-1}	3^{n-2}	3^{n-1}	3^{n-2}	$3^{n-2}\times4$	3	9

N(n) : n변수 검색 횟수

모듈 수에 있어서 Fang의 경우 부분함수의 규모가 제안한 방식보다 크므로 이를 회로로 구현하는 모듈자체의 크기 또한 크다. 따라서, 함수에 따라서, 보다 적은 모듈이 사용될 수 있으며 혹은 더욱 많은 모듈이 사용될 수도 있다. 표 10에서와 같이 Fang의 경우 가장 단순한 모듈이 사용되는 경우와 가장 복잡한 형태의 모듈이 사용되는 경우가 함수에 따라 변동이 생기고 그에 따라 내부결선의 복잡 도 또한 변동이 있다. 그러나 함수 내에는 동일한 함수 값을 갖는 경우가 있으므로 본 논문에서 제안한 함수 분해 법을 사용할 경우 동일 부분함수를 얻는 경우의 수가 상대적으로 부분함수의 규모가 큰 Fang의 방법보다 증가하게 된다. T-gate 수만을 고려한다면 가장 복잡한 경우(Max) 가장 간단한 경우보다 4배 더 많은 T-gate를 사용하게된다.

2. 모듈 구성 및 적용

모듈 구성은 부분 함수 분해와 매우 밀접한 관련이 있는데 함수 분해 방식에 따라 모듈의 구성이 달라질 수 있다. Fang의 경우 3-치 논리 함수의 대표 모듈을 제안하여 라이브러리화 하였다. 그러나, Fang의 라이브러리는 함수의 기저에 따라 라이브러리의 구성 원소 수가 증가하게된다. 본 논문의 경우 단일 T-gate를 기본 블록으로 사용하므로 기저 증가에 따른 모듈 증가 요소가 없다.

본 논문에서는 2개의 적용예를 통해 소요 모듈을 비교하였다. [적용예 1] 의 함수의 경우 Fang의 방식으로 8개의 부분 함수를 얻을 수 있고, 제어 함수를 포함해 총 10개의 모듈이 소요된다. 본 논문의 경우 총 18개의 모듈이 얻어지고 간략화 과정[22]을 통해 12개로 줄어들었다. 최종 함수 구현에는 총 19개의 모듈이 사용된다.

[적용예 2] 의 함수의 경우 [적용예 1] 의 함수 값이 $f(0\,2\,2\,1\,0) = 0 \rightarrow 2$로 변화된 경우이다. 여기서, 표 3의 함수는 대칭성이 매우 크고, CM이 비교적 작은 형태이다. 반면, 표 9의 경우 함수 값이 변화하여 이러한 대칭성이 줄어든 형태가 된다. Fang의 알고리듬에서는 변화된 함수 값으로 인해 새로운 모듈이 추가되어, 부분 함수 구현에 9개의 모듈이 사용되었고, 제어 함수를 포함하여 총 13개의 모듈이 필요하다. 본 논문의 경우 부분 함수 구현을 위한 모듈은 함수 변환전과 동일한 12개이고 함수 구현을 위한 전체 모듈 수는 21개가 소요된다. 따라서, 2개의 모듈 증가가 발생하게 된다. 함수 변환에도 불구하고 부분 함수가 증가하지 않은 이유는 이미 존재하는 부분 함수와 동일 함수이기 때문이며, 기존의 방법에 비해 새로 제안된 방법이 동일 부분 함수로 처리 될 수 있는 확률은 모듈이 추가됨에 따라 증가한다. 내부결선에 있어서 적용예 1에서는 새로 제안한 방법이 다소 복잡한 것으로 나타났고 적용예 2에서는 Fang의 경우 사용 모듈이 급격히 증가함에 따라 자연히 내부 결선의 복잡 도 역시 급격한 증가를 보였다. 표 11에서 내부결

선의 수는 계산을 단순화하기 위해 각 모듈의 입출력 단자 수와 모듈내부의 결선만을 고려하였다.

표 11.적용예를 통한 모듈 수 비교
Table 11.Comparison of the modules through examples.

내 용	적용예 1		적용예 2	
	본 논문	Fang	본 논문	Fang
부분 함수수	12	12	8	9
모듈수	19	21	10	13
T-gate 수	19	21	17	25
내부 결선	95	81	105	118

모듈설계 방식은 기본적으로 내부 결선의 감소를 장점으로 하고 있으나, 본 논문의 경우 기존의 모듈 설계에 의한 결선 감소 보다, 더욱 감소율을 증가시킬 수 있다. 이러한 내부 결선 감소 요인을 보면, 모듈 라이브러리 방식과 달리 기본 블록만을 사용하므로 전체적인 게이트 수가 감소되었다. 또한, 제안된 방식에서는 제어 함수를 사용하지 않으므로 불필요한 모듈 추가가 없으며 모듈의 입력 변수를 코드 할당 기법을 사용하여 복잡 도를 감소시켰다. 그러나, 이 경우 코드 할당을 하기 위한 테이블 생성과 알고리듬처리 단계가 증가하는 단점이 있다.
표 12는 회로 설계 방식에 대하여 본 논문과 Fang의 방식을 비교하였다.

표 12.설계 방식에 의한 비교
Table 12.Comparison of design types.

항 목	Fang의 방법	본 논문
모듈 구성 방식	모듈 라이브러리 구성	단일 T-게이트 사용
모듈 적용 방식	제어 함수 사용	제어 함수 사용하지 않음.
입력 변수 제어	모듈에 직접 연결	코드 할당 기법을 이용한 LUT방식
출력 방식	단일 출력	단일/다중 출력

3. 입력 변수 코드 변환

부분 함수분해에 있어서 본 논문은 기존의 방법보다 기본적으로 부분 함수의 개수가 많아진다. 그러나, 부분 함수의 규모 즉, 부분 함수의 원소수가 작으므로 회로 구현시 총 모듈 수가 기존의 방법과 유사하거나 (표 3) 보다 적은 수로 구현이 가능하다.(표 9) 즉, 반복되는 부분 함수를 최소화하여 회로가 간략화 되도록 하는 것이 본 연구의 목적이다. 이러한 연구에서 발생하는 문제점으로는 다음 2가지를 들 수 있다.
첫째, 입력 변수를 코드 할당하기 위한 테이블이 필요
둘째, 모듈 수가 증가함에 따라 테이블 수가 함께 증가하므로 추가 메모리를 필요로 한다.
입력 변수를 간략화 하기 위한 코드 할당 기법은 모듈이 증가함에 따라 코드 길이 및 테이블 수가 증가하게 된다. 표 13 은 모듈 수와 코드 길이 및 코드 할당 테이블 수를 나타낸다. 코드 할당을 하기 위한 전체 테이블 수는 전체 모듈 수와 일치하므로 함수 간략화 과정[22]에서 모듈 수를 줄이는 방법을 통해 코드 할당 테이블 수를 최소화 할 수 있다.

표 13.모듈 증가에 따른 코드 길이 및 테이블 수
Table 13.The code length and tables according to the module increment.

모듈수 k	코드길이 β	테이블수 $T(k)$	최적화된 테이블수 $T'(k)$
5	2	5	2 이하
10	3	5	4 이하
15	3	15	5 이하
20	3	20	7 이하
30	4	30	10 이하
k	$k \le 3^{\beta}$* (3치논리함수)	$T(k) = k$	$T'(k) \ge 1/3 \cdot T(k)$ $T'(k)$ = 최소 정수 값

간략 화된 이후의 테이블 수는 간략화 이전보다 비교적 크게 줄어들게 되는데 총 소요 테이블 수의 1/3 이하로 줄어드는 것을 알 수 있다.
Sasao[15]의 경우 분해된 부분 함수의 출력 부분에 코드를 할당하여 복잡 도를 감소시키는 방식을 사용하고 있으나 함수 분해 방법과 모듈 적용방식의 차이점으로 본 논문과 직접 비교는 어렵다. 이러한 LUT 방식은 FPGA, PLA등에 응용되고 있으므로 본 논문에서 제안한 모듈 분해 방식과 코드 할당 기법을 이용하는 다치 FPGA 혹은 다치 PLA도 제작이 가능할 것으로 사려된다.

V. 결 론

본 논문에서는 다치 논리함수의 모듈러 설계에 적합한 부분 함수 분해 방법을 제시하였다. 부분 함수의 검색횟수는 고전적 함수 분해 방법보다 줄어들었고 검색 방식 면에서도 매우 단순하다. 그러나, Fang의 방식과는 테이블 검색 횟수는 동일하나, 검색 방식에 있

어서는 본 논문의 경우가 다소 복잡하다.

모듈 적용 면에서 본 논문에서는 T-gate를 기본 모듈로 사용하므로 모듈 라이브러리 방식의 단점인 제어 함수를 생략할 수 있었다. 소요 모듈 수는 함수 분할 표에서 행 변수의 CM이 작고, 대칭성이 큰 경우(적용예 1) 기존의 방식과 비교하여 전체 T-gate 규모에 있어서 비슷한 복잡 도를 나타내었다. 그러나, 함수가 행 변수에 대한 대칭성이 줄어든 경우(적용예 2) 기존의 방식에서는 모듈이 증가된 반면 본 논문의 경우 부분 함수를 표현하기 위한 모듈 증가가 없었고 단지 함수 출력 부분에 모듈이 추가되었다.

회로 설계 과정에서 열 분할 변수를 처리하기 위해 코드 할당 기법을 도입한 것은 본 논문이 처음이며 그 결과 회로 복잡 도를 감소시킬 수 있었다. 그러나, 코드 할당을 위한 테이블 증가가 단점으로 지적될 수 있다. 모듈 설계 방식의 가장 큰 특징인 내부 결선 감소와 설계의 단순함 측면에서, 본 논문은 기존의 방식과 비교하여 객관적인 수치를 제시할 수 없었으며 단지, 부분적인 척도로서 T-gate 개수를 비롯하여 제어 함수, 입력 변수처리 등을 언급하였다. 본 논문에서 제안한 모듈 설계 방식의 특징을 정리하면 다음과 같다.
첫째, 모듈 라이브러리 방식을 사용하지 않고, 단일 T-gate 기본 블록만을 사용한다. 따라서, 제어 함수를 생략할 수 있었다.
둘째, 열 분할 변수 부분에 코드 할당 기법을 적용하여 입력 변수를 간략화하고, 각 모듈이 동일한 입력 변수 공유할 수 있도록 하였다.
세째, 회로 간략화 과정 후 회로 구성이 기존의 방식과 비교하여 T-gate 개수에서 대등하고, 함수 복잡도가 증가할수록 기존의 방식보다 더욱 간단한 회로를 얻을 수 있었다. 그러나, 코드 할당 기법을 사용함으로 인해 알고리듬 단계와 테이블이 증가될 수 있으며 이로 인한 메모리 요구 및 데이터 처리 부담이 증가할 가능성이 있으므로 추후 연구 과제로는 본 논문에서 새로 제안한 코드 할당 기법의 단점으로 지적된 코드 할당 테이블 수를 간략화 하기 위한 간략화 기법의 연구가 필요하다.

참 고 문 헌

[1] Tadeusz Luba, "Decomposition of Multiple-Valued Functions." IEEE 25th ISMVL., 1995., pp. 256-261.

[2] R. L. Ashenhurst, "The decomposition of switching function" Proc. Int. Sympo. Theory of Switching, vol. 29, pp. 74-116, 1959.

[3] J. Roth & R. Karp, "Minimization over Boolean Graph" IBM J. Res. Develop., vol. 9, pp. 227-238. April. 1962.

[4] M. Davio, J. Deschamps, and A. Thayse, *Discrete and Switching Functions.* Mc-Graw-Hill Inc.1978.

[5] Sami B, Abugharbieh & Samuel C. Lee, "*Fast Algorithms For the Disjunctive Decomposition of m-valued functions.* part I : The Decomposition Algorithm," IEEE 23th ISMVL., 1993. pp. 118-125.

[6] S. Takami M, Kameyama & T. Higuchi, "Code assignment alogorithm for highly parallel Muliple-Valued combinational circuits," IEEE Proc. 22nd ISMVL pp. 330-336. May. 1992.

[7] V. Shen, A. Mckeller, P. weiner, "A fast algorithm for the disjunctive decomposition of switching functions" IEEE Trans. Compt., vol. C-20, pp. 304-309, Mar. 1971.

[8] E. L. Post, "Introduction to a general theory of elementary propositions," American. J. Math, vol. 43, pp. 163-185., 1921.

[9] M. Orsic "Multiple-Valued Logic Modules" 2nd ISMVL 1972., pp. 95-99.

[10] J. Loader, "Second order and higher order universal descision elements in m-Valued Logic" 5th ISMVL., 1975., pp. 53-59.

[11] S. Thelliez, "Introduction to the study of Ternary Switching structures, Gordon and Breach Science, New York, 1973.

[12] T. Higuchi & M. Kameyama, "Ternary logic system based on T-gate" 5th ISMVL, 1975., pp. 290-304.

[13] T. Higuchi & M. Kameyama "Synthesis of Multiple-Valued Logic networks based on tree-type universal logic modules" 5th ISMVL 1975, pp. 121-130, 1975.

[14] Fang K. Y. and Wojcik, A. S. "Modular Decomposition of Combinational Multiple-Valued Logic Circuits." IEEE Trans. computers vol. c-37, no. 10 Oct. pp. 1293-1301 1988.

[15] Sasao T. Logic *Synthesis and optimization* Kluwer Academic Pub., 1993.

[16] Ronald E. Prather, *Introduction to switching theory a mathematical approach.* "Chapter 7. Decomposition Theory," pp. 405-1967. Allen & Bacon Inc. 1968.

[17] J. S. Choi "A Study on the synthesis of Multiple-Valued Logic functions based on modular decomposition approach," 1989 Proc. KIET., summer., 1989.

[18] M. Kameyama, and T. Higuchi, "Synthesis of Multiple-Valued logic networks based on tree-type universal logic module" IEEE 5th ISMVL., 1975., pp. 121-130.

[19] K. C, Smith, P. G, Gulak "Prospects for Multiple-Valued Integrated Circuits "special issue on Multiple-Valued Integrated circuits" IEICE Trans. Electron. vol. E76-C., no. 3, Mar. 1993.

[20] D. L, Dietmeyer *Logic design of digital systems.* Allyn and Bacon, Boston, 1971.

[21] Ronald E. Prather, *Introduction to switching theory a mathematical approach.* "Chapter 7. Decomposition Theory," pp. 405-1967. Allen & Bacon Inc. 1968.

[22] J.S.Choi, "A Study on the Modular Function Decomposition of Multiple-Valued Logic Functions using Code Assignment Techniques" Ph.D Thesis of Inha Univ. 1997.

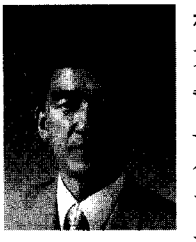
[23] Miller, D.M. and Muzio, J.C.,"Two-Place Decomposition and the Synthesis of Many-Valued Wsitching Circuits", ISMVL-16th, pp. 164-168, Logan, Utah, May 1976.

저 자 소 개

崔 在 碩(正會員)
1988년 2월 인하대학교 전자공학과 졸업(공학사). 1990년 2월 인하대학교 대학원 전자공학과 졸업(공학석사). 1997년 8월 인하대학교 대학원 전자공학과 졸업(공학박사). 1990년 1월 ~ 1995년 4월 (주)기아정보시스템 개발 연구소 근무. 주관심분야는 다치논리시스템, VLSI 설계, DSP & DIP 등 임

朴 春 明(正會員) 第 35卷 C編 第 6號 參照
현재 충주대학교 컴퓨터공학과 조교수

成 賢 慶(正會員) 第 34卷 C編 第 12號 參照
현재 상지대학교 전자계산학과 부교수



朴 承 用(正會員)
1979년 2월 인하대학교 전자공학과 졸업(공학사). 1982년 2월 인하대학교 대학원 전자공학과 졸업(공학석사). 1996년 3월 ~ 현재 인하대학교 대학원 전자공학과 박사과정. 주관심분야는 다치논리시스템, 컴퓨터구조 등 임

金 興 壽(正會員) 第 35卷 C編 第 6號 參照
현재 인하대학교 전자공학과 교수