濟大論文集(自然). 35. 193-202. 1992
Cheju Univ. Jour. (Natural Sci.),
35. 193-202, 1992

다출력함수의 다단 논리 최적화 알고리즘

임재윤*, 이기태**

Multi-level Logic Optimization Algorithms for Multiple Output Functions

*Jea-Yun Lim**, *Gi-Tae Lee***

Summary

The multiple-output logics can be implemented with multi-level logic. Common subexpressions between functions are substituted with intermediate variables. Boolean substitution process between functions is executed to reduce literals, and then by executing factorizing procedure recursively, multilevel logic is simplified with optimal literals.

서 론

최근 VLSI 집적도가 증가하고 설계할 회로가 복잡해짐에 따라, 설계 자동화 시스템에 대한 필요성이 크게 대두되었으며, 이를 일반 다단회로 또는 PLA와 같은 규칙회로로 구성할 경우나, 디지탈 컴퓨터의 제어부 설계와 같은 복잡한 시스템 설계에 응용될 경우 최적 설계를 위해 논리함수의 최적화가 절실히 요구되고 있다. 다출력 함수 실현시 규칙적인 논리구조에서 발생되는 시간 지연 문제를 해결하고 게이트 어레이등에 일반적으로 사용할 수 있도록 최적의 다단 논리회로로 실현하려는 연구가 활발히 진행되고 있다(E. Llawler, 1983). 다단 논리함수는 모든 형태의 논리회로 실현에 적합하고, 이의 최종 목표는 주어진 가격함수 이내에서 최소의 게이트 수를 갖는 다단 회로를 구성하는 것이다. 이러한 다단 논리를 실현하는 방법은 여러가지가 있으며, 이들은 부울망이라는 다단 논리 구조를 생성하는 효율적인 분해(decomposition) 및 인수분해(factorization)된 회로를 구성하는 것이다(R. Brayton, et al., 1987).

종래의 다단 논리 최적화는 주로 공통 논리 재대치 의한 WEAK DIVISION법(R. Brayton : 1982)

* 공과대학 통신공학과(Dept. of Communication Engineering, Cheju Univ., Cheju-do, 690-756, Korea)
** 공과대학 통신공학과 대학원

이 보편적으로 사용되었으나, 이는 각 함수의 인자화 과정에서 발생되는 재대치 문제는 다루지 못했고, 함수 간에 발생될 수 있는 부울 등가회로를 다루지 않았다. 최근 이러한 부울 등가 개념을 사용한 방법들이 제안되었으나 부울 등가회로를 구하는 과정에서 새로운 변수를 추가함으로서 논리최소화를 복잡하게 하여 부울 등가회로를 찾는데 과다한 시간이 소요된다(K. A. Bartlett, et. al., 1988)

본 다단 논리최소화법은 주어진 부울망에 대해서 최적인 다단 회로를 구성하기 위한 효율적인 분해 및 인수분해에 대해서 논하며, 2단 논리최소화 과정을 거친 형태의 함수를 입력으로 받아 초기에 빠른 2단논리 최소화기법을 이용한 부울 재대치 및 함수간 공통 부표현 재대치를 수행한후, 각 함수별 요소화 과정을 거친 논리함수 최적화 기법에 관한 알고리즘을 제시한다.

다단논리함수의 기본성질 및 데이타구조

다단논리 함수 실현시 곱의 합(Sum-of-Product)형의 함수 표현에서 각 논리를 표현하는 최소 단위를 변수(Variable)라하고, 임의 변수와 그의 보수를 리터럴(Literal)이라 하며, 리터럴의 논리곱을 큐브라 하고, 큐브들의 논리합을 표현식이라 한다. 2개 이상의 큐브들에서 생성되는 공통 부표현들은 커넬이라 하며 다단 논리함수에서 이러한 커넬들을 생성케 해 주는 공통인자들을 보조커넬이라 한다. 표현식 f에서 공통된 리터럴이 존재하지 않는다면 이식은 큐브프리(cube free)라고 한다. 예로 F=aef+bef+cef는 리터럴 e에 의해 F=e(af+bf+cf)로 인수분해 될 수 있으며 이때 e를 보조커넬, F′=af+bf+cf를 커넬이라 한다. 그러나 이 커넬은 공통 리터럴 f를 갖으므로 큐브프리가 아니다. 이 공통 리터럴 f로 다시 인수분해 하면 F=ef(a+b+c)로 된다. 여기서 F″=a+b+c는 더 이상의 공통 리터럴를 갖지 않으므로 큐브프리이다.

다단 논리회로는 곱의합(SOP) 형태의 데이타를 받아들여 이 형태에 알맞도록 함수를 저장하며, 초기에 읽어들인 SOP형의 데이타들은 각각의 함수별로 적합한 데이타 구조를 형성한후 각 부함수에 트리구조로서 입력된다. 이때 입력된 각 큐브들은 마스크 및 각 항으로서 표시되며 하나의 큐브에 대한 표시법은 Fig. 1.과 같다.

```
0  1  -  1   :  x4′ x3 x1
   a) single cube
1  1  0  1   :  Mask
0  1  0  1   :  Data
   b) cube representation
```

Fig. 1. Cube representation method.

여기서 마스크는 각 입력변수가 큐브에 존재하는지의 여부를 검사하는 방법으로서, 만일 1이면 해당 입력 변수가 존재, 0이면 존재하지 않음을 표시하며, 데이타표현에서 값이 1이면 해당 정변수를 표시하고, 0이면 해당 부변수 또는 해당 변수가 존재치 않음을 나타낸다. 이는 각 큐브에 해당하는 모든 상태를 정수에서 각 비트로 표시함으로서 변수가 2 이상일 경우는 메모리를 감소시킬 수 있고, 각 변수순으로 저장됨으로서 비교가 용이하여, 입력변수의 증가에 따라 비교 횟수가 일정하게 됨으로서 효율적인 논리망을 구성할 수 있다.

여기서 각 함수에 대한 데이타구조는 Fig. 2. 와 같다. 초기에 각 함수별로 입력된 각 큐브들은 각 함수구조의 subfunction에 2진 트리구조로서 저장된다. 그후 다단 논리 최적화 과정에서 부울 재대치 또는 공통함수 재대치시 생성된 각종 중간변수들은 데이타 구조 intervar에 해당 중간 출력변수에 대한 변수명이 포인터를 저장한다. 또한, 각 함수별 subfunction의 데이타에 의해 분할된 함수중 보조 커넬은 subfunction에 남고, 몫에 해당되는 부함수는 kernel에 저장되고 나머지 함수는 remains에 저장된다. kernel 및 remains도 새로운 함수들이므로 이 각 함수에 대해서도 위의 데이타 형성과정과 동일한 과정을 거친다.

```
struct function {
        struct intervar *ivfunc;
```

```
        struct subfunction *sub;
        struct function *kernel;
        struct function *remains;
        struct function *next;
        };

struct intervar {
        char *ivname;
        struct intervar *next;
        };

struct subfunction {
        int submask;
        int subcube;
        struct subfunction *next;
        };
```

Fig. 2. Data structure of multi-level logic functions.

공통 부표현 재대치

본과정은 다출력 논리함수들간에 공통으로 존재하는 부함수식을 하나의 중간변수로 간주하여 해당 부표현식을 하나의 변수로 대치함으로서 다단 논리실현시 총리터럴의 수를 감소시켜 최소의 리터럴로 함수를 실현시키고자 하는 것이다.

이러한 과정은 함수들간 공통 부표현을 찾는 방법과, 현재는 부표현이 존재하지 않으나, 인수분해 수행과정에서 발생되는 부표현에 의해서 대치되는 방법으로 대별할 수 있으며, 이 과정은 부울 제산과정과 병행해서 실행함으로서 총 리터럴의 수를 간소화 시킬 수 있다.

Fig. 3.은 함수들간의 공통 부표현 중 일부분이 서로 중복될 경우 이를 새변수로 대치하는 과정을 나타낸 것으로

y1=x3 x1′+x3′ x2 x1+x4′ x1

y2=x4 x2′+x3′ x2 x1+x4′ x1

에서 x3′ x2 x1+x4′ x1이 서로 공통되므로 이를 새 중간변수 y3로 대치하면

y1=x3 x1′+y3

y2=x4 x2′+y3

y3= x3′ x2 x1+x4′ x1으로 대치할 수 있다.

이때 각 함수에 대해서 새로운 입력변수가 추가되며, 이는 새 함수명에 대한 포인터의 값을 저장함으로서, 다단논리 최적화 표현 및 신호선 리스트 형성시 출력할 수 있게 한다.

		x4	x3	x2	x1
y1	:	-	1	-	0
		-	0	1	1
		0	-	-	1
y2	:	1	-	0	-
		-	0	1	1
		0	-	-	1

a) before execution

		y3	x4	x3	x2	x1
y1	:	-	-	1	-	0
		1	-	-	-	-
y2	:	-	1	-	0	-
		1	-	-	-	-
y3	:	-	-	0	1	1
		-	0	-	-	1

b) after execution

Fig. 3. Partial common subexpression substitution process.

한편, 함수들간에 공통 부표현이 존재하나, 임의 함수가 포함되면 새로운 변수를 도입하지 않고, 단지 입력변수만을 확장하여 그 함수명에 대한 포인터만을 저장하게 한다. Fig. 4.는 그 과정의 예를 보인 것으로

y1=x3 x_1′+x3′ x2 x1+x4′ x1

y2=x3′ x2 x1+x4′ x1

에서 y2는 y1의 일부분이므로

y1=x3 x1′+y2

y2=x3′ x2 x1+x4′ x1으로 쓸 수 있다.

이때 확장된 입력 변수명을 해당 포함 함수명의 포인터를 저장함으로서 출력시 해당 변수명을 출력 시킬 수 있도록 한다.

		x4	x3	x2	x1
y1	:	-	1	-	0
		-	0	1	1
		0	-	-	1
y2	:	-	0	1	1
		0	-	-	1

a) before execution

		y2	x4	x3	x2	x1
y1	:	-	-	1	-	0
		1	-	-	-	-
y2		-	-	0	1	1
		-	0	-	-	1

b) after execution

Fig. 4. Function common subexpression substitution process.

Fig. 5. a)에서는 두함수 y1, y2간에 공통 부표현이 존재치 않으나 y1을 인수분해 하면 y1=x1 (x3′ x2+x4′)+x3 x1′, y2=x3′ x2+x4′가 되어 y2가 공통이므로, y1=x1 y2+x3 x1′, y2=x3′ x2+x4′로 대치할 수 있다.

		x4	x3	x2	x1
y1	:	-	1	-	0
		-	0	1	1
	:	0	-	-	1
y2		-	0	1	-
		0	-	-	-

a) before execution

		y2	x4	x3	x2	x1
y1	:	-	-	1	-	0
		1	-	-	-	1
	:	-	-	0	1	-
y2		-	0	-	-	-

b) after execution

Fig. 5. Common subexpression substitution during fatorization.

이상의 공통 부표현 재대치 알고리즘에 대한 각 단계를 기술하면 다음과 같다.

〔단계 1〕: 각 함수별로 SOP형태의 데이타를 입력한 후, 각 적항에 해당되는 출력수 및 각 함수별 적항수 계산.

〔단계 2〕: 출력을 최대로 공유하는 적항을 선택, 이 출력이 1이되는 항들을 마스크로 하여 1의 갯수가 1 이상이며 많이 공유하는 적항 선택.

〔단계 3〕: 만일 공통 부표현이 다른 함수 전체를 나타낸다면 이 함수에 해당하는 출력변수로 대치, 그렇지 않고 모든 함수에 부분적으로 존재하는 부표현이라면 새로운 중간 변수로 대치.

〔단계 4〕: 대치된 적항 중 기존에 존재하는 출력함수의 1의 갯수를 뺀 후 각 함수별 1의 갯수를 재조정하고, 더 이상의 공통 부표현이 존재하지 않을때까지 위 과정을 반복수행.

부울 재대치

함수의 다단 논리함수 표현식을 간소화하기 위한 방법에는 크게 산술제산(Algebraic division)과 부울 제산(Boolean division)이 있다. 산술제산은 함수들간의 공통 부표현을 찾아 이를 새 변수로 대치하는 공통 부표현 재대치 및 함수내의 공통인자를 중심으로 인수분해 하여 다단논리 형태로 구성하는 인수분해 방법을 말한다. 그러나 산술제산 이 외에 보다 개선된 다출력 논리함수를 구성하기 위해 함수들간의 부울 등가 관계가 있는 함수를 선정하여 이들의 부울 제산을 수행함으로서 원 함수를 보다 간소화된 형태의 다단논리로 구성할 수 있다. 예로 임의의 두함수 y1 및 y2가 식 (1)과 같이 주어 졌을때, y1과 y2는 산술제산으로서 더 이상 간소화할 수 없다.

$$y1=(ab+cd)e'f'+(ab+ef)c'd'+(cd+ef)a'b'$$
$$y2=ab+cd+ef \qquad (1)$$

그러나 y1을 피 제산함수로 하고 y2를 제산함수로

하여 부울 제산과정을 거치면 몫함수로서

h=a'b'+c'd'+e'f'가 생성된다. 따라서 식 (1)은 식 (2)와 같이 간소화될 수 있다.

$$y1 = y2(a'b' + c'd' + e'f')$$
$$y2 = ab + cd + ef \qquad (2)$$

식 (1)에서는 총 리터럴 수가 24이나 부울 제산을 수행한 결과 총 리터럴수가 13으로서 11개의 리터럴의 수를 줄임으로서 함수를 간소화시킬 수 있다. 이러한 부울제산은 일반적으로 양질의 논리회로를 구성할 수 있으나 새로운 몫함수를 구성한 후, 이에 대한 논리함수 최소화를 거쳐야 하므로, 수행시간이 많이 걸린다는 단점이 있다. 또한 논리 최소화기는 입력변수의 크기에 따라 수행 속도가 달라지게 되므로 가능하면 적은 수의 변수를 갖는 형태로 논리함수를 축소하여 부울 제산을 수행할 필요가 있다.

함수 f의 최소화된 형태에서 나타나는 각 변수들의 집합을 함수 f의 support라 하고, sup(f)로 표시하며 sup(f)를 구성하는 변수의 갯수를 ns(f)로 표시한다. 예로 f=a b'+b c+c'에서 suf(f) = {a, b, c} ns(f)=3이 된다.

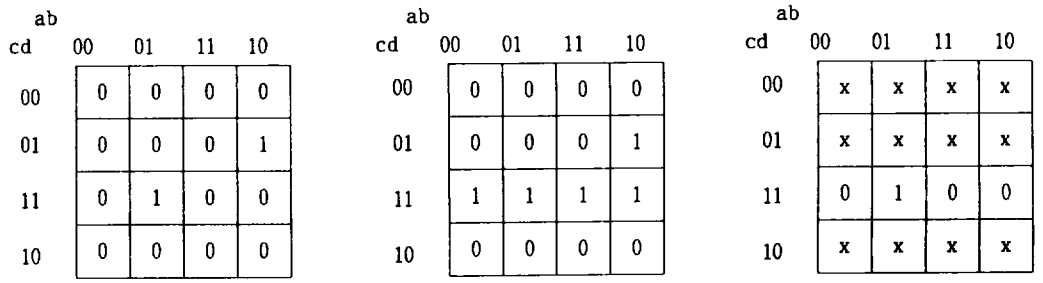
또, 두 함수 f 및 g의 support에서 sup(f) ^ sup(g)=0이라면 두 함수 f와 g는 disjoint support를 갖는다고 말한다.

이러한 부울 제산 함수를 나타내면 Fig. 6.과 같다.

```
BOOLDIV (f, g)
{
  if (f⊄g)
    return (f) ;
  e=Function_mapping (f, g) ;
  h=minimize2 (e) ;
  if (Cost (h)>Cost (f))
      return (f) ;
  else
      return (h) ;
}
```

Fig. 6. Bool division function.

예로 F=a'bcd+ab'c'd, G=cd+ab'd에서 F나 G는 산술 제산으로는 더이상 간소화할 수 없으며, 이때 총 리터럴 수는 12이다. 이들 함수중 F는 G에 포함되므로 이에 대한 부울 제산을 수행한다. Fig. 7.은 이에 대한 함수매핑과정을 나타낸 것이다. 여기서 생성된 함수 H에 대해 논리함수 최소화를 수행하면 H=c'+a'b를 얻을 수 있다.



a) dividend function F　　b) divisor feinction G　　c) quotient function H

Fig. 7. Function mapping process.

따라서 원함수는 F=G(c'+a'b), G=d(c+ab')로 다시 쓸 수 있다. 이때 총 리터럴 수는 8로서 원 리터럴 수에 비해 보다 간소화된 형태로서 함수를 실현할 수 있다.

여기서 피제산 함수를 F, 제산 함수를 G 및 몫함수를 H라 하고 함수 매핑시, 각 민텀들을 mi라 할 때 부울 함수 매핑에 대한 규칙은 다음과 같다.

F(mi)=1 일때 만일 G(mi)=0이라면 제산 불능이므로 복귀하며, 그렇지 않고 G(mi)=1 또는 -1이라면 H(mi)=1을 할당한다. 또, F(mi)=0이고 G(mi)=1이면 H(mi)=0을 할당한다. 한편 이외

의 경우는 H(mi)=-1을 할당한다. 이상의 부울 재대치 알고리즘에 대한 수행단계를 기술하면 다음과 같다.

〔단계 1〕: 함수간의 재산가능 쌍을 선정한 후, 두 쌍중 피제산 F 및 제산 함수 G를 구별.

〔단계 2〕: 두 쌍의 변수들 중 최소의 입력 변수 집합을 선정, 이에 해당하는 변수들에 대한 이름을 비트별로 할당한 후, 최소 입력 변수에 의한 총 최소항집합을 구하여 F 및 G에 대한 함수값을 계산, 각 최소항에 할당.

〔단계 3〕: F 및 G에 대해서 새로운 함수 H로의 함수 매핑을 수행, 만일 제산 불능이 발생하면 복귀.

〔단계 4〕: 생성된 H에 대해서 2단 논리함수 최소화 및 해당함수를 인수분해시킨 후 총 리터럴수를 계산하여, 원 논리 함수 F의 리터럴 수 보다 크면 F를 복귀하고, 그렇지 않으면 H를 복귀.

인수분해 (Factorization)

인수분해의 최종 목표는 최소수의 리터럴을 갖는 인수분해 형태를 얻는 것이며 일반적으로 인수분해된 형태는 유일하게 결정되지는 않는다.

```
FACTOR_GEN(F)
{ COUNT_LIT(F);
  if(Cube_free(F))
    return;
  K=KERNEL_SELECT(F);
  H=F/K;
  R=F-K*H;
  if(common_expression(H, fi))
    substitute(H, fi);
  else
    FACTOR_GEN(H);
  if(Common_expression(R, fj))
    substitute(R, fj);
  else
    FACTOR_GEN(R);
}
```

Fig. 8. Factorization process.

이러한 인수분해 형태를 결정하는 문제를 최소의 리터럴로 해결하기 위해 f=kh+r의 형태로 함수를 인수분해하고 이렇게 생성된 k, h 및 r에 대해 재귀적으로 인수분해를 수행한다. Fig. 8. 은 이러한 인수분해 알고리즘을 나타낸 것이다.

이러한 인수분해 과정은 크게 함수내의 큐브 선택과정과 공통 부표현 재대치 및 재귀적 인수분해 과정으로 구성된다. 공통 부표현 재대치과정은 인수분해 과정중 생성된 커널, 보조커널 및 나머지 함수에 대해서 기존의 함수들과의 동일성 여부를 검사하여 이를 대치하는 과정이다. 큐브 선택과정은 현 함수의 리터럴들 중에서 최대로 묶여질 수 있는 리터럴을 중심으로 커널을 생성하는 수행과정은 Fig. 9. 와 같다.

```
KERNEL_SELECT(F)
{ K=MAX_LITERAL_PAIR(F);
  H=F/K; R=F-H*K;
  Hn=count_cube(H);
  Rn=count_cube(R);
  while(Rn<Hn)
  { if(Common_kernel(R, H))
    { Kr=R/H;
      R-=H*Kr;
      K+=Kr; Rn-=Hn;
    }
  else break;
  }
  if(Common_expresson(K, fi))
    substitute(K, fi);
  else FACTOR_GEN(K);
  return(K);
}
```

Fig. 9. Kernel selection process.

이상의 인수분해 알고리즘에 대한 수행단계를 기술하면 다음과 같다.

〔단계 1〕: 각 함수내의 각 큐브들에 존재하는 리

터럴의 빈도수를 계산한 후, 모든 리터럴의 빈도수가 1이면 더 이상의 공통 리터럴이 존재하지 않는 큐브프리이므로 복귀.

〔단계 2〕: 과정 1에서 구한 빈도수가 최대인 리터럴을 중심으로 이와 각 큐브내에 함께 존재하는 다른 리터럴이 여러 큐브에 가장 많이 공유하는 리터럴을 선택, 이러한 리터럴이 2개 이상인 큐브를 선택하여 이 리터럴로 구성된 항을 기본 커널로하고, 이러한 커널이 속하지 않은 큐브들을 나머지 함수로 구성.

〔단계 3〕: 이 기준커널로 나눈 몫에 해당되는 함수 형태를 보조커널로 형성하며, 나머지 함수에서 보조커널의 각 항과 동일한 형태를 갖고 이로 묶어질 수 있는 리터럴의 집합이 동일한 큐브들을 탐색. 만일, 이러한 큐브들이 존재하면 이를 보조커널로 나눈 몫에 해당되는 리터럴의 곱을 기준 커널과 논리합을 수행하고, 그렇지 않으면 단계 4를 수행. 더 이상의 커널이 존재치 않을 때까지 단계 3을 재귀적으로 수행.

〔단계 4〕: 생성된 커널의 항이 2이상일 경우, 기존의 함수들과의 일치 여부를 검사, 만일 대치 가능하면, 이에 해당하는 함수명으로 대치, 그렇지 않으면 커널에 대한 인수분해 과정을 재귀적으로 반복 수행.

〔단계 5〕: 보조커널을 기존의 함수와 대치가능하면 해당 함수명으로 대치하고, 그렇지 않으면 보조커널에 대해 인수분해 과정을 재귀적으로 반복 수행. 나머지 함수에 대해 기존의 함수와 대치가능하면 해당 함수명으로 대치하고 그렇지 않으면 나머지 함수에 대한 인수분해 과정을 재귀적으로 반복 수행.

〔단계 6〕: 모든 함수에 대해 위과정을 재귀정으로 반복 수행하고, 각 함수들은 내부적으로 트리구조 형태로 저장되며, 출력 형태는 각 함수는 다단논리 및 신호선리스트 형태로 재귀적으로 출력.

다출력 다단 논리 최적화 알고리즘

이상의 각종 유용한 함수들을 종합하여 다단 논리 최적화를 위한 알고리즘에 대한 순서도를 나타내면 Fig. 10.과 같다.

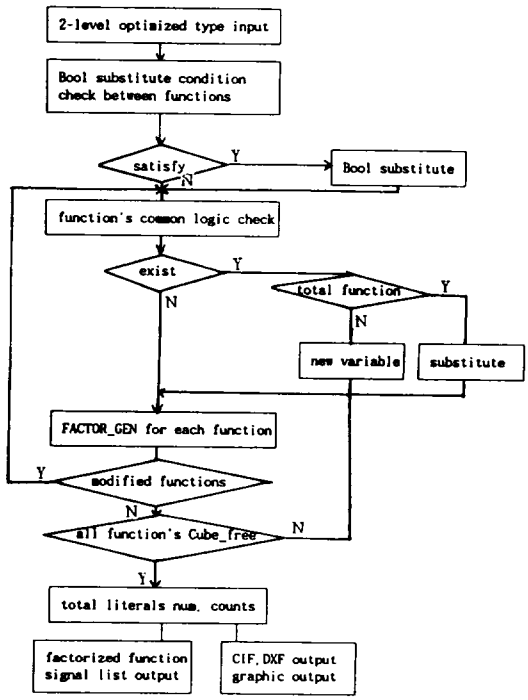


Fig. 10. Multi-level logic optimization algorithm flowchart.

결과 및 고찰

본 다단 논리 최적화기는 SOP 형태의 데이타를 입력하여 이를 최소의 리터럴을 갖는 다단 논리 회로를 구성하는 방법으로서 2단 논리 최소화기와 직접 연결하여 사용할 수도 있고 외부에서 입력 데이타를 받아들일 수 있도록 구성하였으며 그 형태는 Fig. 11.과 같으며, 차후에 Gate array 또는 Standard cell 등에 대한 입력자료로 사용하기 위해, Fig. 11.과 같이 신호선 리스트 형식으로 나타낼 수도 있게 하였다.

```
n     m
p11   p12   · · ·   p1n          f11   f12   · · ·   f1m
p21   p22   · · ·   p2n          f21   f22   · · ·   f2m
 :                                 :
pt1   pt2           ptn          ft1   ft2   · · ·   ftm
```

a) Input form

```
1)   Gi   AND   v1   v2   · · · ·
2)   yi   AND   v1   v2   · · · ·
3)   Gi   OR    v1   v2   · · · ·
4)   yi   OR    v1   v2   · · · ·
5)   yi   EQU   v1
```

b) Net list type

Fig. 10. Input and output form of multi-level logic optimizer.

여기서 Gi는 중간변수를 나타내며, yi는 함수를 표현하고, v1, v2…는 각 입력변수일 수도 있고, 중간변수 또는 함수명일 수도 있다. 또한 보수를 나타낼 경우, 변수앞에 'not'을 붙인다.

본 알고리즘의 효율성을 입증하기 위해 이를 각종 논리식에 적용하여 그 결과를 비교 검토 하였

Table 1. Multi-level logic optimization execution results

	Imputs	outputs	products	SOP	FAC	MULT
EX1	5	1	6	21	15	13
EX2	4	2	7	34	24	19
EX3	4	3	5	23	20	14
EX4	4	3	5	22	18	16
EX5	4	3	8	30	27	23
EX6	4	7	9	88	72	58
EX7	6	5	11	64	46	38
EX8	6	10	33	476	263	222
EX9	6	12	56	443	248	218
EX10	6	10	32	478	281	242
EX11	7	5	132	1125	826	449
EX12	8	6	134	1135	835	514
EX13	8	8	153	1142	925	526
EX14	9	8	186	1253	1025	622

다. 우선 초기의 SOP 형태의 총 리터럴 수를 계산하였고, 각 출력 함수에 대해 인수분해만을 수행한 결과와 공통 부표현 재대치만을 수행한 결과, 함수재대치를 포함한 본 알고리즘의 결과를 리터럴 수 및 수행시간을 비교한 것이 Table 1이다. 여기서 본 알고리즘의 결과는 초기의 리터럴 수에 비해 평균 42%, 인자화만을 수행한 결과에 비해 27%의 리터럴 수의 감소를 가져왔다.

적 요

본 논문에서는 다출력 논리함수의 최적 설계를 위한 다단논리함수 최적화 기법에 대해 논하였다. 다단 논리회로로 실현 시 2단 논리 구조에 의한 다단논리 실현시의 총 리터럴의 수에 비해 약 40%의 리터럴 수의 감소를 가져왔으며 종래의 간소화 방법에 의해 보다 빠른 시간내에 다단논리 회로를 구성할 수 있었다.

논리함수 최소화기를 통해 최소화된 다출력 논리회로의 설계 자동화를 위해 PLA 자동 생성기 및 각종 플립플럽으로 실현시 필요한 입, 출력 특성을 자동 생성케 하는 방법을 제시하였다.

이상에서 제안된 각종 알고리즘들을 VAX-11/780 및 IBM-PC AT상에서 C언어로 프로그램화하여 각종 논리회로에 적용한 후 그 결과를 비교검토 함으로서 본 알고리즘의 효율성을 입증하였다.

앞으로의 연구 과제로서 부울 재대치의 경우 현재는 함수간의 광역적인 부울 계산만을 수행하고 있으나, 요소화 과정중 국소적으로 발생하는 함수들간의 효율적인 부울 재대치가 필요하며, 이를 특정한 CMOS 또는 Gate Array와 같은 논리회로로 실현키 위한 다단 논리 최소화 기법 및 Library Mapping기법에 대한 연구가 수행되어야 할 것이다.

참 고 문 헌

Bartlett, K.A, Kohen, W. Geus, A.D, and Hachtel, G, 1988, Multilevel Logic Minimization Using Implicit Don't cares, *IEEE Trans. on CAD*, Vol. 7, 723~739.

Biswas, N.N, 1985, Multiple Output Minimization, *Proc. 22nd D.A Conf*, 674~680.

Brayton, R and McMullen, C, 1982, The Decomposition and Factorization of Boolean expression, *Proc. of ISCAS*, 49~54.

Brayton, R, Rudell, R, Vincentelli, A.S. and Wang, A, 11987. MISS : A Multiplelevel Logic Minimization system, *IEEE Trans. on CAD*, CAD-6, 1062~1081.

Hong, S.J., Cain, R.G and Ostapko, D.L, 1974, MINI : A Heuristic Approach for Logic Minimization, *IBM J. Res.*, Vol. 18, 443~457.

Lawler, E., 1983, An Approach to Multilevel Boolean Minimization, *Jounal of ACM*, 283~295.