論文 92-29A-4-10

Fanin 제약하의 다단 논리 최적화 시스템의 설계

(Design of Fanin-Constrained Multi-Level Logic Optimization System)

林 春 錫*, 黃 善 泳*

(Chun Seog Lim and Sun Young Hwang)

要 約

본 논문에서는 다단 논리 최적화 알고리듬의 설계와 이에 바탕을 두어 구현한 SMiLE 시스템에 대하여 기술한다. SMiLE 시스템은 fanin 제약을 고려하여 알고리듬 레벨에서 다단 논리 최적화를 휴리스틱 정보를 이용, 정해진 순서에 의거하여 전체 최적화와 부분 최적화를 수행한다. Sogang Silicon Compiler 설계 환경하에서 개발된 SMiLE 시스템은 high-level synthesis 과정에서 출력한 SLIF의 netlist 형태와 Berkeley대학의 equation 형태를 입력으로 하여 최적화된 회로를 SLIF/equation 형태로 출력한다. 표준 테스트 회로인 MCNC benchmarks에 대한 실험결과 가장 널리 사용되는 다단 논리 최적화 시스템인 MIS에 비교할 만한 결과를 보였다.

Abstract

This paper presents the design of mufti-level logic optimization algorithm and the development of the SMiLE system based on the algorithm. Considering the fanin constraints in algorithmic level, SMiLE performs global and local optimization in a predefined sequence using heuristic information.

Designed under the Sogang Silicon Compiler design environment, SMiLE takes the SLIF netlist or Berkeley equation formats obtained from high-level synthesis process, and generates the optimized circuits in the same format. Experimental results show that SMiLE produces the promising results for some circuits from MCNC benchmarks, comparable to the popularly used mufti-level logic optimization system, MIS.

I. 서 론

최근 칩의 집적도가 증가함에 따라 VLSI는 마이크로프로세서를 비롯하여 넓은 응용 분야에 사용되고 있다. 특히 고성능 마이크로 프로세서에서 다단 논리에 바탕을 둔 조합 회로가 콘트롤러의 구현에 이용되는 등 널리 사용되고 있어 이의 설계가 전체 칩의 크기와 동작 성능에 크게 영향을 미친다. 이에 따라 시스템을 구성하는 논리 회로의 효율적인 설계를 위한 시스템의 개발과 알고리듬의 최적화에 대한 연구가 활발히 진행되고 있다.[2,4,5,6,7]

논리 회로는 이단 및 다단 논리로 구현할 수 있다. PLA를 이용한 이단 논리 회로의 경우 면적은 PLA의 행과 열에 의하여 평가할 수 있으며 random logic을 이용한 다단 논리 회로의 경우 트랜지스터의 수에 해당하는 literal의 수로 평가할 수 있다. 논리

*正會員, 西江大學校 電子工學科
(Dept. of Elec. Eng., Sogang Univ.)

接受日字 : 1990年 10月 24日

최적화는 함수를 보다 간단하면서 동일한 기능을 수행하는 형태로 변환하는 과정으로, 칩 구현시 실리콘 면적을 최소화 하는데 목적이 있다. 이단 논리 최소화는 PLA에서 행에 해당하는 product term의 수를 최소화하는 과정이며, 이 논리 최적화 문제는 NP-complete 문제로 중간규모 이상의 논리 회로 최적화를 위하여 MINI,[13] ESPRESSO[4]등의 휴리스틱 알고리듬이 개발되었다. 그러나 요구되는 시스템의 규모가 커짐에 따라 면적과 지연시간이 커지게 되어 PLA로는 고속 동작 회로설계의 제약 조건을 만족시키기 어렵게 되고 이를 해결하기 위해 다단 논리 회로를 사용하게 되었다. 다단 논리 최적화 시스템인 BOULDER,[2] MIS[5]가 연구, 발표되었으나 다단논리 최적화를 위한 효과적인 휴리스틱 알고리듬의 구현은 상대적으로 복잡하고 고려하여야 할 요인이 많으며, 아직도 해결해야 할 문제들이 남아있어 보다 효과적인 휴리스틱 알고리듬의 개발을 위해 계속 연구되고 있다.

다단 논리 최적화는 접근 방법에 따라 알고리듬적 최적화와 rule-based 최적화로 구분되며, 최적화 과정은 기술 독립적 최적화 단계를 거쳐 기술 종속적 최적화로 구현된다. 알고리듬적 최소화는 다시 최적화 대상이 출력함수 전체인지 하나인지에 따라 전체 최적화와 부분 최적화로 구분하기도 한다. 전체 최적화는 한 매크로를 구성하는 여러 출력 함수들 중 공통된 부분을 하나로 대치하여 면적을 줄이며, 부분 최적화는 하나의 함수를 최소의 literal을 갖도록 만들어 면적을 줄인다.[4,5,8]

알고리듬적 접근방식을 택한 MIS는 이와 같은 최소화 과정을 거친 후 기술 매핑을 하나 이 과정에서 fanin 제약 조건을 만족하지 못하는 게이트는 decomposition을 수행하여 fanin 제약을 만족시키도록 함으로써 전체 면적과 global level의 수가 증가하게 된다. 본 논문에서 기술된 다단 논리 최적화 시스템인 SMiLE(sogang multi-level logic minimizer)은 MIS와 같이 알고리듬적 접근방식을 택하였으나 전체 최적화 과정 중의 커널 추출 프로시저와 부분 최적화 과정중의 factoring 프로시저에서 fanin 제약조건을 만족시키도록 함으로써 MIS에서와 같은 기술 매핑시의 overhead를 줄이고 global level과 전체 면적을 감소시키도록 하였다. SMiLE은 현재 개발중인 SSC(sogang silicon compiler)[15]에서 VHDL과 CHDL을 읽어들여 합성한 SLIF(sogang logic interchange format)의 netlist 형태나 equation 형태와 fanin 제약을 입력으로 받아들여 최적화된 netlist나 equation 형태를 출력하게 된다. 전체 최적화 과정의 흐름을 그림 1에 보였다.

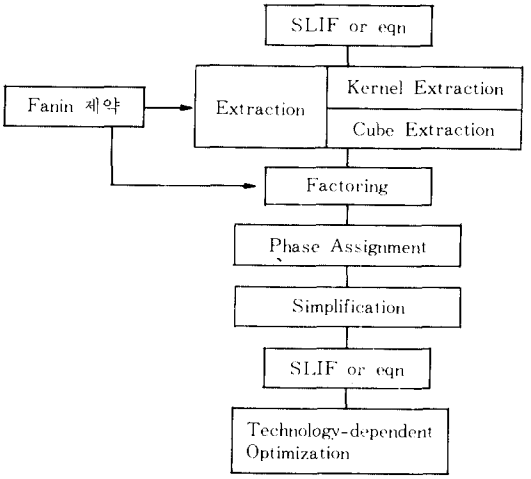


그림 1. 최적화 과정 흐름도
Fig. 1. Optimization process flow.

기술 독립적인 최적화 과정을 거치면 library component selection[1,14], technology mapping 등의 최적화 과정을 수행하게 된다.

Ⅱ. 다단논리 표현 및 비용함수

1. 다단논리의 표현

다단논리를 표현하기 위한 자료구조는 함수를 구성하는 변수들 사이의 관계와 함수와 함수간의 관계를 표현할 수 있어야 한다. SMiLE은 다단 논리를 표현하기 위하여 직병렬 트리구조를 사용하였다.[8] 직병렬 트리에서 각 노드는 중간 변수나 출력 함수를 나타내며, 노드내의 cube의 연결 상태는 함수를 구성하는 cube간의 관계를 나타낸다. 직병렬 트리구조에서 함수와 함수간의 가로 연결은 논리합을, 세로 연결은 논리곱을 의미하며, 함수를 구성하는 변수들 사이에도 같은 표현 방식을 적용한 구조이다. 그림 2는 함수 F, G의 직병렬 트리구조를 나타낸 것이다. N1 노드에 연결된 a, b는 병렬 연결, 즉 sum을 나타내고 N3 노드의 c, d는 직렬 연결로써 product를 의미한다. 노드 구조체에서는 함수의 최적화를 효율적으로 수행하기 위해 노드의 자료 구조속에 cube의 수, 노드의 비용, 노드의 식별자 등의 필드를 두었고, 노드에의 접근을 빠르게 하기 위하여 hash table을 사용하였다. 변수는 노드 안에서 배열로 나타나며 함수 역시 하나의 변수로 사용할 수 있다.

F=(a+b) (c+dG)+cd;
G=cd+c'e;

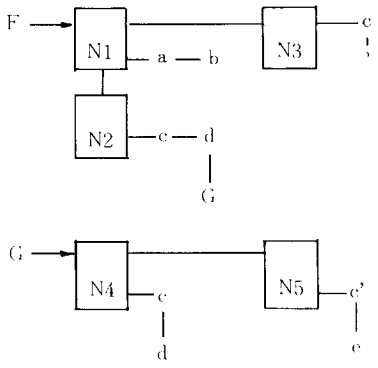


그림 2. 직병렬 트리
Fig. 2. Series-parallel tree.

2. 비용함수

노드의 비용은 그 노드를 구성하는 입력 변수의 수와 AND/OR gate의 수로 다음과 같이 평가할 수 있다. NC(f)는 노드의 비용이며, NP(f)는 노드 f를 구성하는 product term의 갯수이다. NL(n)은 n번째의 cube를 구성하는 literal의 수를 의미한다.

$$NC(f)=\sum_{n=1}^{NP(f)}(NL(n)-LP(n))+NP(f)$$

where, if n-th cube has only one literal then
LP(n)=1
else
LP(n)=0

일반적인 노드의 비용은 위와같이 정의하며, 커널에서의 비용함수는 3장에서 기술한다.

Ⅲ. 전체 최적화 과정 (Global Optimizations)

다단 논리함수의 최적화는 주어진 함수의 기능을 유지하며 최소의 면적, 즉 최소의 게이트 수를 갖는 회로를 찾는데 목적이 있다. MOS 구현 회로에서는 복잡한 함수 자체가 하나의 복합 게이트(compound gate)로 구현할 수 있기 때문에 칩 면적은 트랜지스터의 수로 평가할 수 있으며, 트랜지스터는 함수에서 literal로 표현되므로 비용은 literal의 합으로 계산할 수 있다. 전체 최적화는 다출력 논리회로에서 둘 이상의 함수를 동시에 고려하여 공통된 표현(common subexpression)을 하나의 새로운 변수로 대치함으로서 literal의 수를 줄이는 과정이다.[5] MIS에서는 이 과정에서 fanin을 고려하지 않기 때문에 fanin 제약없이 가장 좋은 커널을 추출하게되나 이 경우 추출된 커널이 기술 매핑시 fanin 제약을 만족시키지 못한다면 global level과 전체 면적의 증가를 가져온다. SMiLE에서는 fanin 제약을 만족하며 전체 면적을 감소시키는 커널을 우선으로 선택하게 함으로써 기술 매핑시의 MIS에서와 같은 overhead를 줄일 수 있도록 하였다.

SMiLE은 전체 최적화 과정을 2단계로 수행 하였다. 첫 단계로 주(primary) 출력 함수에서 커널을 추출하여 중간 함수로 대치하고, 다음 단계로 각 함수의 공통 cube를 찾아 역시 중간 함수로 대치한다.

1. 커널의 추출 (Kernel Extraction)

둘 이상의 주 출력 함수에서 공통인 부분 표현식을 찾아내어 이것을 하나의 새로운 중간 함수로 생성시켜줄 때 전체적인 비용의 감소를 가져올 수 있다. 공통인 부분 표현식은 커널 추출 과정을 통하여 구하며 여기서 추출되어 나온 중간 함수는 비용에 따라 반전 형태로 쓰일 수 있다. 커널 추출 과정은 전체 최적화의 성능을 좌우하는 과정이며 다단 논리 최적화의 핵심이 되는 부분이다. 커널 추출시에 fanin을 고려하여 커널을 추출한다. 커널 추출은 크게 2개의 루틴으로 구성되었다.

(1) Level-0 Kernel의 선택

커널이란 cube-free(더 이상 factored 되지않는 형태)이고 적어도 두개 이상의 cube로 이루어진 함수 표현을 말한다.[4,7,10] 이 커널은 직병렬 트리의 깊이 정도에 따라 n차의 레벨이 있을 수 있다. 높은 레벨의 커널까지 고려한다면 보다 최적화된 다단 논리식을 구할 수 있으나 이들 커널을 추출하기 위해서는 계산 시간이 크게 증가하기 때문에 SMiLE 시스템에서는 level-0 커널만을 고려하였다. 추출된 level-0 커널들은 커널 리스트에 등록되며, 커널 리스트에는 각각의 커널이 함수로 부터 추출될 때 얻을 수 있는 literal의 감소치를 기록하여 커널 선택시에 이용하도록 하였다. 이 literal 감소치(literal savings)를 LS(k)라 하면 LS(k)는 다음과 같이 구한다. 여기서 CNC(f, k)는 노드 f가 커널 k에 의해 변환되었을 때의 노드 비용이다.

$$LS(k)=NC(f)-CNC(f,k)$$

Level-0 커널을 구하는 단계에서는 AND 게이트에 대한 fanin 제약만을 고려하며 OR 게이트의 fanin 제약은 커널 교집합에서 고려한다. 선택할 커널의

```
kernel(i, f)
{
  /* i:i-th variable constituting f */
  /* f:function whose kernel is to be extracted */
  if (f≠φ) then return;
  if (f is cube-free){
    for each cube c_i in f do {
      if (LC(c_i)>AND_FANIN)
        /* LC means Literal Count */
        f=f - c_i;
    }
    Insert f to kernel list;
    return;
  }
  n=number of variables in function f;
  for (j=i;j≤n; j++){
    v=j-th variable in f;
    if (v appears more than once){/* on-phase */
      g=Divide (f, v);
      if (g is not cube-free)
        kernel (j+1, g);
      else{
        k=two or more cubes exist in g which
            satisfy fanin constraint;
        if (k≠φ){
          for each cube c_i in k do {
            if (LC(c_i)>AND_FANIN)
            k=k-c_i;
          }
          Insert k to kernel list;
          return;
        }
      }
    }
    if (v'appears more than once){/* off-phase */
      h=Divide (f, v');
      if (h is not cube-free)
        kernel (j, h);
      else {
        k=two or more cubs exist in g which
            satisfy fanin constraint;
        if (k≠φ){
          for each cube c_i in k do{
            if (LC(c_i)>AND_FANIN)
            k=k-c_i;
          }
          Insert k to kernel list;
          return;
        }
      }
    }
  }
}
```

그림 3. Level-0 커널의 추출
Fig. 3. Extraction of level-0 kernel.

product term이 AND fanin 제약을 초과할 경우, 커널에 의해 생성된 노드에 대해 decomposition을 수행해야 하므로 이를 줄이기 위해 fanin 제약을 초과하는 product term은 버린다. Level-0 커널을 구하는 알고리듬을 그림3에 보였다. 그림3에서 i, j는 함수를 구성하는 변수 중 i, j번째 위치에 나타나는 변수를 의미한다. Devide는 함수에서 해당 변수가 들어 있는 cube만을 선택하여 그 변수만을 제하고 남은 cube들로 이루어진 함수를 돌려준다.

이 프로시져를 수행하면 주 출력 함수 각각에 대하여 커널 트리가 형성되며, 한 예를 그림4에 보였다. f=abcdh+abe+bef+af+ag로 주어진 출력 함수의 경우 커널 프로시져를 수행하면 그림4와 같은 커널 트리를 구할 수 있다. 그림4에서 밑줄은 level-0 커널을 나타내며 만일 fanin 제한이 AND-2라면 (cdh+e)는 생략한다.

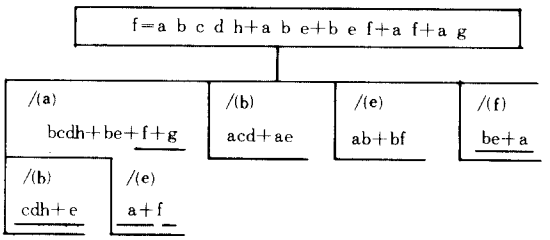


그림 4. 커널 트리의 예
Fig. 4. A kernel tree example.

(2) 커널의 교집합(Kernel Intersection)

만약 두 함수 F와 G의 커널이 교집합을 갖고 있다면 함수 F와 G는 커널 h, k(h∈K(f), k∈K(g))에 공통인 cube들의 집합으로 나누어 진다.[5,7,9] 이 cube를 cube-free common divisor(CFCD)라 한다. CFCD는 커널 교집합이며 커널간의 교집합을 구하는 과정의 complexity는 커널의 갯수를 n이라고 할때 커널 교집합의 수가 2^n-n-1이 되므로 NP-complete이다.[12] SMiLE은 polynomial 시간안에 커널 교집합을 구하고, fanin 제약을 만족하며 함수의 비용을 가장 많이 줄이는 CFCD를 구하기 위하여 다음과 같은 휴리스틱한 방법을 사용하였다. 아래의 예에서 각 함수의 level-0 커널은 이미 선택했다고 가정한다.

단계 1 : 커널 리스트에 나타난 형태가 각기 다른 cube를 KCST(kernel cube symbol table)에 등록하고 각 cube에 일련번호를 부여한다. KCST는 cube와 부여한 일련번호의 순서쌍으로 이루어 진다. 하

나의 cube는 새로운 변수로 변환되며, 변수에 부여된 일련번호는 이 cube가 변환된 커널의 구성 변수로 되었을 때 변수가 차지하게 될 위치를 나타낸다. 예를 들어 함수 f, g, h의 커널들 K(f), K(g), K(h)가 다음과 같이 주어졌을 때:

K(f)={a+b+c, b+c, cd+e}

K(g)={a+b, ab+e, d+e}

K(h)={b+c, c+d, d+e}

KCST={(a, 1), (b, 2), (c, 3), (cd, 4), (e, 5), (ab, 6) (d, 7)}가 된다.

각 KCST 사이에는 우선 순위를 둔다. 어떤 변수가 커널을 구성하는 변수가 되기 위해서는 적어도 두 함수 이상에 입력으로 사용되고, 한 함수에서 여러번 사용되는 것을 커널로 추출하는 것이 유리하므로 다음과 같이 변수에 우선순위를 부여한다. 아래 식에서 VP란 해당 변수의 위치 값(value of position)이고, α는 KCST의 i번째 변수가 몇개의 출력 함수에 있는가를 나타내며, β는 커널 집합에서 i번째 변수가 몇개 있는가를 나타낸다. 비교 우선 순위는 α를 먼저 비교한 후 같으면 β를 비교하여 순위를 결정한다. 변환된 커널의 구성 변수의 수가 OR fanin 제약을 만족하지 않으면 우선 순위가 큰 변수 순으로 커널을 재구성한다.

$$VP(KCST(i), \alpha, \beta) : \alpha = \sum_{n=1}^{\#\ of\ POs} EV(KCST(i)),$$

$$\beta = \sum_{n=1}^{\#\ of\ POs} NV(KCST(i))$$

$$EV(v) = \begin{cases} 1: \text{if v exist in primary output} \\ 0: \text{otherwise} \end{cases}$$

NV(v): # of variable v apperance in primary output

단계 2 : 커널 리스트의 커널을 KCST를 참조하여 변환하고 변환된 커널과 해당 함수, 그리고 literal 감소에 대한 정보를 교집합 리스트(intersection list) I_LIST[0]에 기록한다. I_LIST의 각 원소는 변환된 커널(translated kernel)을 나타내는 TK 필드와, 해당 함수, 커널의 기대치(MK:merit of kernel)의 세 필드를 갖고 있으며, 각 원소들은 linked list로 연결한다. 이때의 커널 기대치는 커널 k를 선택했을 경우의 전체 논리 회로에서 줄일 수 있는 비용의 감소치이며 다음과 같이 계산한다. 커널 k는 AND-OR fanin 제한을 만족하는 커널이다.

k=ITK(VTK(TK_i))

TK_i:i-th translated kernel

VTK:Valid TK(OR fanin 제약을 만족하도록 우선 순위가 높은 변수로 구성된 TK)

ITK:Inverse TK(실제로 구성된 커널)

$$MK(k) = \sum_{n=1}^{\#\ of\ POs} LS(k)$$

단계1의 예에서 K(f)의 첫째 커널은 OR-2 fanin 제약을 만족하지 못한다. 이 커널의 VTK를 구하기 위하여 VP를 구하면 VP(KCST(1), 2, 3), VP(KCST (2), 3, 5), VP(KCST(3), 2, 4)가 되고 변수의 우선 순위는 2>3>1이므로 TK(111…)=(−11…)이며 ITK는 b+c가 된다.

단계 3 : I_LIST[0]의 각 원소의 변화된 커널간에 교집합을 구하여 교집합이 2개 이상 존재하면 I_LIST[1]에 연결하고 기대되는 비용 감소치를 계산하며 만일 TK 필드가 같다면 그 원소는 제거한다. I_LIST의 index를 증가시키며 더 이상 교집합이 없을 때까지 이를 반복한다. I_LIST[i]의 원소중에서 커널의 기대치가 가장 큰 원소가 커널 추출에 효과적이므로 그 원소를 TK를 선택하며, KCST를 참조하여 원래 cube-free형태로 재변환 한다. 여기서 선택된 cube-free가 찾고자 하는 CFCD이다.

단계 4 : 선택한 커널을 중간 변수로 대치하고 대치한 변수를 함수 리스트에 등록시킨 후, 선택된 원소의 해당 함수를 중간 변수로 표현한다.

그림5에 커널 교집합의 알고리듬을 보였다. 여기서 index j는 원소 번호를 나타낸다. 다음에 알고리듬의 적용의 한 예를 나타내었다.

예) OR-FANIN constraint=2일때

F=ad+bd+cd+af;

G=ae+be+ce+dg+fg;

H=be+ce+dh+fh;

커널은

K(F)={a+b+c, d+f}

K(G)={a+b+c, d+f}

K(H)={b+c, d+f}

KCST={(a, 1), (b, 2), (c, 3), (d, 4), (f, 5)}

이상에서 OR-FANIN을 만족하지 않는 (a+b+c) 커널에서 VP쌍을 구하면

VP(a, 2, 2,), VP(b, 3, 3), VP(c, 3, 3).

그러므로 FANIN을 만족하는 커널 집합으로 재구성하면,

K(F)={b+c, d+f}

K(G)={b+c, d+f}

K(H)={b+c, d+f}

이와 같은 (b+c) 커널과 (d+f)로 위의 식을 최적화 시키면

```
X=b+c;
Y=d+f;
F=Xd+aY;
G=(a+X)e+Yg;
H=Xe+Yh;
```

가 되나, greedy 한 방법으로 (a+b+c)로 커널을 선택하여 최적화를 수행하면 이와는 다른 결과를 얻게 된다.

```
Kernel_Intersection(kernel_list, OR_FANIN)
{
  Build KCST from kernel-list;
  Find TK from kernel_list;/* TK means Translated Kernel */
  Insert TK to I_LIST[0];
  /* I_LIST likes 2-dimensional array structure. */
  i=0 ;
  while (I_LIST[i] has more than one element) {
    n=number of element in I_LIST[i];
    for(j=0; j<n;j++) {
      TK[j]=TK field of j-th element in I_LIST[i];
      MK[j]=Evaluate VTK(TK[j])];
      for(k=j;k<n;k++) {
        TK[k]=TK field of k-th element in I_LIST[i];
        TransKernel=TK[j]∩TK[k];
        if(TransKernel has more than 2 variables)
          Insert TransKernel to I_LIST[i+1];
        if(TK[j]≡TK[k])
          Delete TK[k] from I_LIST[i];
      }
    }
    i++;
  }
  /* from I_LIST, select a kernel */
  depth=size of I_LIST array;
  Select_TK=first TK element of I_LIST[0];
  for(i=0;i<depth;i++){
    n=number of element of I_LIST[i];
    for (j=0;j<n;j++) {
      if(Merit of Select_TK<MK[j])
        Select_TK=VTK(TK[j]);
    }
  }
  selected_kernel=ITK(Select_TK);
  return(selected_kernel);
}
```

그림 5. 커널 교집합 프로시저
Fig. 5. Kernel intersection procedure.

2. 공통 cube의 추출(common cube extraction)

커널 추출을 수행한 후 공통 cube 추출 및 대체를 수행하며, 이는 함수를 구성하는 cube중에서 같은 형태의 cube를 찾아 중간 변수로 치환하여 함수를 표현하는 과정이다. 커널 추출은 함수간의 공통 커널이 대상이나, 이 과정은 모든 함수를 구성하는 cube를 대상으로 공통된 cube를 중간 변수로 대치하여 전체 비용을 줄인다.

알고리듬을 그림 6에 기술하였다. F는 함수 리스트에 속한 모든 함수의 집합을 의미하며 cube 리스트는 F에 속하는 모든 cube와 그 교집합을 갖고 있다. NUM(C)는 cube C가 F에서 나타난 횟수이며, Selected-Cube는 F에서 가장 많이 나타난 cube로 이것을 함수로 대치하여 F에 삽입하고 이 cube가 나타난 함수를 재조정 한다.

```
Common_cube_substitution (F)
{
  cube_list=φ;/* cube_list:a set of cubes*/
  repeat{
    n=number of cubes in F;
    for(i=0;i<n;i++){
      C_i=i-th cube in F;
      m=number of cubes in cube-list;
      for(j=0;j<m;j++) {
        C_j=j-th cube in cube-list;
        if(C_i=C_j) then increase NUM(C_j);
        else {
          C=C_i ∩ C_j;
          if (C∈cube-list) then increase NUM(C);
          else if (C≠φ) then
            Insert C to the cube-list;
        }
        if (C_i∉cube-list) insert C_i to the cube-list;
      }
      MaxCube=NUM(C_0);
      Selected_Cube=C_0;
      for (i=1;i<m;i++);/*m:# of cubes in cube_list */
        if (NUM(C_i)>MaxCube) {
          MaxCube=NUM(C_i);
          Selected-Cube=C_i;
        }
      }
      Create intermediate variable g for SelectedCube;
      Insert g to the function list;
      if (LC(g)=AND-FANIN)
        Do-satisfy-fanin-constraint(g);
      m=number of cubes in cube-list;
      for (i=0;i<m;i++) {
        Ci=i-th cube in F;
        if(C_i=Selected-Cube) then replace C_i with g;
        else if (C_i⊃Selected_cube) then
          replace C_i with {g ∩ (C_i/g)};
      }
    } until no more same cube;
  }
```

그림 6. 공통 cube 대체 프로시저
Fig. 6. Common cube substitution procedure.

Ⅳ. 부분 최적화(Local Optimizations)

두개 이상의 함수를 고려한 전체 최적화가 끝나면 각각의 함수에 대해 literal의 수를 줄이는 부분 최적화 단계를 수행한다. SMiLE은 공통인 literal을 찾아 factored form을 형성하는 factoring 과정과 중간 함수를 반전(complement)시키는 것이 전체 비용을 감소시킬 수 있다면 중간 함수를 반전시키는 위상조정(phase assignment) 과정, 그리고 중간 변수를 포함하는 cube가 불필요한 항(redundent term)을 가지고 있는가를 조사하여 있다면 don't care set을 이용, 불필요한 항을 없애주는 단순화 과정으로 부분 최적화를 수행하였다.

1. Factoring

Factoring은 한 함수내에 공통으로 들어있는 literal 또는 cube를 묶어 factored form으로 나타냄으로써 literal의 수를 감소시키는 과정이다.[3,6,9] Factoring의 효과는 나눔수를 선택하는 방법과 오퍼레이션을 수행하는 방법에 따라 좌우되며, 나눔수로 literal을 선택하고 algebraic 오퍼레이션을 수행하는 LF(literal factoring), Boolean 오퍼레이션을 수행하는 BF(boolean factoring), level-0 커널을 나눔수로 택하고 휴리스틱 정보를 사용하여 오퍼레이션을 수행하는 QF(quick factoring)등이 있다.[5,6,11]

SMiLE에서 사용한 factoring은 extraction 과정의 커널 프로시져를 이용 level-0 커널을 생성한 후 factoring의 효과가 가장 크도록 이중에서 가장 많은 literal을 포함하며 AND-OR fanin 제약 조건을 만족하는 커널을 선택하여 algebraic 오퍼레이션을 수행했다. 나눔수의 선택은 커널 트리를 형성한 후 트리의 깊이는 factored된 literal의 수와 일치하므로 가장 트리의 깊이가 깊고 절약되는 literal의 수가 많은 커널을 선택한다. 나눔수를 선택한 후 함수 f에 algebraic division을 행하면 몫 h와 나머지 r을 얻는다. 선택한 함수 f에 대하여 더 이상의 나눔수가 존재하지 않을 때까지 반복하며 함수 리스트 F의 모든 함수에 대하여 factoring을 행한다.

2. 위상 조정(Phase Assignment)

위상 조정은 커널 추출로 중간 변수를 생성한 후 전체적으로 inverter수를 감소시키기 위한 과정이다.[5,8] 전체 함수집합 내의 중간 변수들은 정(positive)과 부(negative)의 두가지 형태로 표현 가능하다. Literal이 논리값 1로 사용될 때와 논리값 0으로 사용될 때의 literal의 전체 증감을 계산하여 이득이 있는 논리값으로 중간변수를 정의하면 inverter의 수를 감소시킬 수 있다. 이를 수행하기 위하여 모든 중간변수가 나타난 수를 계산하여 2회 이상 나타난 중간 변수에 대해 complement 시킨 후 이것이 전체 비용을 감소시킨다면 이 변수가 나타난 모든 함수를 재조정하고, complement시킨 변수를 함수 리스트에 등록한다.

그림7은 fanin 제약을 고려하였을 때의 위상 조정 알고리듬을 나타낸 것이다. 이 경우 함수 리스트 F의 함수 f가 변수로써 함수 G(=F−f)에서 f와 f'의 두가지 형태가 나타나면 위상을 바꾸어도 inverter의 절약은 없다.

```
Phase_Assignment(F) /*F:a set of functions */
{
  for each f∈F do{
    if(both f and f' used in some g∈F) then exit;
    h=COMPLEMENT(f) ;
    if h can not satisfy fanin constraint{
      Do_satisfy_fanin constraint(h);
      if cost(h)>cost(f) then continue;
    }
    if (INVERTER_COUNT(f)>INVERTER_COUNT(h){
      f=h;
      Change the phase of f;
      Adjust g using f';
    }
  }
}
```

그림 7. 위상 조정 프로시져
Fig. 7. Phase assignment procedure.

3. 단순화(Simplification)

주어진 Boolean 함수는 동등하나 형태가 다른 함수로 표현할 수 있다. 논리함수의 경우 논리 최소화 과정에서 주 입력변수 또는 중간 변수로 구성된 함수가 생성되어 이 변수가 다른 함수를 표현한다. 이때 함수는 불필요한 변수를 포함하는 수가 있으며 이 경우 함수의 비용이 증가하게 된다. 이런 불필요한 변수는 dc-set(don't care set)을 이용하여 제거하고, 보다 간단한 형태의 함수를 구하는 과정이 단순화이다. 일반적으로 Boolean 함수에서 dc-set은 다음과 같은 조건을 만족한다.[5,10]

$$dc\text{-}set = v'w + v\,w'$$

여기서, v는 중간 변수이고 w는 v를 표현하는 expression이다.

예를 들어 k=ab, X=ak+c라 하면, dc-set=abk'+(a'+b')k이고 이 dc-set을 이용하여 X=k+c로 단순화할 수 있다. 이 조건은 Boolean 함수에서 변수들 사이의 관계를 함축적으로 나타내며, 이 관계의 이용은 다단 논리 최적화의 필수 과정이다. SMiLE은 단순화를 보다 효율적으로 수행하기 위하여 on-set, off-set 그리고 dc-set을 이용하는 LBM(literal blocking matrix)[4,10] 방법을 사용하였다. LBM의 열은 cube를, 행은 LBM을 구성하는 모든 cube를 구성하는 literal을 나타내며, on-set과 off-set의 충돌(on-set과 off-set에 있는 literal의 위상이 반대인 것) 이 있는 위치를 '1'로, 다른 부분은 '0'으로 만든 행렬이다. On-set과 off-set의 구분은 하나의 변수간에 충돌이 존재하면 가능하므로 충돌이 중복되어 있는 literal 중에 비용을 최소로 하는 literal만을 남긴다.

그림8은 단순화의 알고리듬을 기술한 것으로 함수 Simplify()는 위에서 설명한 불필요한 변수를 이단 논리 최소화를 선별적으로 적용하여 제거하는 과정이다. 프로시져 Minimum_Literal_Support는 주어진 LBM에서 만일 열에서 단 하나의 '1'만이 존재하면 이것은 제거할 수 없는 literal (essential literal)이므로 EssLitList(essential literal list)에 삽입하고 그 행과 열을 제거하며, 한 행이 다른 행에 포함 된다면 그 행 역시 제거한다. 이 프로시져를 반복하여 더이상 줄일 수 없을때까지 LBM의 행과 열을 줄인다. LBM 구성중에 제거 가능한 행이란 선택한 행에서 '1'인 각 열에 '1'의 수가 2개 이상인 경우(conflict가 2개 이상 발생)를 의미한다. 구성한 LBM에는 가장 적은 literal수를 갖는 LBM이 존재하므로 이를 구하기 위해 각 LBM의 비용(구성 literal수)을 계산하여 가장 작은 비용의 LBM을 찾는다. 이 LBM을 구성하는 literal을 EssLitList에 삽입하고 프로시져 Reconstruct()에서 EssLitList와 on-set을 가지고 최소의 literal을 갖는 함수를 재구성한다.

```
SIMPLIFICATION(F)
{
  for each function f in F
    dcset=v'f+f'v;
    g=Simplify (f, dcset);
    g"=FACTORING(g);
    f"=FACTORING(f);
    cost_g=cost evaluation of g" ;
    cost_f=cost evaluation of f" ;
    if (cost_g<cost_f) then f=g" ;
  }
}
Simplify(f, dcset)
{
  onset=f-dcset;
  offset=f'-dcset;
  Make LBM from onset and offset;
  EssLitList=Minimum_Literal Support(LBM, onset);
  x=Reconstruct(EssLitList, onset);
  /* delete literal(∈onset) not in EssLitList */
  return(x);
}
```

그림 8. 단순화 프로시져
Fig. 8. Simplification procedure.

V. 실험 결과 및 고찰

SMiLE은 UNIX 환경하에서 C로 구현되었다. MIS는 command script의 기술 순서에 따라 실과가 달라지므로, 결과 비교를 위해 실험에 사용한 MIS command script는 MIS 2.0에서 제공하는 standard script(~cad/lib/misII/lib/script)를 따랐다.

표 1은 표준 테스트 회로로 사용되는 MCNC benchmark 회로에 대해 MIS version 2.0과 SMiLE의 실험 결과를 비교한 것이다. 표 1의 A는 fanin 제약없는 경우의 MIS와 SMiLE의 결과를 나타낸 것이며, 표 1의 B는 fanin 제약 조건으로서 MIS에서는 and-3 gate와 or-2 gate를 매핑한 결과이다. 이 표에 따른 전체 회로에 대한 결과는 fanin 제약을 주지 않았을 경우에는 MIS의 결과가 14.4% 좋으며 SMiLE에서 커널 추출과 factoring시에 fanin 제약 조건

표 1. MCNC benchmark 회로에 대한 실험 결과
Table 1. Experimental results for MCNC benchmark circuits.

Benchmark	#ins	#outs	initial	A:non-constraint		B:(fanin and3, or2)	
				MIS	SMILE	MIS	SMILE
				#lits.	#lits.	#lits.	#lits.
5xp1	7	10	293	155	204	210	218
con1	7	2	23	23	30	34	32
f2	4	4	36	32	32	40	32
f51m	8	8	36	160	227	220	241
misex1	8	7	122	72	73	97	76
misex2	25	18	188	113	127	145	146
rd53	5	3	140	62	65	91	66
rd73	7	3	840	140	180	206	200
sao2	10	4	489	192	249	267	264
z4ml	7	4	252	42	54	60	54

을 주어 수행한 결과는 SMiLE이 3% 좋은 결과를 보인다.

표 1의 결과로부터 단순히 알고리듬에 의한 면적 최소화 과정 이후에 테크놀러지 매핑시키는 방법보다 사용될 gate library의 fanin 제약 조건을 커널 추출과 factoring시에 고려할 경우 좋은 결과를 가져옴을 알 수 있다.

MCNC benchmark 회로에 대해 fanin 제약 조건을 고려하여 SMiLE을 수행시킨 결과 회로와 면적의 최소화만을 고려하여 SMiLE을 수행시킨 결과 회로 각각에 대하여 테크놀러지 매퍼로 최적화시킨 결과를 표 2에 나타내었다. (A)는 fanin 제약 없이 SMiLE을 수행시킨 후 테크놀러지 매핑시킨 결과이고, (B)는 fanin 제약 하에 SMiLE을 수행시킨 후 테크놀러지 매핑시킨 결과로서 비교하면 fanin 제약 조건을 고려하여 기술 독립 최적화시킨 결과가 평균 15.6%의 면적 감소를 가지 왔다. 계산된 면적은 주어진 라이브러리의 게이트의 면적을 기준으로 회로 구성 게이트들의 면적 종 합으로 구하였으며 사용된 라이브러리는 총 26개의 게이트로 이루어지 있다. 표 1에서 보인 SMiLE 수행결과는 fanin을 고려할 경우 회로의 면적이 증가되므로 fanin 제약 없이 면적치만을 cost function으로 사용한 결과가 더 효율적으로 보이지만 실제 테크놀러지 매핑을 수행한 후의 결과인 표 2는 fanin 제약을 고려하여 주는 것이 면적의 감소에 유리함을 보여 주고 있다. 이는 실험에 쓰인 테크놀러지 매퍼 silos-II[16]가 그래프 커버링 방식을 이용한다는 점에서 그 원인을 찾을 수 있다. 그래프 커버링에 의한 테크놀러지 매핑에서는 입력 회로의 DAG 형태를 유지하면서 입력 회로의 모든 게이트를 2-input NAND 게이트와 인버터로 재구성시켜 라이브러리 게이트를 매핑시켜 주므로 fanin 제약하에 최적화된 회로는 2-input NAND 게이트로 재구성 될 경우 그렇지 못한 회로에 비하여 더 적은 면적을 차지하게 된다.

표 2. 테크놀러지 매핑 후 결과
Table 2. Results after technology mapping.

bench mark	(A) non-constraint Area	(B) fanin(and3, or2) Area
5xp1	3830	3000
9sym	6890	4960
con1	430	500
f2	440	440
misex1	1780	1360
misex2	3210	2380
rd53	1360	1210
rd73	4730	2760
sao2	5120	3550
z4ml	1220	950

기술 독립 다단 논리 최적화 시스템과 테크놀러지 매퍼는 서로 유기적 관계에 있으며 논리 설계 시스템의 front-end인 다단 논리 최적화 시스템에서 fanin과 같은 기술 의존적인 cost 조건을 미리 고려하여 줌으로써 보다 나은 결과를 보인다.

Ⅵ. 결 론

본 논문에서는 알고리듬적 접근 방식으로서 커널 추출과 factoring시에 fanin 제약 조건을 고려하여 다단 논리 회로를 최적화시키는 SMiLE 시스템에 대하여 기술하였다. 커널 추출을 위하여 수행 시간의 부담이 적으면서 reasonable한 결과를 얻을 수 있는 algebraic division 방식을 이용하였으며, 추출된 커널 중 최적의 커널을 선택하기 위한 비용 함수에 커널 선택에 따른 면적의 감소치만이 아닌 사용자에 의해 주어진 fanin 제약 조건을 고려해줌으로써 원하는 fanin 제약 하에서 최소 면적을 가지는 회로를 설계할 수 있게 하였다. 알고리듬 레벨에서 기술 독립 최적화된 논리 회로가 칩으로 만들어지기 위해서는 반드시 기술 의존적인 매핑 과정을 거쳐야 하므로 알고리듬 레벨의 최적화 단계에서 기술 매핑을 고려하여 최적화를 수행한다면 기술 매핑시 보다 좋은 결과를 얻을 수 있다. SMiLE의 최적화 결과는 그래프 커버링 방식을 이용한 테크놀러지 매퍼 soilos-II[16]의 매핑 결과에 직접 영향을 미친다. 실험 결과의 표 2에서 보인 결과는 기술 독립적인 논리 최적화 과정에서 fanin 제약을 미리 고려할 경우 12.7% 향상된 테크놀러지 매핑 결과를 얻을 수 있음을 보여준다.

앞으로 테크놀러지 매핑시 함수를 보다 최적화할 수 있도록 하기 위하여 알고리듬 레벨에서 fanout 제약 조건을 추가하고, library에서 제공되는 여러 종류의 compound gate들에 대한 매핑도 지원할 수 있도록 연구가 지속될 것이다.

參 考 文 獻

[1] A.J. De Geus and D.J. Gregory, "The SOCRATES logic synthesis and optimization system", in *Design Systems for VLSI Circuits*, G. De Micheli(Ed.), Martinus Nijhoff Pub.: Dordrecht, Netherlands, pp. 473-498, 1987.

[2] D. Bostick, G.D. Hachtel, R. Jacoby, M.R. Lightner, P. Moceyunas, C.R. Morrison, D.

Ravenscroft, "The Boulder Optimal Logic Design System", in *Proc. IEEE Int. Conf. on CAD*, pp. 62-65, Nov. 1987.

[3] M.A. Breuer, *"Design automation of Digital systems(Vol. 1) theory and Techniques"*, Prentice-Hall : Englewood Cliffs, NJ., 1972.

[4] R.K. Brayton, G.D. Hachtel, C.T. McMullen, and A. Sangiovanni-Vincentelli, *"Logic Minimization Algorithms for VLSI Synthesis"*, Kluwer Academic Publishers: Boston, Mass., 1985.

[5] R.K. Brayton and R. Rudell, A. Sangiovanni-Vincentelli, and A. Wang, "MIS : A multiple-level logic optimization system", *IEEE Trans. Computer-Aided Design*, vol. CAD-6, no. 6, pp. 1062-1081, Nov. 1987.

[6] R.K. Brayton. "Factoring logic functions". *IBM J. Res. Develop.*, vol. 31, no. 2, pp. 187-198, Mar. 1987.

[7] R.K. Brayton, E. Detjens, S. Krishna, T. Ma, P. McGeer, L. Pei, N. Pjillips, R. Rudell, R. Segal, A. Wang, R. Yung, A. Sangiovanni-Vincentelli, "Multiple-Level Logic Optimization System", in *Proc. IEEE Int. Conf. on CAD*, pp. 356-359, Nov. 1986.

[8] R.K.Brayton, R. Camposano, G. De Micheli, R. Otten and J. von Eijndhoven, "The yorktown silicon compiler", *in Silicon Compilation, D.* Gajski(ed.), Addison Wesley Pub.: Reading, Mass., 1987.

[9] R.K. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, A. Wang, "Multi-level logic optimization and the rectangular covering problem", *in Proc. IEEE Int. Conf. on CAD*, pp. 66-69, Nov. 1987.

[10] R.K. Brayton, "Algorithms for multi-level logic synthesis and optimization", *in Design Systems for VLSI Circuits*, G. De Micheli (Ed.), Martinus Nijhoff Pub.: Dordrecht, Netherlands, pp. 197-248, 1987.

[11] R.K. Brayton, G.D. Hachtel, and A.L. Sangiovanni-Vincentelli, "Multilevel logic synthesis", *IEEE Proceedings*, vol. 78, no. 2, pp. 264-300, Feb. 1990.

[12] S. Devadas, A. Wang, R.K. Brayton, and A. Sangiovanni-Vincentelli, "Boolean Decomposition in Multi-Level Optimization", *in Proc. IEEE Int Conf.* on CAD, Nov. 1988, pp. 290-293.

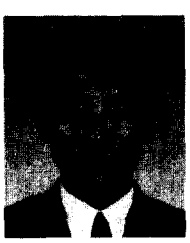
[13] S.J. Hong, R.G. Cain, and D.L. Ostapko, "MINI : A Heuristic Approach for Logic Minimization," IBM J. Res. Develop., Vol. 18. Sept. 984, pp. 443-458.

[14] 이 재형, 황 선영, "성능 구동 논리회로 자동 설계 시스템", 대한 전자공학회 논문지, 28-A 권, 1호, 1991년 1월, pp. 74-84.

[15] 전 홍신, 이 해동, 황 선영, "서강 실리콘 컴파일러의 High Level Synthesis System의 설계", 대한 전자공학회 논문지, 28 A권, 6호 1991년 6월, pp. 488 499.

[16] 김 태선, 황 선영, "논리 회로의 기술 매핑 시스템 설계", 대한 전자 공학회 논문지, 20 A권 2호, 1991년 2월, pp. 88 99.

著 者 紹 介



林 春 奭(準會員)

1963年 9月 18日生. 1990年 2月 서강대학교 전자공학과 졸업. 1992年 2月 서강대학교 전자공학과 대학원 졸업. 1992年 2月～현재 금성사 정보기기연구소 연구원. 주관심분야는 CAD 시스템, Computer Architecture 및 VLSI설계 등임.

黃 善 泳(正會員) 第28卷 A編 第1號 參照

1954年 5月 20日生. 1976年 2月 서울대학교 전자공학과 졸업. 1978年 2月 한국 과학원 전기 및 전자공학과 공학석사 취득. 1986年 10月 미국 Stanford대학 공학박사학위 취득. 삼성 반도체 주식회사 연구원, Stanford 대학 CIS연구소 연구원, Fairchild Semiconductor 기술 자문. 1989年 3月～현재 서강대학교 전자공학과 교수. 주관심분야는 CAD시스템, Computer Architecture 및 Systems Design, VLSI 설계 등임.