論文93-30A-11-15

다단 논리합성을 위한 성능 구동형 회로 다단기

(Performance-Driven Multi-Levelizer for Multilevel Logic Synthesis)

李 載 興*, 鄭 正 和*

(Jae Heung Lee and Jong Wha Chong)

要 約

본 논문에서는 설계자의 제약 조건및 테크놀로지 매핑을 고려한 새로운 성능 구동형 회로 다단기를 제안한다. 성능 구동형 회로 다단기는 2단 논리회로로 부터 설계자의 요구에 맞도록 면적, wire 수, 함수레벨, 지연시간, fanin, fanout 등을 고려하여 다단 논리회로로 변환하는 다단 논리 합성기이다. 셀 매핑 과정을 통하여 셀 라이브러리 정보를 참조하여 논리회로의 성능을 계산하였으며, 그 성능의 향상 정도에 따라 cost 함수를 정의하고, 이 cost 함수에 따라 회로 다단화 과정을 반복적으로 수행하는 휴리스틱 알고리듬을 제안하였다. 또한 MCNC benchmark 데이타를 사용하여 실험하였으며 그 결과 본 논문에서 제안한 방법이 우수함을 알 수 있었다.

Abstract

This paper presents a new performance-driven multi-levelizer which transforms a two-level description into a boolean network of the multilevel structure satisfied with user's costraints, such as chip area, the number of wires and literals, maximum delay, function level, fanin, fanout, etc..

The performance of circuits is estimated by reference to the informations in cell library through the cell mapping phase, and multi-levelization of circuits is constructed by the decomposition using the kernel and factoring concepts. Here, the saving cost of a common subexpression is defined to the sum of area and delay saved, when it is substituted. The experiments with MCNC benchmarks show the efficiency of the proposed method.

Ⅰ. 서 론

최근 반도체 집적 회로는 종래의 설계 전문가의 수작업으로 설계 가능하던 한계치를 초월하여 단일 칩 내에 수 백만개의 트랜지스터가 집적되는 VLSI 단계에 이르면서 컴퓨터를 이용한 설계자동화에 대한 요구가 크게 증가하고 있다.

종전의 설계 전문가의 경험이나 능력에 전적으로 의존하는 수작업 설계 방법으로는 설계 면적에 대한 최적화 목표는 어느 정도 달성할수 있으나 설계 시간의 증가, 오류 발생 요인 증가등의 단점을 가지고있다.

따라서 디지탈 시스템 설계에 대한 정확성의 보장과 설계 시간을 단축 하기위해 CAD(Computer Aided Design) 기술을 VLSI 설계에 활용하는 설계 자동화 시스템 개발에 대한 연구가 매우 활발히 진행되고 있다. CAD 시스템의 장기적인 목표는 동작기

* 正會員, 漢陽大學校 電子工學科
(Dept. of Elec. Eng., Hanyang Univ.)
接受日字 : 1992年 2月 16日

술로부터 설계자에 의한 사양을 만족하는 최적에 근사한 해를 구하는 실리콘으로의 자동합성에 있다.[1] 이러한 CAD의 능력은 ASIC(Application Specific Integrated Circuit) 시장의 요구가 빠른속도로 증가함에따라 중요성이 증대되고 있다. CAD 분야는 크게 상위레벨, 논리레벨, physical 레벨 합성으로 구분되며 특히 논리합성에서 논리설계 자동화에의한 설계는 수작업에 의한 설계보다 많은 redundancy를 포함하게 될 뿐 아니라, 설계자의 요구 조건에 맞는 논리회로를 얻는데 어려움이 있다. 이를 해결하기 위하여 논리합성은 크게 PLA 기술에 적합하며 현재 상당한 연구 수준에 도달한 2 단 논리합성과 최근 게이트 어레이, 스탠다드 셀 등과 같은 실현 기술의 발달과 함께 각광을 받고있는 다단논리합성으로 구분할 수 있으며, 다단 논리합성은 기술 독립적인 단계와 기술 의존적인 단계로 구분하여 설계자의 요구 조건에 맞는 논리회로를 얻고자 시도하고 있다. MIS [2] 의 경우 지연시간을 무시하고 칩 면적과 관련있는 문자 수만을 고려하여 다단 논리회로를 구성하고 collapsing 과 restructuring 을 반복적으로 수행하여 면적과 지연시간이 절충된 논리회로를 얻고 있다.[13] 따라서 실현될 기술을 고려하지 않고 다단 논리합성을 수행하기 때문에 설계자가 요구하는 다단 논리회로를 쉽게 얻을 수 없다.

따라서, 본 논문에서는 설계자의 제약 조건 및 테크놀로지 매핑을 고려한 새로운 성능 구동형 다단 논리합성기를 제안한다. 2단 논리회로로 부터 설계자의 요구에 맞도록 면적, wire 수, 함수레벨, 임계경로 지연시간, fanin, fanout 등을 고려하여 다단 논리회로로 변환하는 다단 논리 합성기로써, 셀 매핑 과정을 통하여 셀 라이브러리 정보를 참조하여 논리회로의 성능을 계산하였다. 특히 면적과 지연시간의 향상 정도에 따라 cost 함수를 정의하고, 이 cost 함수에 따라 회로 다단화 과정을 반복적으로 수행하는 휴리스틱 알고리듬을 제안한다. 또한 MCNC benchmark 데이타를 사용하여 실험하였으며 그 결과 본 논문에서 제안한 방법이 우수함을 알 수 있었다.

Ⅱ. 성능 구동형 회로 다단기의 전체적인 구조

다단 논리 회로 생성은 2단 다출력 함수로 부터 최소의 면적과 지연시간및 설계자의 제약조건을 만족하는 factored 형태의 다단 논리회로 표현식을 생성하는 작업이다.

본 논문의 성능 구동형 회로 다단화 과정의 전체 흐름도는 그림 1. 과 같다.

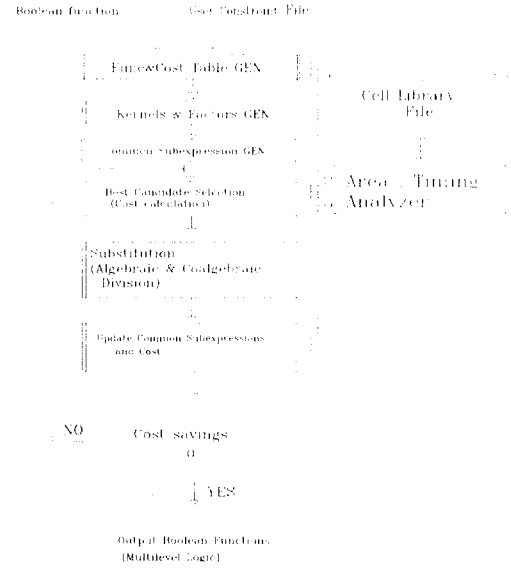

그림 1. 성능 구동형 회로 다단기의 전체 흐름도
Fig. 1. Overall flow of performance-driven multi-levelizer.

먼저 다단화될 주어진 Boolean cover형태의 입력 데이타와 fanin, fanout, 함수레벨,면적과 지연시간에 대한 가중치등을 기술한 설계자 제약조건을 입력으로 하여 주어진 입력 데이타에 대한 면적, 지연시간, wire수, 함수레벨 등을 면적 지연시간 분석기를 통하여 함수및 cost 테이블을 구성한다. 이때 면적과 지연시간을 계산하기 위하여 셀 매핑을 통하여 셀에 대한 라이브러리 정보를 참조 하는데 각 셀에 대한 면적, pin수, 평균 입력 캐패시턴스, 단위 캐패시턴스에 대한 지연시간 증가율, 내부 고유 지연시간들을 포함하고 있다. 그 다음 주어진 함수를 다단화 시키기 위하여 필요한 kernel과 factor를 생성하고 kernel과 cube intersection 을 통하여 공통 kernel 과 cube들을 생성하는데 이들이 공통 부분표현식들이 된다. 이들 공통 부분표현식이 대치되어 중간변수가될 후보자들이며, 각 후보자들에 대해서 대치할 경우에 면적과 지연시간의 향상정도를 나타내는 모든 cost를 계산한다. 가장 cost를 만족하는 하나의 후보자를 선택하여, 그 후보자를 포함하는 함수를 새로운 중간변수로 대치하게 되는데 이때, 주어진 함수값을 선택된 후보자로 나눔으로써 실현할 수 있다. 그 다음 대치된 후보자로 인하여 발생되는 모든 공통 부분표현식을 새롭게 변경시킨다. 이와 같은 작업을 반복 수행함으로써 더 이상의 cost 향상이 없으면 입력과 마찬가지로 Boolean cover 표현식으로 출력을 내면서 회로 다단화 과정을 마치게 된다.

Ⅲ. 부울회로에 대한 면적과 지연시간 분석기

주어진 다출력 함수 F = {$f_1, f_2, \cdots f_n$}의 부울회로로 부터 함수 레벨, 함수 형태, 면적, 지연시간을 분석하는 것으로 전체적인 흐름도는 그림 2.와 같다.

```
Area_Timing_Analizer(F)
  Function_level(F);
    /* 부울회로에 대한 노드레벨 계산 */
  Cell_mapping(F);
    /* 각 노드의 라이브러리의 셀로 매핑 */
  Area_Calculation(F);
    /* 각 노드의 면적의 합 계산 */
  Delay_calculation(F);
    /* 각 노드에 대한 전체 지연시간 계산 */
```

그림 2. 부울회로에 대한 면적 및 지연시간 분석 흐름도.

Fig. 2. Overall flow of area and timing analyzer.

먼저, 부울회로의 함수레벨를 계산하는데 이는 함수레벨에 따라 지연시간이 관계가 있고 지연시간 계산시에 레벨에 따라 단계적으로 계산할 필요가 있기 때문이다. 그 다음 각 노드를 라이브러리의 셀로 매핑하여 셀 형태를 결정하게 된다. 결정된 라이브러리의 셀 형태에 따라 라이브러리 정보를 이용하여 면적과 지연시간을 계산하는데, 이때 각 노드에 대해서 최대 fanin 수를 초과하는 게이트들에 대해서는 게이트 분할을 수행하여 계산한다. 지연시간 계산에서는 각 노드에 대한 local 지연시간과 primary 입력으로 부터의 전체 경로 지연시간으로 나누어 계산하게 된다.

1. 함수 레벨의 계산 알고리듬

다음 그림 3. 은 주어진 부울회로에 대한 함수레벨 계산 알고리듬을 나타내고 있다.

```
Function_level(F)
  while(F ≠ φ) {
    for(each node f_i ∈ F) {
      if(supp(f_i) = all PI) /* f_i 의 입력이 모두 primary 입력 */
        flevel[f_i] = 1;
      else
        for(each f_j ∈ supp(f_i) and  f_j ∉ PI) {
          if(f_j 가 함수레벨이 결정되지 않았다면)
            insert(F1, f_i); break; /* 결정되지않은 노드 집합에 저장 */
          else
            flevel[f_i]=max(flevel[f_i], flevel[f_j])+1;
        }
    }
    F = F1
  }
```

그림 3. 함수레벨 계산 알고리듬

Fig. 3. The algorithm for calculating the function level.

주어진 부울회로 F에 대한 함수레벨 계산은 부울회로의 각 노드에 대해서 그 노드의 support를 고려하여 결정하게 되는데 supp(f_i) 는 함수 f_i 의 입력 변수의 집합으로 정의한다. 한 노드의 모든 support가 primary 입력이면 그 노드의 함수레벨은 1로 결정되고 그렇지 않으면 그 support중에서 가장 큰 함수레벨 + 1로 그 노드의 함수레벨이 결정된다. 만약 support들 중에서 아직 함수레벨이 결정되지 않은 노드들의 경우에는 그 노드들의 집합을 함수 F1에 포함시켜서 반복 수행시키게 되는데 이것은 primary입력으로 부터 각 노드의 레벨을 결정되는 것을 나타내며, 각 노드는 sum of product 표현식으로 주어지기 때문에 실제 회로의 게이트 레벨과는 다르다.

2. 셀 매핑

주어진 부울회로에서 각 노드의 셀 형태를 결정하여 라이브러리의 셀 정보를 이용하여 면적과 지연시간을 계산하는데 이용할 수 있다. 셀 매핑은 그래프의 패턴 매칭과는 달리 함수레벨에서의 매칭 방법을 사용하고 있다. 부울 회로에서 각 노드에 대한 셀 매핑에 대한 알고리듬은 그림 4.와 같다.

```
Cell_mapping(f_i)
  if((NC(f_i) = 1) and NL(f_i) = 1) /* Single line type */
       return(SLINE or INV);
  else if(NC(f_i) = NL(f_i)) {    /* OR type 결정 */
       cell_type = OR2 + (NC(f_i) - 2);
  }
  else if(NC(f_i) = 1)              /* AND type 결정 */
       cell_type = AND2 + (NL(f_i)-2);
  else                               /* AND-OR type */
       switch(NC(f_i))  {
          case 2 : cell_type = find_two_cover(f_i); break;
          case 3 : cell_type = find_three_cover(f_i); break;
          case 4 : cell_type = find_four_cover(f_i); break;
          default: cell_type = SOP;
       }
  return(cell_type);
```

그림 4. 셀 매핑 알고리듬

Fig. 4. The algorithm for mapping to cells.

여기서, NC 와 NL 은 각각 큐브와 문자 수를 나타낸다.

위의 알고리듬에서 Single line 형태에서는 주어진 노드의 입력 신호선이 1개로써 그 신호선의 형태에 따라 INV,SLINE으로 결정되고 OR 또는 AND형태에서는 최대 5입력 OR 또는 AND게이트를 정의하여 5입력 이상인 경우 ORXX 또는 ANDXX를 return하게 되고 그 이하인 경우에는 해당하는 입력 게이트로 결정하였다. 한편, AND_OR 형태에서는 큐브 수에 따라 구분하였으며 2개의 큐브로 구성된 함수인 경우 find_two_cover()에서는 각 큐브의 입

력 수및 변수 수, 큐브 상호간의 distance및 극성등을 비교하여 XOR, XNOR, MUX21, AOI21, OAI21, AOI211,OAI22등을 결정한다. 3개 또는 4개의 큐브로 구성된 함수인 경우, find_three_cover()와 find_four_cover()에서는 OAI211과 AOI22를 결정하게 되는데 3개 이상의 큐브를 포함하는 함수에 대해서는 경우수가 상당히 많기 때문에 위의 방법으로 셀 형태를 결정하는데는 한계가 있으며 모든 경우수를 찾기에는 지나친 시간의 소비가 많기 떠문에 일반적으로 SOP를 return하여 primitive한 AND 와 OR게이트의 구성으로 생각하여 면적 및 지연시간에 대한 정보를 이용하게 된다.

3. 부울회로의 면적계산 알고리듬

부울회로에서 한 노드의 면적은 앞에서 이미 결정된 그 노드의 함수 형태에 따라 라이브러리의 면적 정보로 얻어지지만 함수 형태가 SOP, ANDXX, ORXX 로 결정된 노드는 라이브러리에서 정보를 제공하지 않기 때문에 다음 그림 5.와 같이 primitive한 AND 와 OR게이트의 면적 합으로 얻어진다.

```
SOP_size(fi) {          /* sum of product area */
    for(each cube Ci ∈ fi)
        area += AND_size(NL(Ci));
    area += OR_size(NC(fi));
    return(area);
}
AND_size(p) {           /* AND area */
    while( p > MAXAND) {
        n_gate = p / MAXAND;
        area += n_gate * lib_area(MAXAND);
        p = p - n_gate*(MAXAND-1);
    }
    if(p>1) {
        switch(p) {
            case 2 : cell_type = AND2 ; break;
            case 3 : cell_type = AND3 ; break;
            case 4 : cell_type = AND4 ; break;
            case 5 : cell_type = AND5 ; break;
        }
        area += lib_area(cell_type);
    }
    return(area);
}
```

그림 5. 부울회로에서 SOP(Sum of Product) 면적 계산 알고리듬

Fig. 5. The algorithm for calculating the area of SOP.

임의의 SOP형태의 함수 f_i에 대한 면적은 AND term들에 대한 면적들의 합과 그 term들의 OR term 면적의 합으로 얻어지는데 그들은 AND_size() 와 OR_size() 함수를 통하여 계산된다. 그림 5.의 AND_size(p) 에서 p는 그 term의 product size 를 나타내는데 만약, p가 MAXAND, 즉, AND게이트의 최대 fanin수 보다 클때는 최대 fanin AND 게이트로 게이트분할을 수행하여 최대 fanin 게이트들의 면적합으로 계산하고 OR_size(s) 도 마찬가지 방법으로 수행된다.

4. 부울회로의 지연시간 계산

본 논문에서 사용하는 게이트 g의 지연시간 모델은 다음과 같이 주어진다.

$$Delay(g) = ID(g) + TD(g) + WD(g)$$

(ID:intrinsic delay, TD:transition delay, WD:wire delay)

여기서 ID(g)는 게이트 g의 내부 고유 지연시간을 나타내며 게이트의 사용용도에 무관하게 일정한 상수값을 가지며 라이브러리에서 게이트의 입력핀에서 출력핀까지 지연시간을 상수값으로 저장하고 있다. TD(g)는 게이트 g의 전달 지연시간을 나타내며, 다음과 같이 주어진다. [8]

$$TD(g) = SLP(g) * \sum_{i \in FANOUT(g)} ICP(i)$$

여기서 SLP(g)는 게이트 g의 단위 입력 캐패시턴스당 지연시간의 증가율을 나타내며 ICP(i)는 게이트 i의 입력 캐패시턴스로 정의하며 게이트의 입력 pin에 따라 다르다. 따라서 각 입력 pin의 평균치가 라이브러리에 저장되어 있다. WD(g)는 wire 지연시간을 나타내며 다음과 같이 주어진다.

$$WD(g) = (WCP * g의\ fanout수) * SLP(g)$$

여기서 WCP는 1개 fanout 수 증가에 따른 wire 캐패시턴스의 증가율을 나타내며 전체 chip 크기에 따라 정해진다. [15]

앞에서 정의한 게이트에 대한 지연시간 모델을 이용하여 부울회로에 대한 각 path 의 지연시간을 계산하는데 primary input으로 부터 primary output으로 각 함수 레벨에 따라 함수의 지연시간을 구한다. 이때, 설계자의 fanin 제약 조건을 만족하지 않는 함수들에 대해서는 게이트 decomposition을 통하여 라이브러리의 지연시간 정보를 이용하여 각

path에 대한 longest path 지연시간을 계산함으로써 각 함수의 지연시간을 얻을 수 있다.

5. Cost 함수

부분 표현식 ci에 대한 Cost 함수는 그 부분 표현식을 새로운 중간 변수로 대치 하였을때, 면적과 지연시간의 향상 정도로 다음과 같이 정의한다.

$$COST(c_i)=\alpha * SVAREA(c_i)+\beta * SVDELAY(c_i),$$

여기서, α 와 β 는 면적과 지연시간의 가중치로 설계자에 의해서 주어진다.

Ⅳ. Decomposition 과 factoring

본 논문에서 사용하는 decomposition 알고리듬의 대략적인 흐름은 다음과 같다.

DECOM_KF(F)
  1) 후보 공통 부분 표현식 생성
While (COST)0인 후보가 존재)
  2) 가장 좋은 후보 부분 표현식 선택
  3) 새로운 변수로 대치
  4) 후보 부분 표현식들의 변경
ENDWhile

위 알고리듬의 기초는 후보 부분 표현식을 생성하고 선택된 부분 표현식을 새로운 중간 변수로 대치하는것으로써 이미 [2] 에서 정의한 kernel 들을 구하여 그 kernel 들의 공통 부분을 이용하여 얻어질 수 있다. 이 단계에서는 부울회로에서 함수를 공통으로 사용하여 전체 면적을 감소시키기 위하여 2 개 이상의 함수에 공통으로 존재하는 부분 표현식을 찾는다. 이들 공통 부분 표현식으로 부터 cost의 합을 가장 감소시키는 공통 부분 표현식을 선택하여 새로운 중간변수로 대치하게 되는데 이때 대수 나눗셈 알고리듬 [2] 을 도입하여 이용한다. 또한, 새로운 중간 변수로의 대치 때문에 발생되는 관련된 후보 부분 표현식을 수정한다. 위 알고리듬 각각에 대해서 좀더 자세하게 기술하면 다음과 같다.

1. 공통부분 표현식 생성

2단 논리회로로 부터 다단 논리회로를 생성하기 위하여는 주어진 부울회로로 부터 제일 먼저 새로운 중간노드를 생성하기 위한 후보자 즉, 공통부분 표현식들을 생성하여야 한다. 본 논문에서의 공통부분 표현식 생성 알고리듬의 전체 구성도는 그림 6.과 같다.

```
CSG(F) {    /* Common Subexpression Generation */
   for(each function fi∈F )
     (K,Fac)=KF_gen(fi);/*Kernel and Factor generation */
   CK = Common_kern_gen(K); /* Kernel intersection */
   Compl_common_kern_gen(CK); /* Complement of common kernel 첨가 */
   CC = Common_cube_gen(F); /* Common cube generation */
   return(CK ∪ CC);
}
```

그림 6. 공통부분 표현식 생성의 전체구성도
Fig. 6. Overall flow of the generation of common subexpressions.

먼저 주어진 부울회로의 각 노드에 대한 kernel 과 factor(co-kernel 이라고도 함)를 KF_gen( )에서 생성한다. Common_kern_gen( )은 서로 독립적인 kernel들 사이에 교집합을 구함으로써 공통 kernel, CK를 생성하도록 하였고, 또한 공통 kernel의 complement 도 공통 kernel에 포함시켰다. 그 다음 Common_cube_gen()에서 공통 cube를 생성하였는데, 주어진 함수의 cube 들에 대한 교집합으로 얻어진다. 공통 kernel, CK 와 공통 cube CC 합집합이 얻고자 하는 공통부분 표현식이 된다.

2. Kernel 및 Factor 생성 알고리듬

함수 f에 대한 kernel 및 factor 생성 알고리듬은 [2] 에서 정의 한 바와 같이 그림 7. 에 나타내고 있다.

```
KF_gen(f, fac, vindex) {
    for(i = vindex; i < NI(f); i++)  {
        cf = common_factor(f);
        if(cf = φ or {0,1,2,.....,vindex-1} ∩ cf = φ) {
            f0 = zero_cov(f, i);/*함수 f에서 i번째 변수가 zero인 부분함수*/
            f1 = one_cov(f, i);
            if(NC(f0) ≥ 2)
                PUTinput(fac, i, 0);/* fac 의 i번째 변수에 zero 를 삽입 */
                KF_gen(f0, fac, i+1);
            if(NC(f1) ≥ 2)
                PUTinput(fac, i, 1);
                KF_gen(f1, fac, i+1);
        }
    }
    if(level ≤ K_lev or NV(f) ≤ K_var)
        KF_insert(f, fac); /* kernel과 factor 를 저장 */
    return;
}
```

그림 7. Kernel 및 Factor 생성 알고리듬
Fig. 7. The algorithm for the generation of kernels and factors.

여기서, NI, NV, K_lev, K_var 은 각각 입력수, 변수 수, kernel 레벨, kernel 변수 수를 나타낸다.

먼저 argument vindex는 vindex보다 작은 모든 변수에 대하여 이미 kernel 생성 과정을 거쳤음을 의미한다. common_factor( )는 함수 f의 모든 cube 들이 공통으로 갖는 인수를 찾는 함수이며 조건 (cf

=ø or {0,1,2,…,vindex-1} ∩ cf = ø) 는 이중으로 kernel이 생성됨을 방지하기 위함이다. zero_cov( )와 one_cov( )는 함수 f에서 i번째 변수가 0 또는 1이 됨에 따라 f0와 f1로 분할하는 함수이며 이들 함수들이 cube의 수가 2개 이상인 함수들에 대해서만 kernel과 factor를 생성하도록 하고 그 kernel의 level에 따라 level 조건을 만족할 때만이 KF_insert( )라는 함수를 통하여 생성된 kernel과 factor를 저장하게 하였는데 이는 함수 f의 문자수가 증가함에 따라 지수적으로 증가하기 때문에 실제적으로 필요한 kernel만을 구하여 적용하기 위함이다. 뿐만아니라, 그 밖의 다른 조건을 즉 variable수, 입력수, cube수의 조건을 만족하는 kernel만을 생성하여 적용할 수도 있다.

3. 공통 kernel의 생성

앞에서 기술한 kernel 생성 알고리듬으로 부터 조건에 맞는 모든 kernel 들을 구하고나서, 서로 독립적인 kernel 들에 대한 kernel 들의 교집합을 구하는데 이는 kernel 사이에 공통함수를 생성하고자 하기 때문이다. [2] 이 문제를 해결하기 위하여 kernel들이 다음과 같이 주어졌다고 할때

$K=\{k_1, k_2, k_3, \cdots k_n\}$ 여기서 $k_i=\{c_1, c_2, \cdots c_{mi}\}$

cube c_i에 대한 kernel 테이블 $CK(c_i)$을 cube c_i를 포함하는 kernel 들의 집합으로 정의하고, kernel k_i에 대한 cube function $CF(k_i)$을 kernel ki에 포함되어 있는 cube 들의 집합으로 정의하며, 전체 kernel K에 대한 CF(K)를 각 kernel k_i에 대한 $CF(k_i)$들의 합으로 정의 할때, CF(K)에 대한 factor를 구하여 해당된 cube 들의 합이 kernel 들의 교집합이 된다.

예를 들어 kernel 들이 다음과 같이 주어졌을때

$K = \{k_1, k_2, k_3\}$
여기서, $k_1 = ab + cde + fg + hi$
$k_2 = ab + cde + fg + fh$
$k_3 = ab + fh + gh$

새로운 표현식 CF(K)를 얻기 위해 각 kernel의 cube 에 대해서 새로운 변수 c_i를 도입함으로써 각 cube는 다음과 같이 대치되고 이에 따른 CF(K)를 얻을 수 있다.

$c_1 = ab$, $c_2 = cde$, $c_3 = fg$, $c_4 = hi$, $c_5 = fh$, $c_6 = gh$
$CK(c_1) = \{k_1, k_2, k_3\}$, $CK(c_2) = \{k_1, k_2\}$
$CF(k_1) = c_1c_2c_3c_4$, $CF(k_2) = c_1c_2c_3c_5$, $CF(k_3) = c_1c_5c_6$
$CF(K) = c_1c_2c_3c_4 + c_1c_2c_3c_5 + c_1c_5c_6$

또한 CF(K)에 대한 kernel과 factor를 각각 K(CF(K)) 과 F(CF(K))로 정의했을때

$K(CF(K)) = \{c_2c_3c_4 + c_2c_3c_5 + c_5c_6,\ c_4 + c_5,\ c_2c_3 + c_6\}$
$F(CF(K)) = \{c_1,\ c_1c_2c_3,\ c_1c_5\}$

로 나타낼수 있으며 CF(K) 의 factor들은 kernel 들의 교집합과 관계있고 CF(K) 의 kernel 들은 교집합에 포함된 각 kernel 의 집합과 관계가 있음을 알 수 있다. 따라서 kernel 들의 교집합 I(K) 은 다음과 같이 쓸 수있으며

$I(K) = \{c_1, c_1 + c_2 + c_3, c_1 + c_5\}$
$= \{ab, ab+cde+fg, ab+fh\}$

실제적으로는

$k_1 \cap k_2 \cap k_2 = ab$
$k_1 \cap k_2 = ab + cde + fg$
$k_2 \cap k_3 = ab + fh$

로 나타남을 알수있다.

결국 구해진 kernel 들의 교집합이 공통 부분 표현식들이며 이들 교집합들 중에서 앞에서 이미 기술한 면적및 지연시간 분석기를 적용하여 가장 cost를 만족시키는 공통 표현식을 선택하고 그 선택된 표현식을 새로운 변수로 대치시키는 일련의 작업을 계속하게 된다.

4. 대수 나눗셈 알고리듬

일반적으로 함수 f 를 k 로 나눌때 f = kh + r 로 표현되며 함수 f에 대한 인수 k의 대수 나눗셈 알고리듬 [2] 으로써 그림 8.과 같다.

```
AL_divide(f, k)
U = 함수 f 에서 k 에 있는 문자로 제한한 함수
V = 함수 f 에서 k 에 없는 문자로 제한한 함수
/*f 의 j 번째 항을 uj와 vj로 표현 */
Vi = {vi ∈V | uj = ki}
h = ∩ Vi
r = f - kh
return (h , r)
```

그림 8. 대수 나눗셈 알고리듬
Fig. 8. The algorithm for algebraic division.

예를 들어 f = ac + ad + bc + bd + e 이고 k = a

+ b로 주어졌을때AL_divide(f , k)는
U = a + a + b + b + 1
V = c + d + c + d + e
$V_1 = v_j \in V | u_j = k_1 = c + d$
$V_2 = v_j \in V | u_j = k_2 = c + d$로 나타나며
h = c + d, r = e 가 된다.

Ⅴ. 실험 및 결과 분석

본 논문에서 제안한 성능 구동형 회로 다단 논리합성기의 유용성을 보이기 위하여 SUN4 SPARC machine 상에서 C 언어로 프로그램을 실현하여 MCNC benchmark 데이타에 대하여 실험한 결과가 다음 Table 1. 에 나타내고 있다.

표 1. Benchmark 회로들에 대한 실험 결과
Table 1. The results of benchmark circuits.

회 로 명		AREA	WIRE No.	LITERAL No.	FUNC. LEVEL	MAX DELAY	CPU(sec)
5xp1	PLA	639	399	294	1	48.7	
	MULTI(a)	420	225	174	4	45.5	47.5
	MULTI(d)	434	246	197	3	38.8	91.2
adr4	PLA	763	475	340	1	60.1	
	MULTI(a)	184	91	77	5	46.7	43.6
	MULTI(d)	193	94	79	5	40.1	46.3
misex1	PLA	261	162	122	1	33.9	
	MULTI(a)	223	128	97	3	31.1	11.3
	MULTI(d)	236	144	115	2	25.2	12.6
rd84	PLA	208	112	90	5	64.3	
	MULTI(a)	137	69	64	6	64.3	9.0
	MULTI(d)	188	96	88	5	64.3	9.7
sqr6	PLA	467	294	221	1	45.3	
	MULTI(a)	397	230	171	3	45.3	29.0
	MULTI(d)	399	233	177	3	39.1	39.9
bw	PLA	775	466	345	1	33.9	
	MULTI(a)	579	333	257	3	34.8	36.2
	MULTI(d)	628	373	264	3	31.1	35.2
rd53	PLA	309	194	140	1	45.3	
	MULTI(a)	161	92	71	3	40.9	42.7
	MULTI(d)	177	101	76	3	40.2	44.5
vg2	PLA	1709	1099	804	1	60.1	
	MULTI(a)	398	228	182	4	59.0	55.0
	MULTI(d)	400	241	199	5	53.8	62.9
root	PLA	781	487	346	1	60.1	
	MULTI(a)	476	276	209	4	63.2	92.6
	MULTI(d)	500	299	230	4	52.3	178.0
f51m	PLA	701	439	321	1	60.1	
	MULTI(a)	410	224	176	3	45.3	58.4
	MULTI(d)	447	216	197	3	40.8	47.5
sqn	PLA	451	286	207	1	48.7	
	MULTI(a)	334	200	151	3	40.8	39.0
	MULTI(d)	345	209	158	2	38.9	48.4

표 1.의 실험 결과는 설계자가 제약 조건을 어떻게 주는가에 따라 그 결과는 크게 달라질 수 있다. 표 1.의 결과에서 PLA는 주어진 함수에 대하여 2단 논리 최소화 과정을 거친 결과로 칩 면적, wire 수, 문자 수, 최대 함수 레벨, 최대 지연 시간, 실행시간을 나타내고 있으며, 최소화 과정에서는 ESPRESSO-II를 이용하였다. MULTI(a)와 MULTI(d)는 각각, 칩 면적과 지연시간에 가중치를 부여하고 회로 다단화 과정을 수행한 결과인데, 그 가중치의 부여 정도에 따라 설계자가 원하는 다단 논리회로를 쉽게 얻을 수 있다. 한편, MULTI(a)와 MULTI(d)의 결과는 MIS [2] 의 "simplify" 명령에 해당하는 논리 최소화 과정을 거치지 않은 결과이며, 셀에 대한 라이브러리 정보는 LSI_Logic 데이타 북 [15] 의 AREA, PIN의 수, 내부 고유 지연시간(intrinsic delay), 단위 입력 캐패시턴스에 대한 지연시간의 증가율(slope), 평균 입력 캐패시턴스(ICP)를 포함하고 있다.

Ⅵ. 결 론

본 논문에서는 주어진 부울회로 표현식으로 부터 설계자의 제약조건에 맞는 다단논리회로 구조의 부울회로로 변환하는 새로운 성능 구동형 다단논리 합성기를 제안하였다.

설계자의 제약조건 즉, 지연시간, 칩 면적, 신호선수, 함수레벨, 문자 수등에 가중치를 부여할 수 있도록 하여 설계자가 원하는 다단 회로의 구조를 구성할 수 있도록 하였다. 이때 실제 셀 라이브러리 정보를 참조함으로써 보다 실제에 가까운 회로 분석을 하였기 때문에, 실제 technology mapper와 결합할 경우 회로의 성능은 크게향상되며, 다양한 설계요구조건에 맞는 회로를 빠른 시간내에 생성할 수 있을 것이다.

參 考 文 獻

[1] A.S.Vincentelli, "Synthesis of LSI circuits", Design systems for VLSI circuits logic synthesis and silicon compilation, Martinus Nijahaff, 1987.

[2] R.Rudell, R.K.Brayton, A.Sangiovanni-Vincentelli and A.Wang, "MIS: A Multi-level Logic Optimization System.", *IEEE Transactions on Computer Aided Design CAD-6*, Nov. 1987.

[3] D.G.Bostick, G.D.Hachtel, R.M. Jacoby, M.R.Lightner, P.H.Moceyunas, C.R.Morrison, and D.Ravenscroft, "The Boulder Optimal Logic Design System", ICCAD 1987.

[4] C.L.Wey, T.Y.Chang, "An Efficient Output Phase Assignment for PLA

Mimization.", *IEEE Transactions on Computer Aided Design CAD-9*, pp. 1-7, Jan. 1990.

[5] T.Sasao, "Input Variable Assignment and Output Phase Optimization of PLA's.", *IEEE Trans. Comput.*, vol. C-33, pp. 879-894, 1984.

[6] J.A.Darringer, et al., "Local synthesis through local transformations", *IBM J. Res. Develop.*, vol. 25, no. 4, pp. 272-280, 1981.

[7] R.Brayton, R.Rudell, A.Sangiovanni-Vincentelli, A.Wang, "Multi-level logic optimization and the rectangular covering problems", ICCAD 1987.

[8] K.A.Bartlett, W.Cohen, A.DeGeus, and G.D.Hachtel, "Synthesis and optimization of multilevel logic under timing constraints", *IEEE Trans. Computer-aided design*, pp. 582-596, Oct. 1986.

[9] R.K.Brayton and C.T.McMullen, "The decomposition and factorization of boolean expressions", *Proc. Int. Symp. on Circuits and systems*, pp. 49-54, 1982.

[10] R.K.Brayton, G.D.Hachtel, C.McMullen, and A.Sangiovanni-Vincentelli, "Logic minimization algorithms for VLSI synthesis.", Boston : Kluwer Academic, 1984.

[11] *H.Mathony, U.G.Baitinger, "CARLOS: An Automated Multilevel Logic Design* System for CMOS Semi-Custom Integrated Circuits." *IEEE Trans. Computer-aided Design*, pp. 346-355, 1988.

[12] K.A.Bartlett, B.K.Brayton, G.D. Hachtel, "Multilevel Logic Minimization using Implicit-Don't Care." *IEEE trans. Computer-aided Design*, pp. 732-740, 1988.

[13] R.Brayton, A.Sangiovanni-Vincent-elli, K.Singh, A.Wang, "Timing Opimization of Combinational Logic.", Proceedings of ICCAD-88, Santa Clara, CA, NOV. 1988.

[14] D.Gregory, K.Bartlett, A.J.deGeus, D. D.Hachtel, "SOCRATES: A System for Automatically Synthesizing and Optimizing Combinational Logic.", Design Automation Conference, Las Vegas, NV. June 1986.

[15] LSILogic "LCB15 Macrocells Macrofunctions", March 1988.

著者紹介

李載興(正會員) 第 28 券 A 編 第 10 號 參照
현재 대전산업대학교 전자계산학과 조교수

鄭正和(正會員) 第 28 券 A 編 第 10 號 參照
현재 한양대학교 전자공학과 교수