論文 88-25-8-17

VLSI의 설계검증을 위한 계층적 회로 추출 알고리듬

(Hierarchical Circuit Extract Algorithm for VLSI Design Verification)

林 載 允*, 林 寅 七*

(Jae Yun Lim and In Chil Lim)

要 約

본 논문에서는 VLSI의 마스크 도면정보로부터 회로도를 효율적으로 추출하기 위한 계층적 회로 추출 프로그램을 완성하였다. 마스크 레이아웃 데이타는 CIF 형태의 요소로 구성하며 이들 각 요소를 quad-tree 형태로 보관함으로서, 회로 탐색시간 및 기억공간을 감소시켰다. 본 알고리듬은 CIF 입력부, quad-tree 생성 부, 트랜지스터 검증부 및 연결 list 생성부 등으로 구별되며, 출력을 그래픽 형태로 표시할수 있게 하였다.

본 회로 추출기는 회로의 계층적 구조를 그대로 유지하면서 트랜지스터의 위치 및 연결도를 중복없이 정확히 검증할 수 있으며, 이로부터 회로도를 추출할 수 있다.

본 시스템의 syntax는 YACC 및 LEX를 이용하여 설계하였으며, 이를 프로그램화 하여 각종 회로에 적용하여 그 결과를 검토 함으로써 본 알고리듬의 유용성을 보였다.

Abstract

A Hierarchical Circuit Extract Algorithm, which efficiently extract circuits from VLSI mask pattern information, is programmed. Quad-tree is used as a datastructure which includes various CIF circuit elements and instances. This system is composed of CIF input routine, Quad-tree making routine, Transistor finding routine and Connection list making routine.

This circuit extractor can extract circuit with hierarchical structure of circuit.

This system is designed using YACC and LEX. By programming this algorithm with C language and adopting to various circuits, the effectiveness of this algorithm is showed.

I. 서 론

최근 반도체 집적 기술의 발달로 칩의 집접도가 증가함에 따라, 인간에 의한 설계가 거의 불가능해짐으로서 각종 CAD 기술이 등장하게 되었으며, 이에 따라 각종 CAD 프로그램이 개발되어 회로 자동 설계에 이용되고 있으며, 자동 설계된 회로중 일부는 설계자가 직접 설계함으로서 회로를 완성하게 된다.

최종 설계된 회로는 그 회로가 기능적으로 정상동작을 하는지의 여부를 검증할 필요가 있으며, 회로

*正會員, 漢陽大學校 電子工學科
(Dept. of Elec. Eng., Hanyang Univ.)

接受日字 : 1988年 2月 23日

의 집적도가 증가함에 따라 인간에 의한 검증이 거의 불가능해 짐으로서 각종 시뮬레이션 프로그램 및 테스트 방법들이 고안되어 회로의 기능 검증에 이용되고 있다.

최근에는 설계된 마스크 도면으로부터 회로를 역으로 추출해 내어, 원회로와 비교하여 그 기능을 검증하기 위한 회로 추출에 대한 각종 알고리듬들이 개발되어 회로 검증에 사용되고 있다.[5],[6]

이러한 회로 추출기의 주요 목적은 설계된 회로의 원 기능을 검사함과 동시에 그 결과를 시뮬레이터 또는 논리 추출등에 대한 입력을 제공하는데 있다.[7]

한편 대규모 회로 설계시 반복적으로 사용되는 회로나, 일정기능을 갖는 회로는 그 기본 논리단위로 분할하여 서로 독립적으로 설계한 후, 필요시 이를 불러 설계하는 계층적 회로 설계가 널리 사용되고 있으며 이는 기능적으로 분할하여 설계하므로 설계가 용이하고 회로 변경시 해당기능 블럭만을 변경함으로서, 전체회로 변경에 따른 시간 및 노력이 감소되므로 일반회로 설계에 널리 사용된다.[1],[2]

회로 추출기는 마스크 도면으로부터 원회로를 추출하여 그 회로의 기능을 검사하는 방법으로서, 종래에는 모든 회로의 계층구조를 일단 BOX 형태로 flatten 시킨후 각 회로 요소에 대해 트랜지스터 및 회로 연결도를 추출해내는 방법으로서 flatten에 따른 시간이 많이 들고, 회로의 모든 요소를 저장하기 위한 과다한 기억용량이 필요하게된다.

한편 회로 요소의 저장방법으로는 일반적으로 linked-list 방법이 많이 사용되고 있으나, 이 방법으로는 회로 요소의 기하학적 위치에 대한 정보가 불충분하므로 회로 요소의 삽입, 삭제 시 불필요한 영역도 검사해야하므로 회로의 크기가 증가함에 따라 이에 따른 회로 요소탐색 시간이 기하급수적으로 증가하게된다.

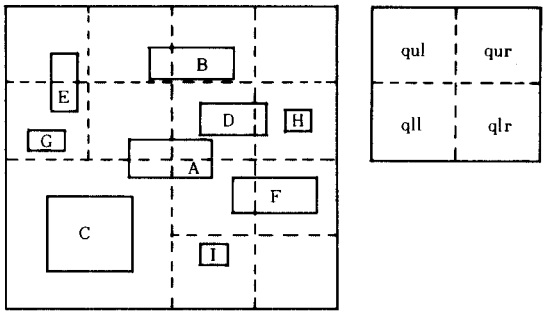
최근에는 회로의 기하학적 크기에 따라 일정영역별로 회로 요소를 분리 저장함으로서 회로 탐색시간을 대폭적으로 줄일 수 있는 quad-tree가 고안되어 VLSI 설계에 널리 사용되고 있다.[4]

이에 따라 본 논문에서는 탐색시간 및 기억공간의 감소를 위해 VLSI회로의 마스크 데이타의 한 형태인 CIF의 각 회로 요소를 quad-tree 형태의 데이타 구조로 보관한 후, 회로의 계층구조를 유지하면서 회로도를 추출하는 계층적 회로 추출 알고리듬을 제안한다.

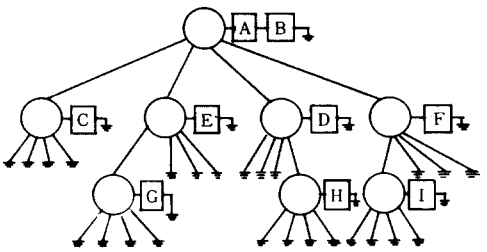
우선 CIF 형태의 데이타를 quad-tree 형태로 보관한 후, 계층 구조의 각 블럭별로 트랜지스터의 위치 및 종류를 발견하고, 트랜지스터의 배치된 형태를 기초로 각 등전위점을 추출하여 신호선 리스트를 생성한 후 원 논리 회로를 추출하여 논리검증을 수행한다. 이렇게 추출된 결과는 그래픽 터미날상에서 schematic 형태로 확인할수 있다.

Ⅱ. 회로 요소에 대한 데이타 베이스

본 회로 추출기의 데이타 구조로서는 계층적 구조의 데이타를 보다 효율적으로 처리하고, VLSI 각종 회로의 기하학적 정보를 보다 정확하고 규칙적으로 구성하기 용이하며 요소의 삽입, 삭제, 확인 등에 따른 탐색등이 효율적인 quad-tree 구조를 채택하였으며, 하나의 quad-tree 내에서는 형태별로 linked-list 구조를 갖도록 구성하였다. quad-tree는 kedem[4]이 제안한 이래 각종 회로설계에 많이 사용되고 있으며, 요소의 크기에 따라 해당 영역에 데이타를 저장하는 형태로 초기의 사각영역을 설정하고 이를 4등분하여 적정 위치에 요소를 위치시키는 형태이다. 그림1(a)는 임의의 요소들을 나타낸것이고 이를 quad-tree로 구성한 것이 그림1(b)이다. 여기서 A, B, C… H, I는 해당 quad-tree에 속한 요소들을 나타낸것이



(a) Object 예



(b) (a)에 대한 Quad-tree구조

그림 1. Quad-tree 및 데이타 구조
Fig. 1. Quad-tree and data structure.

고 한 quad 내의 4 sub-quad 들은 서로 독립적이며, 이 요소들과의 연관 관계를 갖는 요소는 오직 상위 quad 들에 속한 요소들 뿐이다.

한편 quad-tree는 모든 요소를 quad의 위치에 정확히 위치 시키는 full quad-tree와 일정수 이상의 데이타를 한 quad 내에 위치 시키는 adaptive quad-tree로 크게 나눈다. full quad-tree는 요소 저장에 따른 기억용량의 낭비가 크므로 본 시스템에서는 adaptive quad-tree 구조를 선택하였으며, 적정치로서 10개의 요소를 최소치로 잡았으며 회로의 규모에 따라 이를 임의로 조절할 수 있게 하였다.

1. Quad-tree 구성

CIF 형태는 계층적 데이타구조에 적합한 형태를 갖고 있다. 우선 적은 단위의 블럭형태의 데이타를 구성한 후 이를 적절한 이름으로 명명한 후 상위 블럭에서는 이의 이름에 해당하는 블럭을 적정 위치에 복사 함으로써, 계층적 구조를 갖는 데이타를 구성할 수 있다.

본 논문에서는 BOX 및 블럭을 하나의 데이타 요소로 간주하여 이를 quad-tree 형태로 보관한다. quad-tree 내의 데이타는 제한 갯수 이하 이면 해당 quad 내에 종류별로 linked-list 형태로 보관하고, 그 이상이면 그 quad 내의 해당 quad 영역에 recursive 하게 보관시킨다.

일반적으로 회로 구성요소는 크게 BOX, 다각형, 블럭 등으로 구성되며 그 형태는 다음과 같다.

```
B   width  height  xpoint  ypoint ;

P   xp1, yp1, xp2, yp2, ···xpn, ypn ;

C   blocknum  T  xpoint, ypoint ;
```

여기서 BOX의 크기는 xpoint-width/2, ypoint-height/2, ypoint+width/2, ypoint+height/2의 값으로 계산되어 보관되며, 다각형은 BOX 형태로 분할된후 보관시킨다. 또 블럭은 원 블럭의 크기를 계산하여 이를 현 블럭의 해당 quad 영역 내에 보관 시킨다. 여기서 사각형의 구조를 갖는 BOX 형태의 회전 및 확대, 축소등에 대한 '데이타들은 이를 새로 계산한 후 새로운 BOX 형태로 보관하게 함으로서 블럭간에는 복사만이 허용하도록 구성하였다.

2. CIF 블럭의 구성

계층구조를 형성하기 위하여 일정 회로 요소를 그 기능별로 블럭화시키며 하나의 블럭은 BOX, 다각형, 타블럭의 복사등으로 이루어진다. 이러한 블럭의 구성형태는 다음과 같다.

```
DS   blocknum scales  Dx, Dy ;
     :
     BOXes ;
     다각형들 ;
     타 블럭의 복사들 ;
     :
DF ;
```

여기서 blocknum은 현 블럭의 고유 번호로서 후에 상위 계층의 회로에서 복사될 고유번호이며, scales는 설계 기술에 따른 변환 크기로서 일반적으로 a/b 형태를 갖는다. 한편 Dx, Dy는 본 논문에서 설정하여준 현 블럭의 기준점으로서 만일 이 값을 사용하지 않으면 기준점은(0,0)이 된다. 이 점은 후에 복사될때 새로운 이동점으로서 참조되게 된다. 우선 해당 블럭의 내부 회로 구성요소들의 각각의 위치 및 크기에 의해 전체 블럭의 초기 크기를 설정한후, 이를 이 블럭의 초기 quad 크기로 설정한후, 각 회로 요소들을 해당 quad 위치에 할당하게 된다. 다각형은 BOX 형태로 변환하여 보관하고, 복사할 타 블럭들은 그블럭을 찾아 현 블럭내에서의 그 블럭의 상대적 위치 및 크기를 산출하여, 적정 quad를 발견한후 그 위치에 복사할 블럭의 포인터만을 보관한후 후에 이 블럭을 참조 함으로서 flattern에 의한 시간 및 기억 용량을 감소시킬 수 있다.

이러한 블럭들은 블럭이름의 순서대로 binary tree로 구성함으로서 블럭의 증가에 따른 블럭의 탐색을 빠르게 할수 있다.

한편, 타 블럭의 복사 형태는

```
C   cnum  T  Cx, Cy ;
```

의 형태를 갖으며 cnum은 복사할 블럭의 이름이며,

```
예) DS   1   2 / 1  5, 2 ;
       L   CP;
       B   10   4  5, 2 ;
    DF;
    DS   2   2 / 1  10, 10;
       L   CND;
       B   4   14  22, 13;
       C   1   T   22, 13;
    DF;
```

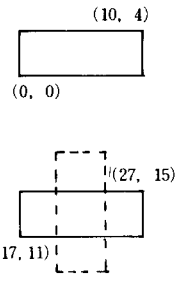


그림 2. 복사된 블럭의 상대위치
Fig. 2. Relative position of copied block.

Cx, Cy는 현 블럭에 복사시킬 위치로서, 블럭정의시 기준점이 이 위치에 오게되며, 만일 복사할 블럭의 원 기준점을 Px, Py라 하고, 복사할 블럭내의 임의의 한점을 Ix, Iy라 할때 이 점이 현 블럭 내에서의 상대적 위치 Nx, Ny는 다음과 같이 계산된다.

$$N_x = I_x + C_x - P_x \quad (1)$$

$$N_y = I_y + C_y - P_y \quad (2)$$

〔Quad-tree 구성 알고리듬〕

회로 요소를 Quad-tree에 저장하는 함수는 다음과 같으며 recursive 한 형태를 갖는다.

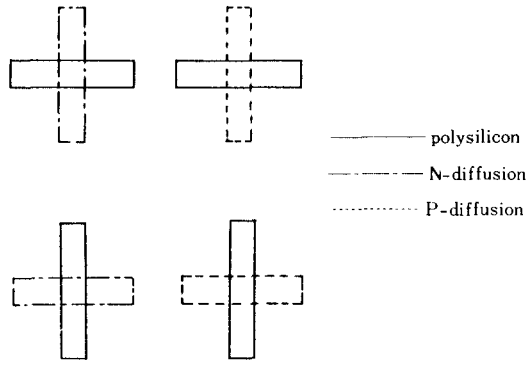
```
QUAD_tree(Q, Obj)
QUAD *Q ;
OBJECT *Obj ;
{
 if(!Q)
   return ;
 if(Q→qnum<MINQ)
 { 이 QUAD 내에 linked list 형태로 Obj 보관 ;
   return ;
 }
 else if(만일 Obj가 이 QUAD의 Q→qll 영역에 속하면)
 { 만일 Q→qll가 NULL 이면
     →새 QUAD 생성후 Q→qll에 할당 ;
   QUAD_tree(Q→qll, Obj) ;
   return ;
 }
 else if(만일 Obj가 이 QUAD의 Q→qul 영역에 속하면)
 { 만일 Q→qul가 NULL 이면
   →새 QUAD 생성후 Q→qul에 할당 ;
   QUAD_tree(Q→qul, Obj) ;
   return ;
 }
 else if(만일 Obj가 이 QUAD의 Q→qur 영역에 속하면)
 { 만일 Q→qur가 NULL 이면
     →새 QUAD 생성후 Q→qur에 할당 ;
   QUAD_tree(Q→qur, Obj) ;
   return ;
 }
 else if(만일 Obj가 이 QUAD의 Q→qlr 영역에 속하면)
 { 만일 Q→qlr가 NULL 이면
     →새 QUAD 생성후 Q→qlr에 할당 ;
   QUAD_tree(Q→qlr, Obj) ;
   return ;
 }
 else
 { 이 QUAD 내에 linked-list 형태로 Obj 보관 ;
   return ;
 }
}
```

Ⅲ. 계층적 회로 추출 알고리듬

본 시스템은 회로의 마스크 도면으로부터 계층적 구조를 그대로 유지한 채로 논리 회로도를 추출하는 시스템이다. 원 계층구조가 갖는 토폴로지를 변경시키지 않고 알고리듬을 수행하므로 전 회로요소를 저장하는데 필요한 기억용량을 대폭적으로 감소 시킬 수 있으며, 불필요한 부분에 대한 탐색을 수행하지 않으므로 회로 추출에 소요되는 시간도 크게 줄일수 있다. 이러한 회로 추출 알고리듬은 크게 트랜지스터 인식과정 및 등전위 신호선 리스트를 생성하는 과정으로 나눈다.

1. 트랜지스터 인식 알고리듬

트랜지스터는 polysilicon과 diffusion이 교차하는 곳에 생성되며 크게 P형 트랜지스터와 N형 트랜지스터로 구별되며 방향에 따라 다음과 같은 4가지 종류로 나눌수 있다.



(a) N형 트랜지스터 (b) P형 트랜지스터

그림 3. N형 및 P형 트랜지스터
Fig. 3. N type and P type transistor.

이들이 각각 설계규칙을 만족하고, 이들 사이에 contact이 존재하지 않으면 트랜지스터로 인식하여, 발견된 트랜지스터는 종류 및 방향과 함께 binary-tree를 사용하여 X 및 Y 값 순으로 저장한다. 계층적 트랜지스터 검출은 quad내, 또는 quad 들간에서 box

들간의 트랜지스터 검출, box와 복사한 블럭들간의 트랜지스터 검출 및 복사할 블럭들간의 트랜지스터 검출등으로 이루어진다.

1) BOX들간의 트랜지스터 검출

이는 현 블럭내의 각 quad내 및 quad간의 polysilicon box와 dffusion box쌍을 발견한후 이들간의 겹침여부를 검사하여, 겹침이 있을 경우 설계규칙을 만족하고 이사이에 contact 이 발견되지 않으면 트랜지스터 임을 감지하여, diffusion의 종류에 따라 트랜지스터의 종류를 구별하고, polysilicon의 방향에 따라 트랜지스터의 방향을 결정한후, 두 box가 만나는 중심점을 트랜지스터의 위치로하여 블럭내에 binary 형태로 보관한다. 또한 발견된 트랜지스터는 후의 등전위점 추출을 위해 diffusion 영역을 두부분으로 나누어서 각각 세 신호선으로 분리하여 서로 다른 전위점으로 보관한다.

〔BOX간 트랜지스터 발견〕

```
BB_TR_check(Q)
QUAD *Q ;
{ if(!Q)
    return ;

  for(Q내의 서로다른 BOX 쌍에 대해)
    만일 두 BOX가 트랜지스터를 형성하면
      트랜지스터의 종류, 방향별 저장 ;
  for(Q내의 한 BOX인 B에대해)
    INTRA_BB_TR_check(Q,B) ;

  BB_TR_check(Q→qll) ;
  BB_TR_check(Q→qul) ;
  BB_TR_check(Q→qur) ;
  BB_TR_check(Q→qlr) ;
}

INTRA_BB_TR_check(Q,B)
QUAD *Q ;
BOX  *B ;
{ if(!Q 또는 Q의 범위가 B와 겹치지 않으면)
    return ;

  for(현 QUAD 내의 모든 BOX인 BB에 대해)
    만일 B와 BB가 트랜지스터를 형성하면
      트랜지스터의 종류, 방향별 저장 ;

  INTRA_BB_TR_check(Q→qll,B) ;
  INTRA_BB_TR_check(Q→qul,B) ;
  INTRA_BB_TR_check(Q→qur,B) ;
  INTRA_BB_TR_check(Q→qlr,B) ;
}
```

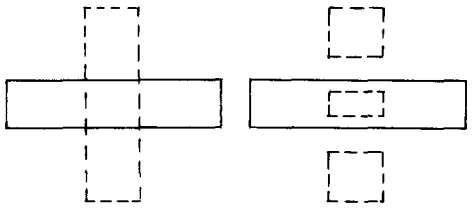


그림 4. 발견된 트랜지스터의 분할
Fig. 4. Partition of selected transistor.

2) BOX와 블럭간 트랜지스터 검출

이는 하나의 BOX와 복사된 블럭들간의 트랜지스터 검출을 수행하는 과정으로서 해당 BOX와 교차관계에 있는 블럭들을 선택하여, 이블럭내의 BOX 또는 복사 블럭과의 트랜지스터 형성여부를 검사하는 과정으로서 한 블럭 내에서 recursive하게 수행된다. 다음은 BOX와 블럭과의 트랜지스터 형성과정을 보인것이다.

```
DS  1  2 / 1 ;
    L  CP;
    B  6  2  3,3  ;
    L  CND;
    B  2  14  3,7  ;
DF;
DS  2  2 / 1 ;
    L  CP;
    B  14  2  11,11;
    C  1  T  8,0 ;
DF;
```

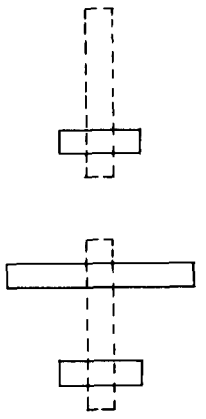


그림 5. BOX와 블럭 사이에서의 트랜지스터 검출
Fig. 5. Transistor find between BOX and BLOCK.

그림(a)는 하나의 기본 블럭을 정의한 것으로 BOX간 트랜지스터 발견 여부 검사에 의해 하나의 트랜지스터가 발견되었으며 그림(b)에서는 복사한 블럭 1과 polysilicon BOX에 의해서 새로운 트랜지스터가 생성된 동시에 블럭 1에 의한 트랜지스터와 함께 2개의 트랜지스터를 발견할 수 있다. 즉 복사할 블럭에 포함된 트랜지스터는 위치만 식(1) 및 (2)에 의해

서 변경시켜 현 블럭내에 포함 시키고 발견된 새로운 트랜지스터를 계속 추가시키면 된다. 만일 새로 발견된 트랜지스터가 기존의 트랜지스터와 중복이되면 그중 하나만을 선택하게 된다. 이에 대한 알고리듬은 다음과 같다.

〔BOX와 BLOCK간 트랜지스터 발견 알고리듬〕

```
BC_TR_check(Q)
QUAD *Q ;
{ if(!Q)
    return ;
  for(현 QUAD의 BOX인 B와 복사할 블럭인 QB에 대해)
  { QB의 환산된 새 기준점 QX,QY 계산 ;
    INTRA_BC_TR_check(B,QB→Q,QX,QY) ;
  }
  BC_TR_check(Q→qll) ;
  BC_TR_check(Q→qul) ;
  BC_TR_check(Q→qur) ;
  BC_TR_check(Q→qlr) ;
}

INTRA_BC_TR_check(B,Q,X,Y)
BOX *B ; QUAD *Q ; POINT X,Y ;
{ if(!Q 또는 Q의 범위가 B와 교차하지 않으면)
    return ;
  for(Q내의 모든 BOX에 대해)
  { X 및 Y에 의해 재조정된 새 BOX와 B와의 트랜지스터검출
    →트랜지스터의 위치, 종류별 저장 ;
  }
  for(Q내의 블럭 BQ의 QUAD인 QA에 대해)
  { 계산된 새 기준점을 XA,YA 산출 ;
    INTRA_BC_TR_check(B,QA ,XA,YA) ;
  }
  INTRA_BC_TR_check(B, Q→qll,X,Y) ;
  INTRA_BC_TR_check(B, Q→qul,X, Y) ;
  INTER_BC_TR_check(B, Q→qur,X,Y) ;
  INTER_BC_TR_check(B, Q→qlr,X,Y) ;
}
INTER_BC_TR_check(B,Q,X,Y)
BOX *B ; QUAD *Q ; POINT X,Y ;
{ if(!Q 또는 Q의 범위와 B가 교차 하지 않으면)
    return ;
  for(Q 내의 모든 BOX에 대해)
  { X 및 Y에 의해 재조정된 BOX와 B와의 트랜지스터 검출
    →트랜지스터의 위치, 종류별 저장 ;
  }
  for(Q내의 블럭 BQ의 QUAD인 QA에 대해)
  { 계산된 새 기준점을 XA,YA 산출 ;
    INTRA_BC_TR_check(B,QA,XA,YA) ;
  }
  INTER_BC_TR_check(B, Q→qll,X,Y) ;
  INTER_BC_TR_check(B, Q→qul,X,Y) ;
  INTER_BC_TR_check(B, Q→qur,X,Y) ;
  INTER_BC_TR_check(B, Q→qlr,X,Y) ;
}
```

3) 블럭 및 블럭간 트랜지스터 검출

이는 현 블럭내의 복사된 두 블럭 사이에 교차 또는 포함관계에 있을 경우 이들 블럭간의 트랜지스터를 발견하는 과정이다. 일반적으로는 블럭간에는 서로 교차관계가 없게 설계함을 원칙으로 하나, 교차관계가 존재하면, 트랜지스터 존재여부를 검사하며, 이는 교차된 해당 블럭들 간에서 다시 BOX와 BOX간, BOX와 블럭간의 트랜지스터 발견 과정을 사용하여 회로 추출한다. 다음은 하위 블럭으로 정의된 두 블럭에 의해 형성된 트랜지스터에 관한 예를 보인것이다.

```
DS  10  2 / 1  20,2 ;
L   CP;
    B  40  4  20,2 ;
DF;
DS  20  2 / 1  20,10;
L   CND;
    B  4  20  8,10;
L   CPD;
    B  4  20  32,10;
DF;
DS  30  2 / 1  20,10;
    C  10  T  20,10;
    C  20  T  20,10;
DF;
E
```

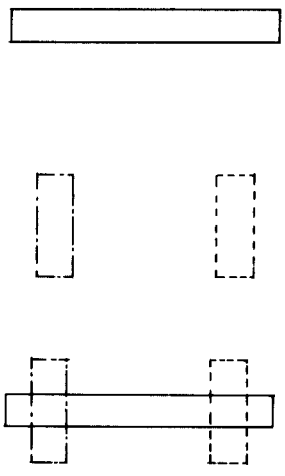


그림 6. 블럭간 트랜지스터 검출
Fig. 6. Transistors between blocks.

〔블럭과 블럭 사이의 트랜지스터 검출 알고리듬〕

```
CC_TR_check(Q)
QUAD *Q ;
{ if(!Q)
    return ;
  for(Q 내의 두 블럭 쌍에 대해)
```

```
{ 두 블럭의 상대 위치 계산 ;
  BOX-BOX, BOX-BLOCK간 트랜지스터 발견 과정
      반복 ;
}
for(Q내의 모든 블럭 QA에 대해)
  하부 QUAD와 QA와의 트랜지스터 발견 과정 수행 ;
CC_TR_check(Q→ql) ;
CC_TR_check(Q→qu) ;
CC_TR_check(Q→qur) ;
CC_TR_check(Q→qlr) ;
}
```

2. 회로 연결도 검증

마스크 도면으로부터 트랜지스터의 위치 및 종류별 탐색을 한 후 트랜지스터를 중심으로 등전위점을 찾아 원 회로도를 추출해내는 과정으로, 신호선 리스트를 생성하여 원 회로도와 비교하여 회로를 검증하는 과정이다. 이 연결도 검증은 계층구조시 하위 레벨의 신호선 삽입과정, 트랜지스터와 기존의 신호선을 기초로하여 새로운 신호선 리스트를 생성해 나가는 과정으로 대별할 수 있으며, 최종 회로도는 그래픽 또는 신호선 리스트로 표시할 수 있다.

1) 신호선 분할 및 신호선 리스트 병합

한 블럭 내에서 트랜지스터의 위치를 발견한 후, 이 블럭내외 복사된 하위 블럭들 중 트랜지스터가 현 블럭내에서 새롭게 정의된 트랜지스터가 존재할 경우, 하위 블럭의 원 신호선 리스트들중 diffusion 영역에 해당하는 신호선을 변형 시킨후, 현 블럭내에 존재하는 신호선과의 연결관계를 고려하여 신호선 리스트로서 삽입 시킨다. 한편 새롭게 트랜지스터를 생성치 않는 하위 블럭들은 해당 신호선들을 기존신호선들과의 연결관계를 고려하여, 신호선 리스트로서 삽입시킨다.

이들 중 하위 블럭 신호선 분할 및 병합 알고리듬을 나타내면 다음과 같다.

```
Block_Wire_insert(Block)
BLOCK *Block ;
{
  for(현 블럭 내의 모든 복사된 블럭들에 대해)
  { 만일 현 블럭내의 회로 요소에 대해 트랜지스터를
    생성하면→복사 블럭의 해당 신호선을 분리하여
            현 블럭의 신호선 리스트로 병합하고 ;
    만일 단지 복사 기능만을 가지면
          →현 블럭의 신호선 리스트로 병합 ;
  }
}
```

2) 신호선 연결 및 분리

신호선은 크게 등전위점을 연결하는 방법과 서로 다른 전위점일경우 분리하여 저장하는 방법이 있다. 신호선 중 동일층 끼리 교차하거나, 다른 층이지만 contact 이나 via 등으로 연결되면 동일 신호선으로 간주하여 동일 그룹으로 둔다. 또한 동일 그룹 중에서 신호선의 연결관계에 의해서, 두 신호선을 한 신호선으로 병합하거나, 다른 방향으로 신호선을 구별하여 저장한다. 그림 7 은 병합 또는 구별 저장의 예를 보인것으로 그림(a)는 동일 층의 신호선이 동일 방향에서 교차하면 하나의 신호선으로 병합하며, (b)와 같이 다른 층이나 contact 이나 via 등으로 연결되고 방향이 동일하면 하나의 신호선으로 병합하나, (c)나 (d)처럼 동일 층으로 교차하지만 방향이 서로 다를 경우, 다른 층으로서 contact 또는 via로 연결되는 경우 동일 신호선 그룹내에 다른 신호선으로 저장한다.

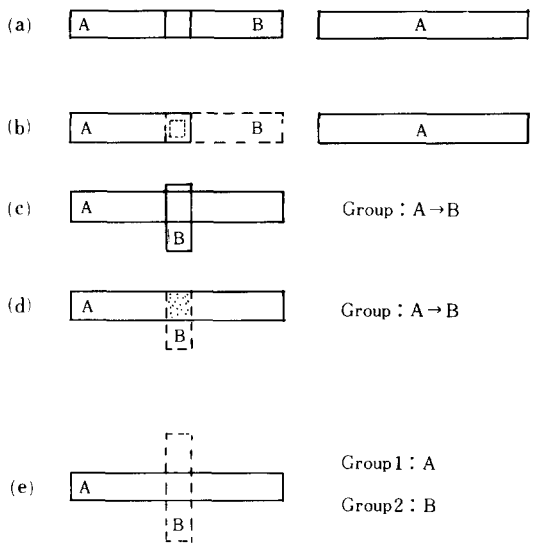
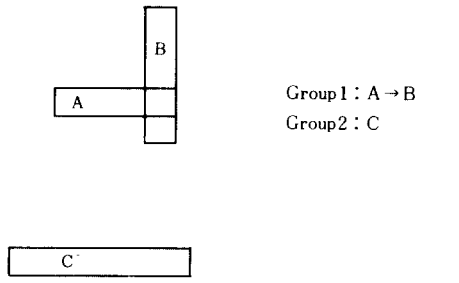


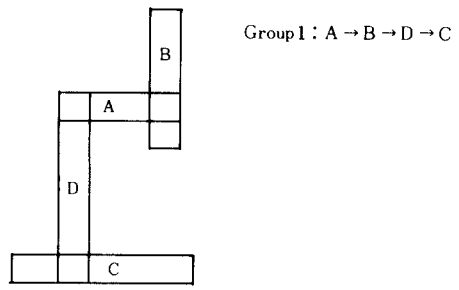
그림 7. 신호선의 병합 및 분리
Fig. 7. Merging and position of signal.

한편 두 신호선이 서로다른 층으로 교차하고 두층 사이에 contact, via가 존재하지 않으면 기존의 신호선과의 교차여부를 검사하여, 동일 층으로서 교차관계에 있는 신호선 그룹을 찾으면 그 그룹내에 저장하고, 발견치 못하면 새로운 그룹을 설정하여 이곳에 저장한다. 또 신호선 연결과정에서 한 신호선에

의해 다른 두 그룹이 병합가능하면, 두 그룹을 하나의 그룹으로 병합하면서, 그룹내에서 병합가능한 신호선은 병합시킨다. 그림 8은 동일 신호선이 서로 다른 그룹에 속해 있을때 새 신호선에 의해 두 그룹을 병합하는 과정을 보인 것이다.



(a) 병합 전



(b) 병합 후

그림 8. 신호선의 그룹 병합
Fig. 8. Group Merging of signals.

이상의 알고리듬을 정리 하면

```
Wire_search(Q)
QUAD *Q ;
{ if(!Q)
    return ;
  for(Q 내의 서로 다른 두 BOX 들에 대해)
  { 두 BOX가 교차할 경우
    { 두신호선 선택여부check ;
     if(두 신호선이 동일층이면)
      동일 신호선으로 간주 신호선 리스트로 삽입 ;
     else
      {if(두 신호선 사이에 contact나 Via가 존재하면)
       동일 신호선으로 간주, 신호선 병합 ;
      else
       다른 신호선으로 간주 독립적으로 신호선 삽입 ;
     }
     신호선 병합가능시 병합 ;
   }
  }

  for(Q의 BOX와 하부 QUAD에 대해)
   recursive하게 동일한 방법으로 신호선 검사, 병합 ;

  Wire_search(Q→qll) ;
  Wire_search(Q→qul) ;
  Wire_search(Q→qur) ;
  Wire_search(Q→qlr) ;
}
```

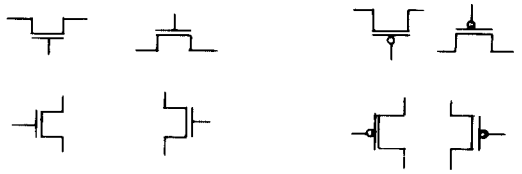
Ⅳ. 회로 추출기의 출력 형태

본 회로 추출기의 출력형태는 트랜지스터의 위치 및 종류별 리스트 및 신호선에 대한 그룹별 리스트 형태로 나타낼 수도 있고, 그래픽터미날 상에서 schematic 형태로 표시할 수도 있다. 트랜지스터 및 신호선 리스트는 시뮬레이션등 다른 목적으로도 사용될 수 있게 출력하였다.

Schematic으로 표시할 경우 트랜지스터는 P형 및 N형으로 나누고 위치에 따라 그림 9와 같이 각각 4종류로 출력되며, 신호선 리스트는 그림 10과 같이 선으로서 표시하고 동일 신호선이 교차할 경우 점으로서 연결한다.

한편 회로 추출 결과를 Net list 형태로 출력시킬 수 있으며 트랜지스터의 경우 트랜지스터의 종류 및 이와 연결된 신호선들의 연결상태로서 표시된다.

이에 대한 예로서 그림11(a)는 inverter 회로에 대한 마스크도면을 나타낸 것이고, 이에 대한 도면 정보로서 CIF 형태를 사용하였으며 이를 b에 나타내었으며, 이를 입력으로 회로추출 알고리즘을 수행한 후 그 결과를 Netlist로 표시한 것이 그림11(c) 이며 이는 HILO-3등의 논리 시뮬레이션의 입력으로 사용



(a) N형 트랜지스터 (b) P형 트랜지스터

그림 9. 트랜지스터의 schematic 출력 형태
Fig. 9. Schematic output of transistor.

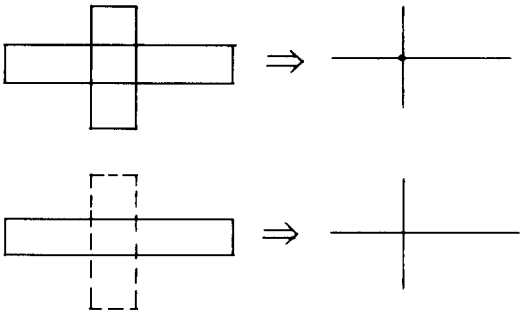
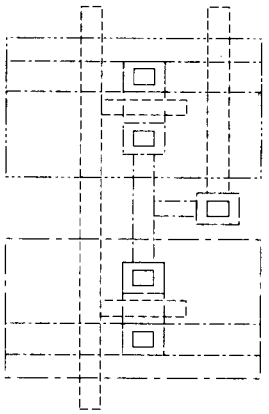


그림10. 신호선의 schematic출력 형태
Fig. 10. Schematic output of signal.

할 수 있다. 한편 이에 대한 schematic을 나타내면 그림11(d)와 같으며, 이를 그래픽 터미날 상에서 확인할 수 있다.

V. 본 시스템의 구성도

본 시스템은 VAX 11/780 UNIX 4.3 BSD 상에서 자동 구문 생성, 분석기인 YACC 및 LEX를 사용하여 구문을 구성하였으며, 필요한 각종 함수들은 C언어를 사용하였고 전체 프로그램 길이는 약 8000 라인 정도이다. 또한 그래픽 터미날로서는 Tektronix 4111 및 4207을 이용하였으며 Mouse 및 Tablet 등을 보조로 사용할 수 있게 하였다.



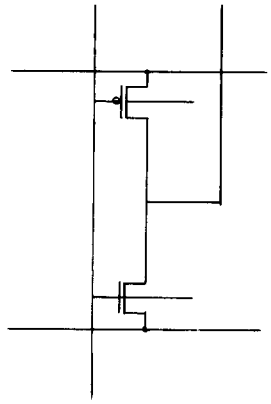
(a) inverter 예

```
DS     5032   2  /    1    26    86;
L      CND;       B    8    24    26,   26;
L      CPD;       B    8    24    26,   78;
L      CP;        B   16     4    26,   26;
                  B   16     4    26,   78;
                  B    4   104    16,   52;
                  B    4    48    40,   80;
L      CM;        B   48     8    24,   18;
                  B   48     8    24,   86;
                  B    4    28    26,   52;
                  B    8     4    32,   52;
L      CNW;       B   48    36    24,   78;
L      CPW;       B   48    36    24,   26;
L      CC;        B    8     8    26,   34;
                  B    8     8    26,   18;
                  B    8     8    26,   86;
                  B    8     8    36,   70;
DF;
```

(b) (a)에 대한 CIF 도면 정보

```
PMOS   26   26   VDD   W1    W2;
NMOS   26   78   W2    W1    VSS;
W1     16    0   16    104;
W1     16   26   34     26;
W1     16   78   34     78;
W2     26   38   26     66;
W2     26   52   40     52;
W2     40   52   40    104;
VSS     0   18   48     18;
VDD     0   86   48     86;
```

(c) 추출된 net list



(d) schematic 출력 결과

그림11. 마스크도면정보로 부터의 net list 및 schematic 출력
Fig. 11. Net list and schematic output from Mask layout.

본 회로 추출기의 입력 형태로는 CIF 및 Auto CAD 상의 DXF형태의 데이타를 이용할 수 있으며 DXF 형태는 자동적으로 CIF 형태로 변환된 후 입력으로 사용된다. 내부 데이타베이스로는 Quad-tree형태를 취할 수 있게 하였으며, 회로추출기의 출력 형태는 원 마스크 패턴에 대한 데이타는 CIF 또는 DXF 형태로 출력 할 수 있게 하였고, 각종 트랜지스터는 위치 및 종류별로 화일 형태로 출력할 수 있으며, 등 전위점에 대한 신호선은 CIF의 Wire 형태로 출력할 수 있게 하였다. 또한 그 결과를 그래픽 터미날 상에서 도형 및 신호선으로 표시할 수 있게 하여 검증이 용이하게 하였다. 이에 대한 개략도를 나타내면 그림12와 같다.

부록에서는 본연구실에서 개발한 Zipper CMOS PLA 회로를, 부록A와 같이 마스크 도면으로 작성한 후, 이에 대한 도면 정보로서 부록B와 같이 CIF 형태로 작성한 후, 이를 본 회로 추출기의 입력으로 사용하여 수행시킨 결과를 부록C와 같이 Schematic 형태로 출력하였다. 여기서 하위 블럭에 대한 CIF 형태는 생략하였으며, 이 회로 추출에 소요된 수행시간은 약 7초가 걸렸다.

Ⅵ. 결 론

VLSI 회로의 설계검증을 위한 계층적 회로 추출 알고리듬을 제안하였다. 원 VLSI 회로가 갖는 계층적 구조를 그대로 이용함으로서 종래의 flatten화시킨 방법에 따른 회로 기억용량 및 트랜지스터 탐색시간 및 동전위 신호선 탐색시간을 감소시키며 보다 정확한 설계 검증을 수행할 수 있게 하였다.

또한 각종 회로 요소들에 대한 데이타 베이스로서는 Quad-tree 구조를 갖게 함으로서 탐색시간의 단축 및 불필요한 요소탐색을 배제할 수 있고, 회로요소의 변경에 따른 데이타베이스 구축이 용이하게 구성하였다. 회로추출 결과를 Tektronix 그래픽 터미날 상에서 그래프로 나타낼 수 있게 하였고, 그 결과를 net list 형태의 file로 나타낼 수 있게하여 차후에 시뮬레이션 및 상위층의 회로 설계에 입력으로 줄 수 있게 하였다.

본 알고리듬은 VAX 11/780 UNIX 4.3BSD 상에서 YACC 및 LEX를 사용하여 구문을 구성하였고, 각종 함수는 C언어를 사용하여 프로그램 하였다. 또 회로추출 결과는 각종 CAD 시스템과의 호환성을 고려하여 구성하였다.

앞으로의 연구 과제로는 VLSI 회로에 맞는 각종 파라메타를 자동적으로 산출하여 이를 회로 및 논리 시뮬레이션에 자동입력시켜 회로에 대한 정확한 시뮬레이션을 수행할 수 있는 시스템으로의 확장이 요구되며 현재 이에 대한 연구가 진행중이다.

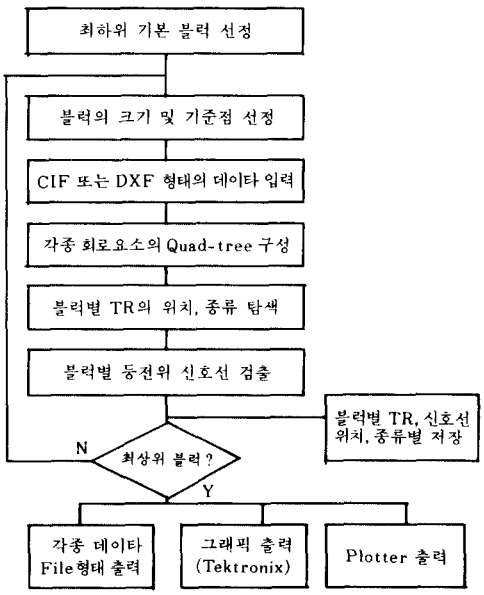
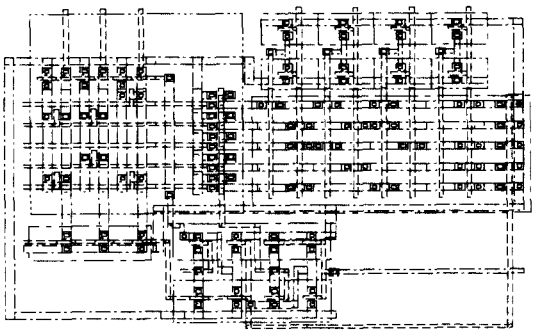


그림12. 본 회로 추출기의 구성도
Fig. 12. Configuration of circuit extractor.

부록A. Zipper CMOS PLA의 마스크 패턴.

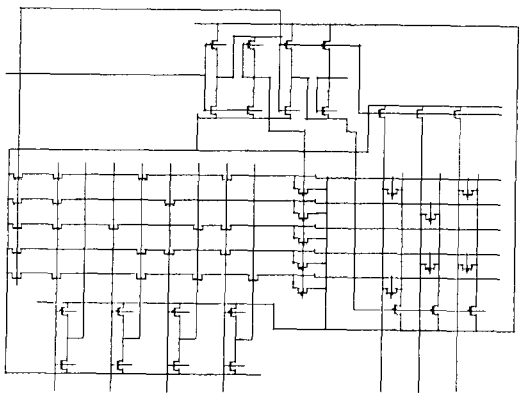


부록B. 부록A에 대한 CIF 형태.

```
DS    6667    2 / 1   0,0;
L     CM;
      B       160     4       88,     314;
      B       8       40      172,    256;
      B       144     8       368,    364;
      B       44      8       30,     24;
```

```
   B      8    100     12,    70;
   B     20      8    234,    92;
   B      8     32    240,    80;
   B     92      8    282,    68;
   B     18      8     41,   120;
   B     18      8     41,   144;
   B     18      8     41,   168;
   B     18      8     41,   192;
   B     18      8     41,   216;
   B      8    128     12,   172;
   B      8    152    284,   140;
   B      8    128    308,   156;
   B    314      8    165,   240;
   B      8     54    318,   263;
   B      8    120    332,   160;
   B      8    120    348,   160;
   B      8    120    364,   160;
   B      8    120    380,   160;
   B      8    120    396,   160;
   B      8    120    412,   160;
   B      8     52    332,   242;
   B      8     52    364,   242;
   B      8     52    396,   242;
   B    112      8    376,   280;
   B      8     24    332,    88;
   B      8     12    348,    70;
   B      8     24    364,    88;
   B      8     12    380,    70;
   B      8     24    396,    88;
   B    128      8    384,    68;
   B      8    304    444,   216;
   B      8     12    412,    70;
L  CPW;
   B    236    136    120,   178;
   B    104     40    376,   280;
L  CNW;
   B    180    232    336,   130;
L  CP;
   B      4    272     20,   244;
   B    174      4    105,   378;
   B      4    128     54,   164;
   B      4    128     78,   164;
   B      4    128    102,   164;
   B      4    128    126,   164;
   B      4    128    150,   164;
   B      4    128    174,   164;
   B      4    128    198,   164;
   B      4    128    222,   164;
   B      4    128    264,   185;
   B     20      4    322,    88;
   B      4     34    308,   243;
   B    112      4    376,   278;
   B    112      4    376,   118;
   B    112      4    376,   142;
   B    112      4    376,   166;
   B    112      4    376,   190;
   B    112      4    376,   214;
   B      4     68    332,    42;
   B      4     68    364,    42;
   B      4     68    396,    42;
   B     96      4    368,    88;
   C   5100      T    296,   272;
   C   5032      T     64,    92;
   C   5032      T    112,    92;
   C   5032      T    160,    92;
   C   5032      T    208,    92;
   C   5100      T     20,   120;
   C   5100      T     20,   144;
   C   5100      T     20,   168;
   C   5100      T     20,   192;
   C   5100      T     20,   216;
   C   5120      T    272,   120;
   C   5120      T    272,   144;
   C   5120      T    272,   168;
   C   5120      T    272,   192;
   C   5120      T    272,   216;
   C   5000      T    308,    88;
   C   5000      T    308,   224;
   C   5113      T     54,   120;
   C   5112      T     54,   144;
   C   5113      T     54,   168;
   C   5113      T     54,   192;
   C   5113      T     54,   216;
   C   5114      T    102,   120;
   C   5114      T    102,   144;
   C   5113      T    102,   168;
   C   5112      T    102,   192;
   C   5114      T    102,   216;
   C   5114      T    150,   120;
   C   5113      T    150,   144;
   C   5114      T    150,   168;
   C   5113      T    150,   192;
   C   5112      T    150,   216;
   C   5114      T    198,   120;
   C   5113      T    198,   144;
   C   5113      T    198,   168;
   C   5112      T    198,   192;
   C   5113      T    198,   216;
   C   5012      T    332,   278;
   C   5012      T    364,   278;
   C   5012      T    396,   278;
   C   5111      T    340,   204;
   C   5111      T    372,   132;
   C   5111      T    372,   180;
   C   5111      T    404,   132;
   C   5111      T    404,   180;
   C   5000      T    332,    80;
   C   5013      T    348,    88;
   C   5000      T    364,    80;
   C   5013      T    380,    88;
   C   5000      T    396,    80;
   C   5013      T    412,    88;
DF;
```

부록C. 부록A에 대한 본 회로 추출기의 회로추출 결과.



參 考 文 獻

[1] N. Weste and K. Eshraghian, "Principles of CMOS VLSI Design: A System Perspective," Addison-Wesley, 1985.

[2] A. Mukjerjee, "Introduction to NMOS & CMOS VLSI Systems Design," Prentice-Hall 1986.

[3] S.P. McCormik, "EXCL: A Circuit Extractor for IC Design," *Proc. 21th DA Conf.* pp. 616-623, 1984.

[4] Gershon Kedem. "The Quad-CIF Tree: A Data Structure for Hierarchical On-line Algorithms," *Proc of 19th DA Conf.* pp. 352-357, 1982.

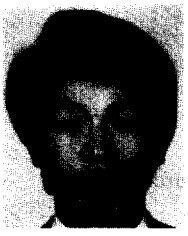
[5] Anoop Gupta, "ACE: A Circuit Extractor," *Proc. of 20th DA conf.* pp. 721-725, 1983.

[6] Marko Chew and Andrzej J Strojwas, "Efficient Circuit Re-extraction For Yield Simulation Applications," *ICCAD 87,* pp. 310-313. 1987.

[7] Ahsan Bootehsaz and Robert A. Cottrell, "A Technology Independent to Hierarchical IC Layout Extraction," *Proc. of 23th DA conf.* pp. 425-431 1986.

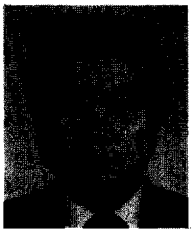
[8] "YACC, LEX", UNIX Programmer's Supplementary Documents, vol. 1 psl: 15-1-psl: 16-13.

著 者 紹 介



林 載 允(正會員)

1957年 8月 15日生. 1981年 2月 한양대학교 전자공학과 졸업. 1983年 2月 한양대학교 대학원 전자공학과 석사학위 취득. 1985年 3月～1988年 2月 한양대학교 대학원 전자공학과 박사과정. 1988年 4月～현재 제주대학교 통신공학과 전임강사. 주관심분야는Silicon Compiler, VLSI Design, AI 기법을 이용한 VLSI 설계 및 Simulation 등임.



林 寅 七(正會員)

1962年 한양대학교 공과대학 졸업 1970年 와세다대학 전자공학과 박사학위 취득. 일본 후지쯔 (주) 정보처리시스템연구소 연구원, 한국과학기술원 대우 교수 University of Illinois at Urbana-Champaign의 Computer Science학과 Visiting Profossor역임. 1964年 한양대학교 강사. 조교수, 부교수. 1977年～현재 한양대학교 교수 재직, 현재 동대학교 전자계산소 소장. IEEE Computer Society Korea Chapter Chairman. 주연구분야는 Computer Architecture, RISC Processor 및 Compiler설계, VLSI Testing/Testable Design, Simulation, Layout, AI기법을 이용한 VLSI 설계 및 Machine-Translation등임.