아날로그 제약 조건을 고려한 집적회로의 레이아웃 자동화

조 현 상[†] · 김 영 수[††] · 오 정 환[†] · 윤 광 섭[†††] · 한 창 호[††††]

요 약

아날로그 집적회로 설계 자동화를 위한 레이아웃 자동화 도구를 제안하였다. 구현된 시스템은 완전 주문형 방식을 채택하고 아날로그 레이아웃의 제약 조건을 고려하였다. 기존의 아날로그 레이아웃 자동화 도구들이 가지고 있는 단점을 보완하기 위하여 변수화된 모듈 라이브러리를 개발, 복잡한 아날로그 모듈들의 레이아웃을 지원하여 확장성을 극대화하였다. 또한 배선 과정에는 기존의 디크스트라 알고리즘을 개선한 동적 다중 경로 알고리즘을 적용하였다. 구현된 아날로그 레이아웃 자동화 도구는 비교기, 연산증폭기 그리고 필터등의 시험회로를 대상으로 시험 수행하였다. 기존의 자동화 도구인 OPASYN과 비교하여 웰합병과 인터디지트형의 모듈로 레이아웃이 수행된 결과를 얻을 수 있었다.

Layout Automation of Integrated Circuits Based on Analog Constraints

Hyun Sang Cho[†] · Young Soo Kim[††] · Jeong Hwan Oh[†] · Kwang Sub Yoon[†††] · Chang Ho Han[††††]

ABSTRACT

A layout automation system for analog integrated circuits is proposed. The implemented system performs full-custom analog layout under the analog layout constraints. In order to overcome the demerits of conventional analog layout systems, parameterized module library is proposed. The system can support complex analog layout modules, resulting in a maximum expandability of the system. Moreover, modified dynamic multi-path algorithm is developed by enhancing the conventional Dijkstra algorithm. Several benchmark circuits such as comparator, op amp, and filter was tested by the system. Layout results compared to OPASYN show well-merging layout and interdigitized layout module.

1. 서 론

집적회로 칩의 집적도가 급속도로 증가함에 따라, 하나의 칩 상에 필요한 시스템을 구현하려는 연구가 활발히 진행되고 있다. 대부분의 혼합신호처리용 단일 칩 시스템의 구조를 살펴보면 입력 및 출력부분은 아날로그 집적회로가 담당하고 나머지 부분은 디지털 집적회로가 담당한다. 시스템 규모가 커짐에 따라서 디지털 집적회로 부분은 디지털 회로 설계 자동화 도구들에 의해 짧은 시간내에 회로 합성 및 레이아웃이 완성될 수 있다[1]. 그러나 아날로그 집적회로 설

※본 논문은 1993-95년도 한국과학재단 연구비 지원에 의한 결과임(과제번호:93-0100-11).

† 정 회 원:LG 정보통신 중앙연구소
†† 정 회 원:삼성전자
††† 정 회 원:인하대학교 전자공학과
†††† 종신회원:인하대학교 전자계산공학과

논문접수:1997년 2월 19일, 심사완료:1997년 7월 24일

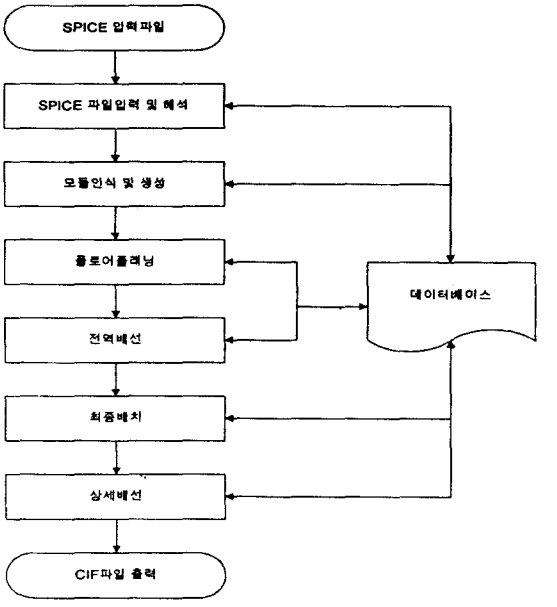
세를 위한 자동화 도구들은 아직 미숙한 단계이기 때문에, 숙련된 설계자의 수작업을 통해 레이아웃을 완성해야 한다. 따라서 아날로그 집적회로의 레이아웃이 혼합신호처리 시스템의 전체 설계시간의 대부분을 소요하게 된다. 이는 아날로그 레이아웃의 자동화 과정이 디지털 회로의 레이아웃과는 달리 소자정합, 신호간섭, 모듈의 비규격성, 기생소자 최소화등의 아날로그 제약 조건들을 고려해서 모듈배치 및 배선이 수행되어야 하기 때문이다.

본 논문에서는 아날로그 제약조건을 고려하는 완전 주문형 방식의 레이아웃 자동화 도구인 INALSYS (INha Analog Layout SYStem)를 개발하고 구현하였다. INALSYS는 스파이스 파일을 입력으로 받아 CIF (Caltech Intermediate Form) 파일 형식의 레이아웃 결과를 출력한다. 입력된 회로는 모듈 인식기에 의해 분할되고 각 모듈의 레이아웃이 모듈 생성기에 의해 생성된다. 높이와 폭에 제한을 받지않는 레이아웃 모듈들은 시뮬레이티드어닐링(simulated annealing)에 의해 배치되고, 최단경로(shortest path) 알고리즘에 의해 배선된다. 배치와 배선 알고리즘 모두 비용을 토대로 최적화하는 알고리즘이므로 이 비용함수를 조정함으로써 아날로그 제약 조건을 고려할 수 있다. 최종 출력파일은 CIF 파일 형식으로 일반 레이아웃 편집기에서도 확인할 수 있다.

본 논문의 구성은 다음과 같다. 2장에서 전체 레이아웃 자동화 시스템의 구조를 제시하고, 3장에서는 레이아웃 자동화 시스템의 각 부분에 적용되는 세부 알고리즘을 제시하였다. 4장에서는 구축된 라이브러리에 관한 실험결과를 제시하였고, 끝으로 5장에서 결론을 기술하였다.

2. INALSYS의 구조

전체적인 INALSYS의 구조는 (그림 1)과 같이 스파이스파일 해석, 모듈 인식 및 생성, 플로어 플래닝, 전역배선, 최종배치 그리고 상세배선 단계로 나눌수 있다. 스파이스 넷리스트 파일이 입력되면 해석단계에서 모듈을 분석하고, 생성된 모듈들의 개개 정보와 상대 위치를 결정한 다음 배선기로 정보를 넘겨주게 된다. 출력 형식은 CIF 파일 형식과 함께 모듈의 정보를 클래스로 구축하여 배선기와 공유하게 된다. 전역배선 과정에서 단자들간의 배선 경로가 결정되고 최종 배치를 거쳐 각 모듈들의 절대적인 위치가 결정되게 된다. 마지막으로 상세배선 단계를 거치면 메탈라인을 포함한 배선 과정까지 모든 과정이 완결된다.



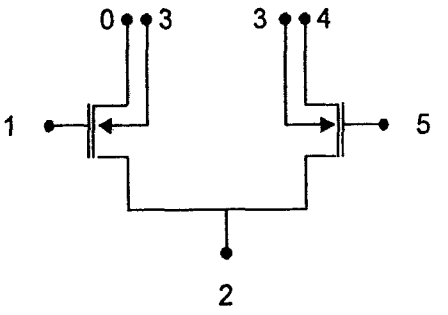
(그림 1) INALSYS의 구조
(Fig. 1) The overall structure of INALSYS

3. 각 단계별 알고리즘

3.1 모듈인식과 생성

디지털 레이아웃 자동화에서의 분할과 대응되는 모듈인식 단계에서는, 입력형식기를 통해 개개의 소자뿐만 아니라 아날로그 고유의 매칭 제약조건을 고려하여 레이아웃되어야 하는 전류미러, 차동증폭기 등의 기본적인 블록들을 인식한다. 모듈생성기가 생성할 모듈들을 인식하기 위하여 입력형식기는 외부 데이터 베이스로 저장된 모듈인식기준을 참조한다. 모듈인식의 기준은 (그림 2)와 같은 스파이스형식으로 기술된 모듈의 구조정보이다.

(그림 2)에서 각 노드의 값들은 상대적 위치들을 표시하고 있으며 −1은 'don't care' 코드이다. 각 모듈의 이름 앞에 위치한 숫자는 우선 순위를 나타내고 있다. 우선 순위에 따라 모듈 들이 인식되므로 하나




```
2 : N-Type Differential Pair( 0:0, 0:1, 0:2, 0:3,
1:0, 1:1 )
{
M0 0 1 2 3 NCH L=-1 W=-1
M1 4 5 2 3 NCH L=-1 W=-1
}
```

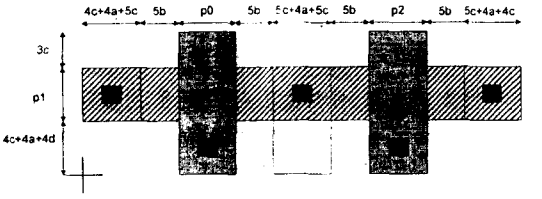
(그림 2) 모듈의 인식 기준
(Fig. 2) Module recognition criteria

의 소자가 여러개의 모듈로 분류되는 일이 없게 된다. 모듈의 이름 밖의 숫자들은 외부 포트와의 연결 관계를 나타내는 정보이다.

모듈생성 단계는 모듈인식 과정을 통하여 인식된 각각의 모듈들에 대하여 블록 레이아웃을 수행하는 단계이다. 현재까지의 자동화 도구들은 모듈생성을 위해 두가지의 방식을 취하고 있다. 하나는 전문설계자가 행한 레이아웃을 미리 정해진 모듈 라이브러리로 저장하여 레이아웃을 생성하는 모듈생성기이고[2][3][4][5][8], 또 다른 하나는 복잡한 모듈 라이브러리를 사용하지 않고 최소로 정의된 초기 모듈 라이브러리를 사용하여 배치 최적화 과정에서 복잡한 구조를 가지는 모듈들을 생성하는 도구이다[6][7]. 전자의 경우는 각각의 레이아웃 모듈을 생성하는 과정을 기술한 생성기 텍스트를 사용하고 있다. 이 방식은 방대한 규모의 모듈 라이브러리를 필요로 하기 때문에, 다른 공정으로의 응용은 모듈라이브러리의 오류발생 가능성, 유지 및 보수 비용등이 증가하게 된다. 후자의 경우, 플로어 플래닝 과정에서 공유비용함수를 조절함으로써 모듈을 생성해 내는 방식을 사용하고 있다. 그러나 플로어 플래닝 단계에 과도한 최적화 과정이 요구되기 때문에, 상당한 수행시간을 필요로하는 단점이 생긴다. 또한 몇몇 모듈들은 공유 최적화과정을 통해 알고리드믹하게 합성하지 못하는 단점도 가지고 있다.

이러한 단점들을 극복하기 위해서 본 연구에서는 공정변수와 모듈 레이아웃의 구조를 변수화하는 새로운 방식의 모듈 라이브러리를 사용하는 아날로그 모듈생성기를 구축하였다. 변수화된 모듈 라이브러리는 확장된 CIF 형식으로 저장하여 복잡한 생성기 텍스트를 대치하는 효과를 가지며, 복잡한 처리기의 구현이 필요하지 않게 된다. 공정 변수와 모듈의 구조를 동시에 변수화함에 따라서 복잡한 구조를 가지는 아날로그 고유의 회로 모듈들도 비용함수를 추가하지 않고 지원할 수 있다.

모듈생성기의 역할은 모듈 라이브러리를 참조하여 인식된 모듈의 내부 레이아웃을 생성하는 것이다. 따라서 설계자의 경험적인 지식과 함께 변수화의 필요성이 요구된다. 모듈 라이브러리는 CIF 기술 언어형식을 사용하였다. (그림 3)은 차동단의 모듈라이브러리 구성을 나타내고 있다. 모듈라이브러리는 프로그램의 일부가 아닌 외부 데이터 파일로 저장하기 때문에 재사용성(reusability)을 극대화할수 있으며 프로그램을 다시 컴파일하지 않고도 수정하거나 추가가 손쉽게 이루어진다. 또한 재사용하기 위해서 공정 변수와 함께 입력회로에 종속되지 않도록 한다.




```
( 2: n-type differential pair );
DS;
L [ndiff];
B [4a * 3 + 4c * 2 + 4f * 4 + p0 + p2] [p1] [L / 2] [W / 2 + 3c];
( size x = [L] );
L [poly];
B [p0] [3c + p1 + 5b + 5c + 4a + 4d] [L / 2 + 4a + 4c + 4f] [W / 2];
B [p2] [W] [L / 2 + 4a * 2 + 4c + 4f * 3 + p0] [y];
( size y = [W] );
L [contact];
B [4a] [4a] [L / 2 + 4c] [3c + p1 / 2];
...
...
L [metal1];
B [4a + 4c + 4f] [p1] [L / 2] [W / 2 + 3c];
...
...
( port: 0 direct: LEFT range: [3c] [L + p1] );
( port: 3 direct: RIGHT range: [L] [W] );
( port: 1 direct: UP range: [4a + 4c + 4f] [L + p0] );
( port: 2 direct: DOWN range: [W] [L + 4a + 4f * 2] );
( port: 4 direct: UP range: [W] [L + p2] );
DF;
```

(그림 3) 모듈라이브러리의 구조
(Fig. 3) Structure of the module library

개선된 CIF 기술 언어형식은 변수를 지원하며, 확장 명령어를 처리할 수 있다. 새로운 모듈을 추가하기 위해서는 입력 형식기의 모듈 인식 기준과 이에 해당하는 모듈 생성기의 라이브러리를 변수화된 형태로 추가하면 된다. 각각의 모듈들은 소자의 크기와 설계 규칙에 따라 변수화되며 대칭과 균형구조를 가지는 아날로그 기본 모듈들을 지원하기 위해서 확장 명령어 형식도 지원한다. 확장 명령어들은 소자의 크기뿐만 아니라 구조들도 변수화해야 할 필요가 있는 멀티핑거(multi-figner) 구조의 MOS소자와 인터디지트형(interdigitized) 소자를 구현하는 데 사용한다. 다른 모듈생성기에서 사용하는 모듈 라이브러리의 생성기 텍스트중 많은 부분이 이러한 소자들을 지원하기 위하여 방대해지기 때문에 확장 명령어 형식을 지원함으로써 모듈 생성기의 구축비용을 최소화하고 유지, 보수 노력을 최소화할 수 있다. 〈표 1〉은 기술언어 형식에 추가한 확장 명령어들을 나타내고 있다.

〈표 1〉 확장 명령어 문법

〈Table 1〉 Format of the extended command set

확 장 명 령 어	기 능
align(a box of a module1, a box of a module 2);	모듈의 정렬
repeat(a module, the number of replications);	반복모듈의 생성
wire(a central point1, a central point 2, width);	추가 배선
delete(a label name of box);	모듈 삭제
#if ... #else if ...	제어 구조

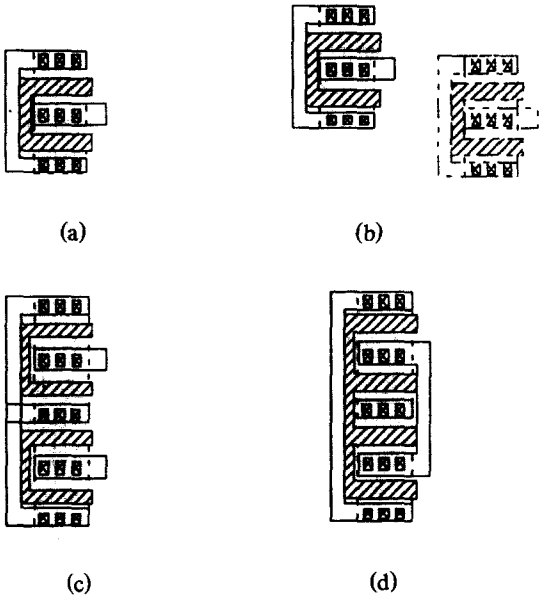
(그림 4)는 확장된 CIF 기술언어 형식을 사용하여 N개의 게이트 폴드를 가지는 멀티 타입 MOS 소자를 기술한 모듈 라이브러리의 예를 보이고 있다. 이와같은 확장명령어를 사용함으로써 모듈 라이브러리 구축시 반복되는 블록을 그 블록을 표현하는 문법으로 대체하여 모듈 라이브러리 내에 자유롭게 생성할 수 있음을 볼 수 있다.

(그림 5)는 (그림 4)에 표시된 확장 모듈라이이브러리에 의해서 모듈이 생성되는 과정을 표시한 것이다. 게이트 폴드가 2인 멀티 타입 MOS 소자가 생성되는 예를 보여주고 있다.

```
( 10: Multi-finger MOS );
#if N == 2
label_name:
DS;
L [ndiff];
B [p1][5c*3+4a*3+5c*3+d*4+p0*2][5c*2+4a*2+5c*2+d+p0+d+p1/2][5c+4a+5c+p0+d+5c+4a/2];
.....
.....
..... layout descriptions of Multi-finger MOS(Gate fold == 2 ) .....
(size x=[5c*2+4a*2+5c*2+d+p0+d+p1+5c+4a+5c]);
(size y=[5c*3+4a*3+5c*3+d*4+p0*2]);
(port: 0 direct: LEFT range: [5c+4a+5c+d+p0+d][5c*2+4a*2+5c*2+d+p0+d]);
(port: 1 direct: RIGHT range: [5c+4a+5c+d+p0+d][5c*2+4a*2+5c*2+d+p0+d]);
(port: 2 direct: UP range: [5c*2+4a*2+5c*2+d+p0+d][5c*2+4a*2+5c*2+d+p0+d+p1]);
DF;
#else if N == 4
repeat(label_name);
align(box_layer_label1, box_layer_label2);
wire(terminal1, terminal2);
#else if N == 6
.....
.....
```

(그림 4) 멀티타입 MOS 소자의 모듈라이브러리

(Fig. 4) Module library for a multi-fingered MOSFET



(그림 5) 멀티 타입 MOS소자의 생성 과정
a)기본 모듈 생성 b)repeat 명령 수행
c)align 명령 수행 d)wire 명령 수행
(Fig. 5) Synthesis process of multi-type MOS device
a)basic module definition b)repeat command performed
c)align command performed
d)additional wire command performed

3.2 가상 콘택 레이아웃 생성

일반적인 모듈생성기들[2][3]은 개개의 모듈들을 생성하는 전용 모듈생성기로 구성되어 있으며 콘택(contact) 생성 루틴도 프로그램내에 코드의 형태로 삽입되어 있다. 따라서 동일한 작업을 수행하는 코드들이 중복되어 있어 비효율적이고 콘택의 생성을 위한 루틴을 수정하기 위해서도 개개의 모듈생성기들을 각각 수정하여야 하는 등 재사용성이 떨어지게 되는 문제점을 가지고 있다. 이러한 문제점을 극복하기 위하여 모듈 생성기로부터 콘택 생성루틴을 제거하고 각각의 모듈라이브러리에 콘택영역만을 표시한 후 가상 콘택 레이아웃 생성기가 통합해서 처리하게 된다.

3.3 플로어플래닝

플로어 플래닝 과정은 슬라이싱 트리(slicing tree)모델에 근거한 시뮬레이티드 어닐링 알고리즘을 사용하여 수행한다. 비용함수에 근거한 최적화 방법은 비용 함수를 통하여 제어를 하기가 쉽다는 것이다. 본 자동화 도구에서 사용하는 비용함수는 전체면적과 소자 종횡비와 개개 소자의 웰(well) 합병을 고려한 페널티(penalty) 항목으로 구성되어 있다. 웰합병의 절차는 일반적인 형태로 기술하기 힘들기 때문에 다른 플로어플래너[4][6]에서는 지원하지 않거나 복잡한 휴리스틱(heuristic) 루틴을 사용하여 수행하고 있다. 하지만 본 레이아웃 자동화 도구에서는 시뮬레이티드어닐링 과정에서 $C_{wellmerge}$를 추가하여 제어함으로써 추가적인 루틴의 사용이 필요하지 않게 된다. (식 1)은 비용함수의 전체적인 구성을 나타낸다.

$$C_{total} = \alpha_{area} C_{area} + \alpha_{aspect} C_{aspect} + \alpha_{wellmerge} C_{wellmerge} \quad (1)$$

α_{area} : 면적 비용함수 가중치
α_{aspect} : 종횡비 비용함수 가중치
$\alpha_{wellmerge}$: 웰 합병 비용함수 가중치

각각의 비용함수들은 (식 2), (식 3), (식 4)의 항으로 구성되어 있다. 또한 (식 5)과 (식 6)은 웰 합병시 사용되는 β_p와 β_n을 결정하는 식이다.

$$C_{area} = (\text{최대 외각 사각형의 면적}) \quad (2)$$

$$C_{aspect} = |w_t - h_t| \quad (3)$$

$$C_{wellmerge} = |w_p/h_p - \beta_p| + |w_n/h_n - \beta_n| \quad (4)$$

w_t : 전체 레이아웃 폭
h_t : 전체 레이아웃 높이
w_p/h_p : p 소자 슬라이싱 트리의 종횡비
w_n/h_n : n 소자 슬라이싱 트리의 종횡비

$$\beta_p = \frac{\Sigma\ P-device\ area}{\Sigma\ P-device\ area + \Sigma\ N-device\ area} \quad (5)$$

$$\beta_n = \frac{\Sigma\ N-device\ area}{\Sigma\ P-device\ area + \Sigma\ N-device\ area} \quad (6)$$

3.4 최단 경로 알고리즘을 이용한 전역 배선

전역배선에는 단방향성인 디크스트라(Dijkstra)[9]의 알고리즘을 무방향성 알고리즘으로 개선하여 구현하였다. 무방향성 다중 경로 알고리즘은 전처리 단계에서 정점간의 정보를 이용해 생성된 그래프를 입력으로 받아 최소 거리 비용을 갖는 다단자 경로를 구현하는

알고리즘이다. 전체적인 다중 경로 알고리즘은 임의의 두 단자를 최단 경로 알고리즘으로 연결한 뒤, 나머지 각 단자에 대해 이미 경유한 모든 정점들과의 최단 경로를 구하여 비교한다. 이중 비용이 가장 적은 경로를 선택하여 이전 경로와 합병시켜 나간다. 이 알고리즘에서는 가능한 모든 경로를 비교하지 않으므로 최소 비용을 갖는 트리를 생성하지 않을 수도 있다. 최소 비용을 갖는 트리를 생성하기 위해서는 처음 두 단자를 선택할 때 모든 가능한 조합에 대해 위와 같은 과정을 반복 수행하여야 한다. 그러나 수행 시간대 성능비를 고려할 때 이알고리즘은 대부분의 아날로그 회로 응용에 만족할 만한 결과를 낳는다.

3.4.1 개선된 디크스트라 알고리즘

기존의 최단 경로를 구하는 디크스트라의 알고리즘은 단방향성의 성질을 가지고 있어, 무방향성에서 나타나는 현상 즉, 마지막 정점으로부터 역행해서 최단

```
        void Shortest_Path( Table T, Vertex V_S, int Maxim )
        /* V_S는 입력되는 Vertex, Maxim은 마지막 정점 */
        {
1:          Vertex V, W;
2:          int i;
3:          if(V_S != Maxim) /* 입력되는 정점이 마지막 정점이 아닌 경우 */
4:          {
5:              for(i=0; i<Maxim+1; i++)
6:              {
7:                  V= Smallest Unknown Distance Vertex;
                    /* 입력되는 정점에서 가장 인접된 정점을 찾는다 */
8:                  if( V == Not_A_Vertex)
9:                      break;
10:                 T[V].known = TRUE;
11:                 for(W=0; W<Maxim+1; W++)
12:                 {
13:                     if(!T[V].known)
14:                     {
15:                         if(T[V].Dist + D(V,W) < T[W].Dist)
16:                         {
                                /* D(V,W)는 정점 V에서 정점 W까지의 최소 */
                                /* 거리 비용을 구하는 함수이다. 반환되는 */
                                /* 값은 거리 비용 D이다. */
                                /* Update W */
                                /* T[W].Dist를 Update시킨다. */
17:                             decrease( T[W].Dist To T[V].Dist + D(V,W));
                                /* T[W].Path를 Update시킨다. */
18:                             T[W].Path = V;
19:                         }
20:                     }
21:                 }
22:             }
23:         }
24:         else /* 입력되는 정점이 마지막 정점일 경우 */
25:         {
                /* 마지막 정점일 경우 역행해서 최단 경로를 구한다. */
26:             for(i=V_S; i>-1; i--)
27:                     Repeat From line 7 To line 10;
28:             for(W=Maxim; W>-1; W--)
29:                     Repeat From line 12 To line 18;
30:         }
31:     }
```

(그림 6) 개선된 최단경로 알고리즘
(Fig. 6) The modified shortest path algorithm

경로를 구하는 현상을 처리할 수 없었다. 이로 인해, 기존의 알고리즘에 입력되는 정점이 마지막 정점인지를 구별할 수 있는 처리 루틴을 첨가해 마지막 정점일 경우 최단 경로 루틴을 역행해서 최단 경로를 구할 수 있도록 개선하여 구현했다. 개선된 알고리즘은 기존의 알고리즘과 비교할 때, 실행 시간 면에서는 기존의 알고리즘의 실행 시간은 $O(|E|\log|V|)$이다. 여기서 E는 간선의 총 개수, V는 정점의 총 개수이고, 개선된 알고리즘의 실행 시간은 입력되는 정점이 마지막 정점인지 구별하는 루틴 실행 시간이 더해져 $O(|E|\log|V|+|V|)$이다. 하지만, $|E|\log|V| \gg |V|$이므로, $|V|$을 무시할 수 있다. 결국, 실행 시간은 $O(|E|\log|V|)$가 되어 기존의 알고리즘과 같은 수준의 실행 시간을 갖는다. 개선된 알고리즘은 (그림 6)와 같다.

3.4.2 최소 거리 비용의 동적 다중 경로 알고리즘 구현

동적 다중경로 알고리즘은 이미 결정된 최단 경로에서 다음의 연결 정점이 입력 되었을 때, 최단경로를 구성하는 각 경로 정점들과의 거리비용을 계산하고 이 거리 비용 중에서 최소거리 비용을 가지는 정점을 검색하여 경로망에 첨가하고, 다음 연결 정점이 입력될 때 첨가된 경로 정점까지 포함한다. 이 알고리즘은 입력된 정점들이 하나의 망을 이룰때 망의 경로는 중복되어도 된다는 회로선의 특성을 적용해 최소 거리 비용 경로를 선택한다. 이 알고리즘은 최소 거리 비용을 가지면서 최적화된 경로망을 지향하지만 거리 비용을 우선적 조건으로 하고 있기 때문에 항상 최적화된 경로망이 선택되는 것은 아니다. 그러므로, 경로추출의 각 단계마다 구해지는 최단 경로는 그때마다 동적으로 저장되어야 한다. 동적 저장을 위한 동적 저장 구조체는 하나의 구조체를 구성하고 3개의 포인터 변수가 이 구조체를 선언한다.

동적 저장 구조체를 형성하면 동적 다중경로 알고리즘을 이용해 경로를 추출할 수 있다. 동적 다중경로 알고리즘은 최소 거리 비용을 우선 조건으로 적용해서 실행된다. 우선, 회로망의 특성을 적용하여 중복되는 경로에 대해서는 중복 경로를 허용하지 않는다. 이로 인해 최적화를 지향하지만, 입력되는 망구성 정점에 의해 항상 최적화 되지는 않는다. 전체적인 알고리즘 구성은 크게 초기 최단 경로를 구하는 단계와

초기 최단 경로에서 구해진 경로 정점들을 가지고, 다음 입력 정점의 최단 경로를 구하는 단계의 두 단계로 나뉘어진다. 각 단계에서 구해진 최단 경로는 그때마다 동적으로 저장 구조체에 저장되어지고, 최단 경로 버퍼에는 중복되지 않게 경로 정점들이 저장된다. 그리고, 하나의 망이 구성되면 최단 경로 버퍼는 초기화되어진다. 동적 다중경로 알고리즘의 의사코드를 (그림 7)에 나타내었다.

```
        void Multi_Routing(struct Vertex *a, struct Edge *p, struct Table *t)
        {
1:          int k,l;
2:          k=망의 첫번째 정점;, l=망의 두번째 정점;
4:          for (망 번호가 NULL일 때까지);
            {
5:              if(망번호==0)
6:              {
7:                      Shortest_Path(k,l);
                        sp_buffer=경로 정점 저장;
                        저장 구조체=SP정점 저장;
                }
8:              else
                {
9:                  k,l에 새로운 정점 입력;
10:                 Shortest_Path(k,l);
                    sp_buffer=경로 정점 저장;
                    저장 구조체=SP정점 저장;
                }
11:             for 망 구성 정점이 NULL일 때까지;
                {
12:                 k=구해진 각 경로 정점;
13:                 l=다음 망 구성 정점;
14:                 line 10을 반복 수행한다;
                    저장 구조체=최소 비용 거리 경로를 검색하여 저장;
                }
15:             sp_buffer free;
16:         }
17:     }
```

(그림 7) 동적 다중경로 알고리즘
(Fig. 7) A dynamic multi-path algorithm

3.5 최종 배치

최종 배치(final placement)는 각 모듈들 간의 배치 정보를 확정하는 과정이다. 전역 배선과정에서 선들의 경로가 결정되면, 각 채널당 배당되는 선들의 수가 결정된다. 결정된 선들의 수에 따라 채널의 폭이 결정된다. 결정된 채널의 폭에 따라 채널 공간을 확보하고, 모듈들을 이동시킨다. 이때, 모듈들의 크기가 불규칙적이므로 하나의 수직선상이나 수평선상의 채널의 위치는 불규칙적이 된다. 이 경우, 상세배선의 복잡도가 증가하여, 배선이 불가능한 경우가 발생할 수 있다. 이에 따라 수평선상이나 수직선상의 채널 영역중 가장 큰 영역을 선택하여, 그 채널영역에 맞추어 채널 공간을 확보한다. 한번 확보된 채널영역은 레이아웃이 완료될 때 까지 유지된다.

3.5.1 채널영역 확보

전역배선에서 전체영역은 모듈영역과 배선영역으로 구분하고 배선영역을 다시 채널영역과 스위치박스 영역으로 나누게 된다. 모듈영역은 모듈생성과 배치과정을 통해서 확정된 모듈들이 위치한 영역이며 채널영역은 모듈과 모듈사이에 위치한 직사각형의 공간으로서 수직 혹은 수평 채널로 나눌수 있다. 이외에 수평 혹은 수직채널간에 배선되는 부정형의 공간을 스위치박스 영역이라 한다. 이 과정에서 선들의 경로가 결정된 결과는 금속선이 지나가는 채널의 번호열로 표시된다. 모든 넷(net)에 대해 검색을 하면 각각의 채널안을 지나가는 금속선의 갯수를 추출할 수 있다. 추출된 금속선의 갯수에서 채널의 넓이를 결정한다. 임의의 채널을 지나는 금속선의 갯수를 num_metal이라 하면, 채널의 넓이는 (식 7)과 같다.

$$\text{채널의 넓이} = num\_metal \times w + (num\_metal + 1) \times d \quad (7)$$

여기서 w, d는 각각 공정에서의 금속선 폭, 금속선과 금속선과의 간격을 나타낸다.

3.5.2 채널영역의 정렬

채널영역을 확보하려면 각각의 모듈들을 이동시켜야한다. 모듈들의 크기는 서로 다르고, 채널이 확보되는 넓이도 다르므로, 하나의 수평선상과 수직선상에의 채널들의 위치는 제각각이 되며, 따라서 채널과 채널, 모듈과 모듈사이에 존재하는 스위치 박스영역

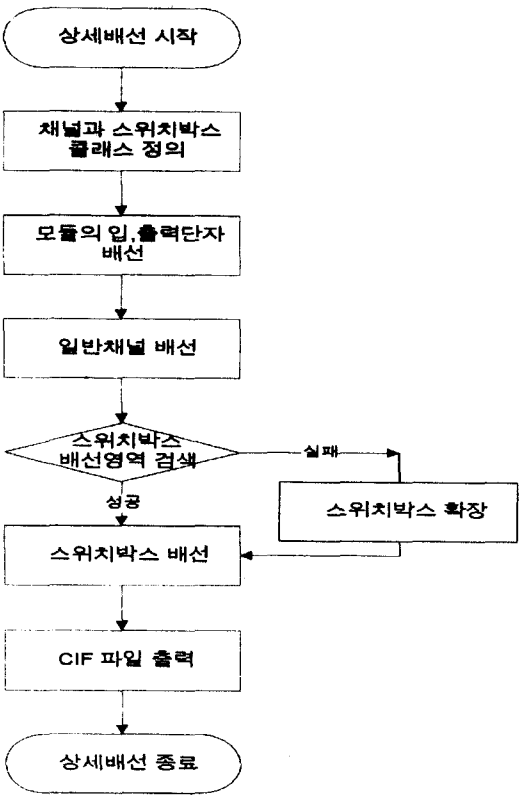
```
int width; // 채널들의 넓이정보 변수
int maximum_width = 0; // 채널의 최대넓이를 저장한다.
for( int count = 0 ; count < 각 선상의 채널의 갯수 ; count++ )
  {
    get_channel_information( width ); // 각 채널의 위치정보를 얻는다.
    if( maximum_width < width ) {
            maximum_width = width;
            }    // 최대 채널넓이를 저장한다.
  }
re_location_channel( maximum_width );
// 채널들의 폭을 재설정하고, 위치정보를 수정한다.
```

(그림 8) 채널영역 정렬 알고리즘
(Fig. 8) Channel region alignment algorithm

이 복잡해지게 된다. 이에 따라, 상세 배선시 배선복잡도가 증가하여 배선이 불가능한 경우가 발생할 수 있으므로 채널의 영역을 통일해 주는것이 필요하다. 하나의 수직선이나 수평선상에 위치한 채널중 가장 넓은 영역을 가진 채널의 넓이에 그 선상의 채널의 넓이를 맞춘다. 이것을 의사코드로 표현하면 (그림 8)과 같다.

3.6 상세 배선

전역배선에서 추출된 정보는 상세배선 단계에서 이용하기 위해 저장된다. 상세배선에서는 추출된 배선경로와 스위치 박스 정보를 이용하여 모듈과 모듈사이를 금속선을 이용하여 연결한다. 상세배선 단계에서 고려되어야 하는 부분은 모듈의 입, 출력 단자에서의 배선과 스위치 박스에서의 배선이다. 모듈의 입, 출력 단자에서의 배선시 단자에 부여되는 우선순위에 따라 배선복잡도가 감소하기도 하고, 증가하기도 한다. 스위치 박스 상세배선은 전역배선에서 추출된 정보를 최대한 활용하되, 필요한 경우 스위치 박스 영역을 확장한다. 최종 레이아웃 결과는 CIF 파일로 출력된다. 제안된 상세배선 알고리즘의 흐름도는 (그림 9)에 제시되어 있다.

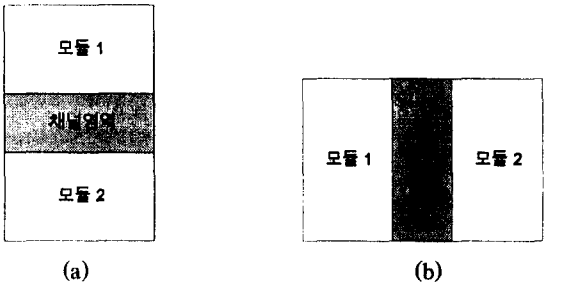


(그림 9) 상세배선의 흐름도
(Fig. 9) Flowchart of the detailed routing

아날로그 회로의 상세배선은 모듈의 크기가 비규칙적이고, 채널의 배치도 비규칙적이므로, 숙련된 설계자가 배선하는 형태로 알고리즘을 작성하는 휴리스틱(heuristic)방법이 흔히 사용된다. 본 논문에서도 채널들의 배치를 정렬하여 처리해야 할 처리루틴을 감소시키고, 레이아웃시 발생할 수 있는 문제들을 예측하여 각 경우에 대한 처리루틴을 작성하는 휴리스틱 방식을 사용하였다.

3.6.1 수직방향 채널과 수평방향 채널안에서의 배선

채널안에서의 배선은 수직 또는 수평의 방향을 갖는 선으로만 배선이 이루어진다. 채널안에서 수평방향과 수직방향의 선들이 동시에 존재하는 경우는 채널안에 배선되어야 할 포트가 있는 경우이다. 본 논문에서는 수평방향의 채널의 경우 메탈 I을, 수직방향의 채널인 경우에는 메탈 II를 이용하여 배선하였다. 수평채널과 수직 채널은 모듈간의 위치에 따라서 결정되며 각각의 예를 (그림 10)에 보였다.



(그림 10) (a)수평채널과 (b)수직채널
(Fig. 10) (a) The horizontal channel and (b) the vertical channel

채널에 있는 포트를 연결하는 금속선은 항상 메탈 I을 사용하였다. 채널안에 배선되는 금속선의 정보는 채널구조체에 저장된다. 채널구조체의 데이터구조는 (그림 11)와 같다.

```
struct channel {
    int *tracks;   // 각 트랙에 배정된 선의 net번호
                   를 저장한다.
    int *tracksituation;  // 각 트랙에 배정된 선이
                          metal 1 인지 metal 2 인
                          지 저장한다.
    int trackcount;  // 트랙의 갯수를 저장한다.
    int left, top, right, bottom;
       // 채널의 좌표정보를 저장한다.
}
```

(그림 11) 채널의 데이터구조
(Fig. 11) Data structure of the channel region

3.6.2 채널간 배선트랙의 설정

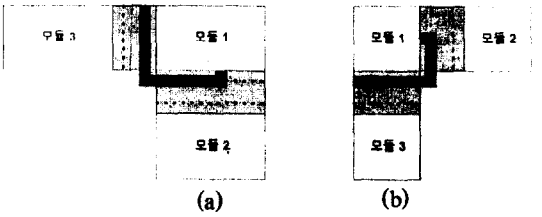
채널과 채널사이를 배선할 때 (식 7)에 의해서 채널내에 배선가능한 경로가 결정되는데 이를 배선트랙이라한다. 배선트랙은 채널의 종류인 수평채널 혹은 수직채널에 따라서 방향이 결정되며 공정변수에 따라서 배선가능한 트랙의 수가 결정된다. 채널안에 어떤 트랙을 배정하느냐에 따라 배선의 복잡도가 좌우된다. 배선의 복잡도가 최소가 되고, 배선이 안정되게 되는 조건은 선의 길이가 최소가 되도록 트랙을 설정하는 것이다. 채널의 방향이 서로 같은 경우에는 채널과 채널사이는 되도록 직선이 되어야 한다. 채널과 채널간에 서로 트랙의 번호가 다르면 선의 배선되는 위치가 서로 다르기 때문에 채널사이의 영역에서 한번이상 선이 꺾어져야 한다. 선이 꺾이는 부분에서는 비아가 들어가기 때문에 배선할 수 있는 공간이 줄어들고, 배선의 안정도도 감소한다. 채널의 방향이 서로 다른 경우에는 되도록 선의 길이가 최소가 되도록 선의 트랙을 할당하도록 한다. (그림 12)은 이러한 알고리즘을 나타낸 것이다.

(그림 13)에는 채널간 배선의 예를 보였다. 채널트랙을 선택하기 위해서는 현재 금속선이 배선되고 있는 채널과 다음에 금속선이 배선될 채널의 위치를 고려해야 한다. (그림 13(a))와 같이 수평방향 채널인 경

```
channel channel1;  // 선이 배선된 채널
channel channel2;  // 연결되어야 할 채널
// 채널의 방향이 서로 같을때
 track = channel1.track;  // channel1 에서 배선된 트랙 저장
 istrackempty = comp( channel2, track );  // channel2에서 같은 트랙이 비어있는지 조사
 if( istrackempty == 1 )  // 같은 트랙이 비어있을때
   {
     input( channel2, track, channel1 );  // channel2의 같은 트랙에 배선정보 저장
   }
 else
   {
     dogleg( channel2, channel1 );  // channel1, 2를 꺾어진 선을 이용하여 연결
   }
 // 채널의 방향이 서로 다를때
 for( ; check != 1 ; track++ )
   {
     track = find( channel1, channel2 );  // channel2에서 channel1과
                                             가장 가까운 트랙 검사
     check = checktrack( channel2, track );  // channel2에서 track이 비어
                                               있는지 검사
   }
 input( channel2, track, channel1 );  // channel2의 트랙에 배선정보 저장
```

(그림 12) 채널트랙 할당 알고리즘
(Fig. 12) Channel track assignment algorithm

우에는, 다음에 통과할 채널이 위쪽에 위치해 있는가, 아래쪽에 위치해 있는가에 따라 채널트랙을 결정한다. 다음 채널이 위쪽에 위치한 경우에는, 현재 채널내의 배선되지 않는 트랙중에서 가장 위쪽에 있는 트랙에 금속선을 배치하고, 그와 반대로 다음 채널이 아래쪽에 위치한 경우에는 채널안의 사용가능한 트랙중 가장 아래쪽을 선택한다. 수직방향 채널의 경우에는, (그림 13(b))와 같이 다음에 통과할 채널이 오른쪽에 위치하는가 왼쪽에 위치하는가에 따라 채널트랙의 위치가 결정된다. 다음에 오는 채널의 위치가 오른쪽에 위치한 경우에는, 현재 채널내의 사용가능한 트랙중 가장 오른쪽에 위치한 트랙을 선택한다. 반대의 경우에는 채널내 가장 왼쪽에 위치한 트랙이 선택된다.



(그림 13) 배선트랙 선택의 예
(Fig. 13) An selection example of routing track

3.6.3 스위치 박스에서의 배선

배선 영역을 구성하고 있는 스위치박스 영역은 채널과 채널사이에 존재하는 정의되지 않는 부정형의 공간으로 정의한다. 스위치 박스는 특성상 두개이상의 채널이 연결되기위한 통로역할을 한다. 본 연구에서는 채널의 위치를 모두 정렬하였으므로, 스위치 박스의 형태는 항상 사각형이 된다. 스위치 박스에서의 배선은 최대 4개의 채널에서 나온선이 동시에 배선되므로 배선 복잡도가 상당히 높다. 배선 복잡도가 낮아지기 위해서는 서로 같은 방향의 채널을 연결할 때, 채널과 채널을 연결하는 선들은 되도록 직선이 되도록 해야한다. 불가피할 경우 채널과 채널사이의 연결선이 중간에서 꺾어져야 하는데, 이 경우는 dogleg라는 함수를 통해 해결하도록 한다. dogleg라는 함수는 스위치 박스안에서 꺾어지는 선이 위치할 수 있는 공간을 찾는 역할을 한다. dogleg 함수의 알고리즘은 (그림 14)와 같다.

3.7 CIF 파일 출력

최종 레이아웃 결과는 CIF파일로 출력하게 된다. 출력될 CIF 파일의 이름이나 공정정보는 모두 상세배선의 정보입력단계에서 결정된다. 우선 현재 진행된 상세 배선에서 나온 정보들을 파일로 출력한 후,

```
dogleg( channel channel1, channel channel2, track1, track2 )
  // channel1, channel2 : 각 채널정보를 저장하고 있는 클래스
  // track1, track2 : 각 채널에 배당된 트랙
  {
    findchannel( channel1 );  // channel1과 인접한 다른 방향을 가진 두
                                 channel 을 찾는다.
    for( ; check == 1 ; leg++ )  // leg : 꺾어질 선의 위치정보를 저장
    {
          check = checkleg( leg, channel3, channel4, track1, track2 );
            // 현재 leg에 저장되어 있는 선의 위치가 channel3과 channel4에
             있는 트랙정보와 충돌하는 지 검색
    }
    routing( channel1, channel2, leg );
    // leg의 위치에 꺾어지는 선 배선.
   }
```

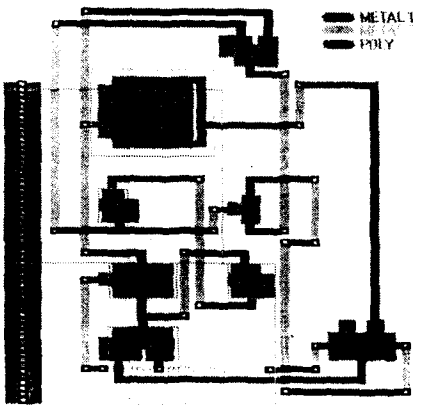
(그림 14) dogleg 함수 알고리즘
(Fig. 14) The algorithm for dogleg function

그 전에 최종배치까지 생성된 정보를 파일로 출력하여 두 정보를 하나의 파일로 합친다. 전체 레이아웃 결과는 상세배선과 모듈들의 정보가 동시에 하나의 파일에 들어있게 된다.

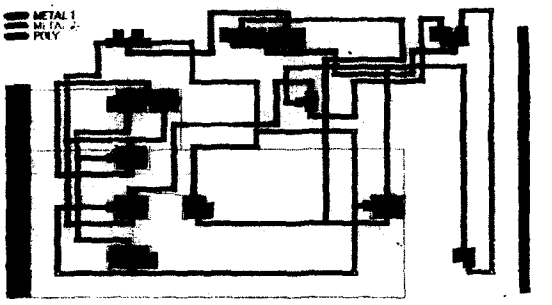
4. 실험 결과

본 논문에서 개발한 INALSYS에 대한 수행검증을 위해 3가지의 회로를 실험에 사용하였다. 세가지의 회로는 각각 2단 비교기, 폴디드 캐스코드 증폭기, 부저항 필터 등이다. 여기서 부저항 필터는 수작업으로 설계된[10] 레이아웃 중 일부를 모듈 라이브러리화하여 수행하였으며, 2단 비교기와 폴디드 캐스코드 증폭기의 경우는 아날로그 자동 합성기 INCSYS[11]에서 합성한 회로이다.

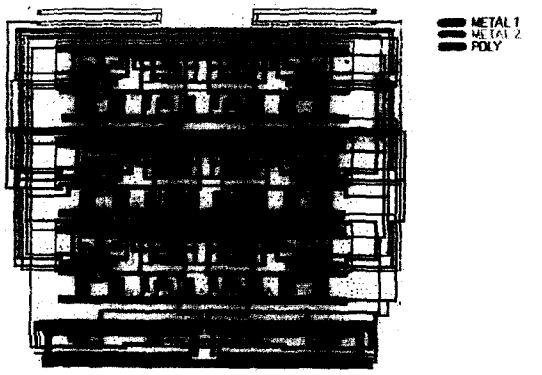
(그림 15)의 2단 비교기 회로는 전체 11개의 소자로 구성되어 있다. 생성된 모듈과 채널의 개수는 각각 8개, 24개이다. 레이아웃에 소요된 시간은 7초가 소요되었다. (그림 16)의 폴디드 캐스코드 증폭기의 경우 생성된 모듈과 채널의 개수는 각각 11개, 32개이다. 전체 레이아웃에 소요된 시간은 12초가 소요되었다. 스위치박스 영역을 사각형으로 규정하여, 모듈과 모듈사이를 정렬하였다. (그림 17)의 부저항 필터의 경우, 지정한 모듈 라이브러리를 사용하여 레이아웃을 수행하였기 때문에, 가장 안정적인 레이아웃 회로를 얻을 수 있었다.



(그림 15) 2단 비교기 회로의 레이아웃 수행결과
(Fig. 15) Layout of the two stage comparator



(그림 16) 폴디드 캐스코드 증폭기의 레이아웃 수행결과
(Fig. 16) Layout of the folded cascode operational amplifier



(그림 17) 부저항 필터의 레이아웃 수행결과
(Fig. 17) Layout of NRL(Negative Resistance Load) filter

전체 모듈의 개수는 5개이고, 수행된 시간은 9초가 소요되었다. 수행된 회로들에 대한 자료들이 〈표 2〉에 제시되어 있다.

〈표 2〉 수행 결과 비교
〈Table 2〉 Comparison of the layout results

수행회로	모듈의 개수	채널의 개수	넷의 개수	단자의 개수	레이아웃 면적	수행 시간
2단 비교기	8	24	8	20	130μm×230μm	7(sec)
폴디드 캐스코드 증폭기	11	32	12	30	533μm×261μm	12(sec)
부저항 필터	5	13	7	51	802μm×948μm	9(sec)

(수행 기종 : Pentium-120MHz PC)

INALSYS의 수행결과를 다른 레이아웃 자동화 도구와 비교하기 위해서 OPASYN[5]의 결과와 비교결과가 〈표 3〉에 표시되어 있다. 대상회로는 OPASYN에서 사용한 2단 CMOS 연산증폭기를 동일하게 사용하였다.

〈표 3〉 본 논문과 OPASYN의 수행결과 비교
〈Table 3〉 Comparison of the execution results with OPASYN

비교 항목	본 논문	OPASYN
대상회로	2단 CMOS 연산증폭기 (동일회로)	2단 CMOS 연산증폭기
수행시간	14초	5분
레이아웃 면적	$1.73\times10^5\mu m^2$	$1.645\times10^5\mu m^2$
사용기종	Pentium-120MHz PC	VAX 8800
사용공정	ISRC 1.5μ공정	MOSIS 3μ공정

〈표 3〉에서 동일한 대상회로에 대해서 본 논문의 INALSYS가 수행시간면에서 장점을 가지는 것을 알 수 있다. 사용한 기종과 공정이 달라서 절대적인 비교는 불가하나 웰합병이 수행되고 인터디지트형의 소자로 레이아웃된 본 논문의 결과가 아날로그 고유의 제약조건을 충실히 만족하고 있다.

5. 결 론

본 논문에서 구현한 레이아웃 자동화 도구 INALSYS는 다음과 같은 특징을 가지고 있다. 첫째, 설계자 혹은 회로 자동합성기가 스파이스 형식으로 제시한 회로에 대해서 레이아웃을 자동화하였다. 연산 증폭기등 특정 회로뿐만 아니라 다양한 회로에 대해서 완전 주문형 방식의 일반적인 레이아웃 자동화 방법론을 제시하였다. 기존의 디지털 레이아웃 자동화 방법론을 개선해서 사용한 아날로그 레이아웃 자동화 프로그램과 비교할 때, 본 논문에서 구현한 자동화 프로그램은 그 시작부터 아날로그의 고유한 특성과 제약조건을 고려하여 구현하였으므로 실제 제작된 회로의 동작을 보장할 수 있다. 둘째, 아날로그 고유의 소자들을 지원하기 위한 변수화된 모듈 라이브러리를 개발하여 확장성을 극대화시켰다. 셋째, 배선 과정을 위하여 개선된 다중 경로 알고리즘을 아날로그 회로의 레이아웃에 적용하여 수행시간 대 성능면에서 만족할 만한 결과를 얻었다. 넷째, 그래픽 사용자 인터페이스(GUI)의 채택으로 전문 설계자의 반복적인 설계 요구의 부하를 덜어줄 수 있는 프로그램으로서 뿐만 아니라 회로설계 초심자로 하여금 부담감없이 전 과정의 자동화를 수행할 수 있게 하여서 교육용 프로그램으로써도 성공적인 사용을 보장할 수 있다.

본 논문에서 개발 및 구현한 아날로그 레이아웃 자동화 도구 INALSYS는 집적 회로 레이아웃의 전 과정을 자동화하는 프로그램으로써 완전 주문형 방식의 아날로그 레이아웃 자동화 방법론을 제시하고 구현하였다.

참 고 문 헌

[1] C. Sechen and A. Sangiovanni-Vincentelli, "TimberWolf3.2: A new standard cell placement and global routing package," *Proc, 23th, Design Automation Conference*, pp. 432-439, 1986.

[2] J. Rijmenants, J. B. Litsos, T. R. Schwarz, and M. G. R. Degrauwe, "ILAC: An automated layout tool tool for analog CMOS circuits," IEEE J. Solid-State Circuits, vol. 24, no. 2, pp. 417-425, 1989.

[3] V. M. Bexten, C. Moraga, R. Klinke, W. Brockherde and K. G. Hess, "ALSYN: Flexible rule-based layout synthesis for analog IC's," IEEE J. Solid-State Circuits, vol. 28, no. 3, pp. 261-267, 1993.

[4] D. J. Chen, J. C. Lee, and B. J. Sheu, "SLAM: A smart analog module layout generator for mixed analog-digital VLSI design," in Proc. IEEE Int. Conf. Computer Design, Cambridge, MA., pp. 24-27, 1989.

[5] H. Y. Koh, C. H. Sequin, and P. R. GRAY, "OPASYN: A Compiler for CMOS Operational Amplifier," IEEE Transaction on Computer-Aided Design, Vol. 9, No. 2, pp. 113-124, February, 1990.

[6] J. M. Cohn, R. A. Rutenbar, and L. R. Carley, "KOAN/ANAGRAMII: New tools for device-

level analog placement and routing," IEEE J. Solid-State Circuits, vol. 26, no. 3, pp. 330-342, 1991.

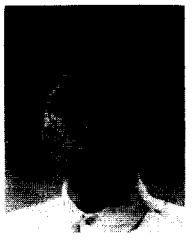
[7] Robert R. Neff, Paul R. Gray, and Alberto Sangiovanni-Vincentelli, "A Module Generator for High-Speed CMOS Current Output Digital/Analog Converters," IEEE Journal of Solid-State Circuits, Vol. 31. No. 3, pp 448-451, March, 1996.

[8] J. D. Bruce, H. W. L, M. J. Dallabetta, and R. J. Baker, "Analog Layout Using ALAS!," IEEE Journal of Solid-State Circuits, Vol 31. No. 2, pp 271-274, February, 1996.

[9] M. H. Harrison, Computer Algorithm, Sara Baase, pp. 167-172, 244-247, 1991.

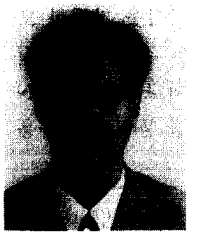
[10] J. S. Hyun and K. S. Yoon, "Design of A 3V-50MHz Analog CMOS Current-Mode NRL Filter," in Proc. IEEE International Symosium on Circuits and Systems, pp. 129-131, 1995.

[11] S. W. Han and K. S. Yoon, "An Analog VLSI Circuit Synthesizer Using Adaptive Algorithm for Implementation of a New Circuit Module," in Proc. IEEE Asia-Pacific Conference on Circuits and Systems, pp. 288-293, 1994.



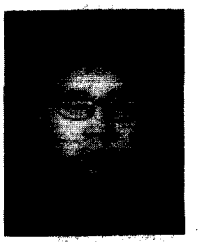
조 현 상

1995년 인하대학교 전자공학과 졸업(학사)
1997년 인하대학교 대학원 전자공학과 졸업(공학석사)
1997년~현재 LG 정보통신 중앙연구소 연구원
관심분야:아날로그 집적회로 설계자동화, 컴퓨터그래픽스



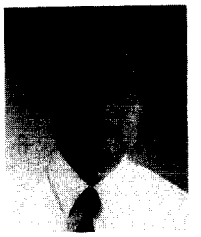
김 영 수

1995년 인하대학교 전자공학과 졸업(학사)
1997년 인하대학교 대학원 전자공학과 졸업(공학석사)
1997년~현재 삼성전자연구소 연구원
관심분야:아날로그 집적회로 설계 자동화, CAD 알고리즘



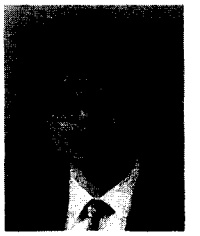
오 정 환

1995년 인하대학교 전자계산공학과 졸업(학사)
1997년 인하대학교 대학원 전자계산공학과 졸업(공학석사)
1997년~현재 LG 정보통신 중앙연구소 연구원
관심분야:집적회로 설계 자동화, 컴퓨터그래픽스



윤 광 섭

1981년 인하대학교 전자공학과 졸업(학사)
1983년 조지아 공대 전자공학과 졸업(공학석사)
1990년 조지아 공대 전자공학과 졸업(공학박사)
1985년~1986년 미국 하니엘사 컨설턴트
1989년~1992년 미국 Silicon System Inc. 선임연구원
1992년~현재 인하대학교 부교수
관심분야:아날로그 집적회로 설계 자동화, 혼성신호 처리 회로설계



한 창 호

1980년 성균관대학교 전자공학과 졸업 (학사)
1982년 서울대학교 대학원 전자공학과 졸업 (공학석사)
1991년 The University of Texas at Austin (공학박사)
1983년~1986년 한국전자통신 연구소 연구원
1991년~1992년 Cadence Design Systems, Inc. 근무
1992년~현재 인하대학교 부교수
관심분야:집적회로 설계 자동화, 컴퓨터그래픽스