論文95-32A-2-16

유연한 구조의 모듈 합성

(Module Synthesis in Flexible Architecture)

吳 命 燮 *, 權 成 勳 *, 申 鉉 哲 *
(Myoung Sub Oh, Sung Hoon Kwon, and Hyun Chul Shin)

요 약

다양한 기능과 성능을 갖는 모듈을 유연한 구조로 자동 합성해 주는 모듈 합성기를 개발하였다. 기존의 모듈 합성기에서 모든 트랜지스터를 수직 또는 수평 한 방향으로만 배치하는 단순한 구조를 사용했던 것과 달리, 새로운 모듈 합성기에서는 트랜지스터의 방향이 90° 의 배수로 자유롭게 정해질 수 있도록 하였다. 유연한 구조는 더욱 많은 디퓨전 영역의 공유를 가능하게 하기 때문에, 모듈의 면적이나 성능의 최적화에 효과적이다. 제안한 방법에서는 먼저 트랜지스터 네트 리스트를 입력으로 받아들여 랜덤하게 초기 배치를 한 후, 주어진 여러 가지 제약을 만족시키면서, 시뮬레이티드 이볼루션 방법을 사용하여 선택된 트랜지스터들을 유연한 구조로 재배치하고 재배선하는 과정을 반복함으로써, 모듈을 합성한다. 배선에는 폴리, 메탈 1, 그리고 메탈 2를 사용하였고, 메이즈에 기초한 배선 기법을 사용하였다. 배선이 불가능할 때에는 배선 영역을 확장하여 배선을 완료시키며, 배선 후에 여분의 배선 영역은 제거하도록 하였다. 제안한 알고리듬을 구현하여 여러 가지 예제에 대하여 실험한 결과, 합성된 모듈은 비교적 만족할 만한 결과를 보여주었다.

Abstract

A symbolic layout generator, called Flexible Module Generator (FMG), has been developed for transforming a given CMOS circuit netlist into an optimized symbolic layout. Contrary to other conventional module generators which place transistors either in horizontal or in vertical direction, FMG places transistors in any multiples of 90°. This flexible layout style can maximize the diffusion sharing and hence can reduce the wire-length for both of area minimization and performance improvement. In FMG, transistors are initially randomly placed and then selected transistors are iteratively replaced using an optimization technique based on simulated evolution. Whenever a transistor is replaced, the affected nets are rerouted. Constraints on the shape, aspect ratio, and critical path delays are considered during the optimization process. Routing is performed by using a modified maze router on polysilicon, metal 1, and metal 2 interconnection layers. Additional routing grids are added, if necessary, for complete routing. Unused rows or columns are removed after routing for area minimization. Experimental results show that FMG synthesizes satisfactory layouts.

* 正會員, 漢陽大學校 電子工學科
(Dept. of Elec. Eng., Hanyang Univ.)
※ 본 논문은 1991 - 1994년도 한국과학재단의 목적 기초 연구지원으로 연구되었음.
接受日字 : 1994년 4月 26日

Ⅰ. 서 론

모듈 합성은 회로의 네트 리스트로 부터 심볼릭 레이아웃(symbolic layout)을 자동 합성하는 과정이다. 공정 기술의 발달과 설계 기술의 발달로 VLSI의 복잡도가 증가함에 따라, 설계자는 회로의 동작 또는 기능 설계에 주력하고 레이아웃은 자동으로 합성하는 것이 설계 기간의 단축에 필수적이다. 합성된 모듈은 레이아웃 방식과 사용되어지는 알고리듬에 의하여 면적과 성능의 효율성이 크게 달라질 수 있다. 모듈을 합성하기 위한 레이아웃 방식은 게이트 메트릭스(gate matrix) 방식[9, 11]과 표준 셀(standard cell)방식 [8, 10]이 주로 사용되어진다.

게이트 메트릭스 레이아웃 방식은 규칙적인 구조라는 장점이 있는 반면, 상단에서 하단까지 이르는 폴리 와이어 때문에 용량성 부하(capacitive load)가 커지게 되어 합성된 모듈의 성능을 저하시키고, 주어진 논리회로에 대하여 수직 폴리 와이어의 개수가 정해져 있기 때문에 레이아웃의 크기를 줄이거나 원하는 가로 세로비(aspect ratio)로 모듈을 합성하기 어려운 단점이 있다. 이러한 단점을 해결하기 위해서, graph-theoretic technique을 이용하여 수직 폴리 와이어의 순서를 재조정함으로써 레이아웃의 크기를 감소시킨 방법이 있고[11], 주어진 회로를 여러 개의 작은 회로(subcircuit)로 분할한 후, 분할된 각 회로를 제한된 게이트 메트릭스 방식으로 합성하여 이들을 배치하고 배선하여 주어진 회로에 대한 심볼릭 레이아웃을 합성한 방법도 있다[1]. 원하는 가로 세로 비로 모듈을 합성하기 위해서, 레이아웃의 행(row)이나 열 (column)의 수를 반복적으로 감소시킴으로써 모듈을 합성한 방법도 발표되었다[4].

표준 셀 방식에서는 확산 브레이크(diffusion break)의 수와 배선 밀도를 최소화하는 연구가 주로 이루어지고 있다. [2]에서는 확산 브레이크의 수를 최소화하는 것이 쌍대 의존적인 레이아웃 방식에 대해 제안된 것과는 달리, 쌍대 독립적인 레이아웃 방식을 제안하였다. 쌍대 독립적인 레이아웃 방식에서는 CMOS 회로의 두 부분, 즉 p-채널 트랜지스터들과 n-채널 트랜지스터들이 서로 독립적으로 최적화 된다. [3]에서는 레이아웃의 면적을 줄이기 위해서 확산 브레이크 수의 최소화에만 중점을 두지 않고 배선 밀도와 트랜지스터의 인접 배치를 동시에 고려하는 혼합 체인화 알고리듬을 사용하였다. 이러한 표준 셀 방식은 게이트 메트릭스 방식에 비하여 원하는 가로 세로 비로 모듈을 합성할 수 있다는 장점이 있는 반면, 주문형(custom)설계에 비하여 면적을 효율적으로 사용하지 못할 수도 있다.

또한 고속 동작을 위해서 가능한 한 메탈 층의 사용을 최대화한 M^3 방식도 있다[5]. M^3 방식은 2개의 메탈 층을 사용하여 지연 시간(delay)을 감소시키고 기능 모듈의 속도를 빠르게 할 수 있는 장점이 있지만, 원하는 가로 세로 비를 얻기가 어렵고 면적 최소화에 제한이 있다[4].

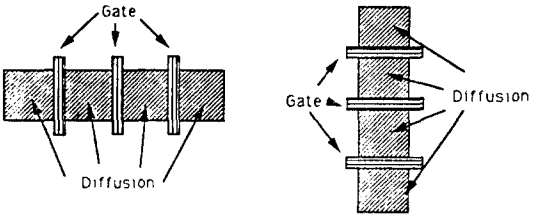


그림 1. 기존의 디퓨젼 공유의 예
Fig. 1. Conventional diffusion sharing.

본 논문에서는 트랜지스터를 모두 수평 또는 수직 방향으로 규칙적으로 배치하는 기존의 레이아웃 방식과는 달리, 합성된 모듈의 면적과 성능을 최적화 하기 위하여, 트랜지스터의 방향이 90° 의 배수로 자유롭게 배치될 수 있도록 한, 새로운 유연한 구조의 레이아웃 방식을 제안한다. 트랜지스터를 자유롭게 배치하면, 디퓨젼 공유(diffusion sharing)를 최대화하고, 배선 길이를 최소화하고, 원하는 가로 세로 비를 얻을 수 있다는 장점이 있다. 그림 1은 기존의 레이아웃 방식에서 사용되어진 디퓨젼 공유의 예로서 모든 트랜지스터를 수직 또는 수평 한 방향으로만 배치하는 구조를 보여준다.

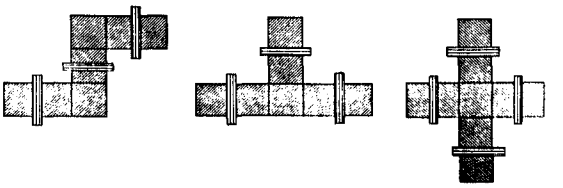


그림 2. 본 논문에서 제안한 디퓨젼 공유의 예
Fig. 2. Diffusion sharing examples in our method.

그림 2는 본 논문에서 제시한 유연한 구조에서의 디퓨젼 공유의 예를 보여준다. 주어진 회로의 네트 리스트를 본 논문에서 제안한 유연한 구조의 모듈로 합성하기 위한 최적화 기법으로는 시뮬레이티드 이볼루션

(simulated evolution)방법[7]을 사용하였다. 시뮬레이티드 이볼루션은, 한 세대(one generation)에서 적응하지 못하는 개체는 제거되고 잘 적응하는 개체들은 계속 생존하도록 하여, 여러 세대가 지나고 나면 최적에 가까운 개체들만 남게 되는 생물학적인 적자 생존 과정에 기초한 방법이다. 이러한 시뮬레이티드 이볼루션 방법을 모듈 합성에 적용하여, 현재의 배치 배선 상태가 나쁜 트랜지스터들을 제거한 후 보다 좋은 위치에 재배치함으로써, 최종적으로 거의 최적에 가까운 심볼릭 레이아웃을 얻기 위한 방법을 개발하였다.

본 모듈 합성기의 새로운 점은, 저자들이 아는 한 최초로, 트랜지스터의 방향이 90°의 배수로 자유롭게 배치되도록 하여 디퓨젼 공유를 최대화하였으며, 배선이 불가능할 때에는 행이나 열의 수를 증가시켜서 배선이 가능하게 하였고, 배선이 완료된 후에는 면적의 최적화를 위해서 여분의 영역은 제거하였다는 것이다. 또한 합성된 모듈의 가로 세로비, 입출력 핀(I/O pin)의 위치, 타이밍등 여러 가지 제약 조건을 만족시키기 위해서, 비용 함수에 이러한 제약 조건들을 포함시켜서 모듈을 합성할 수도 있다. 최적화 방법에서는 시뮬레이티드 이볼루션을 개선하여 사용하였다.

서론에 이어 본 논문은 다음과 같이 구성되어 있다. 제 2 장에서는 새로운 모듈 합성의 전체적인 과정을 설명하고, 제 3 장에서는 단계별 주요 과정의 세부적인 알고리듬을 설명한다. 제 4 장에서는 몇 가지 실험 회로를 이용하여 모듈 합성기의 성능을 알아본다. 마지막으로, 제 5장에서 결론을 기술한다.

Ⅱ. 유연한 구조의 모듈 합성

본 장에서는 유연한 구조로 모듈을 합성하는 전체적인 방법을 설명한다. 그림 3에 전체 알고리듬을 나타내었다.

모듈 합성을 위해서는 먼저 트랜지스터 네트 리스트를 입력으로 읽어 들인 후 초기 해를 구하게 된다. 본 논문에서 제시한 모듈 합성기에 대한 씨드(seed)로서 제공되는 초기해는 기존에 발표된 방법[6, 13]을 이용하여 구하였다. 초기해를 얻은 이후의 모듈 합성 과정은 다음 4단계의 최적화 과정을 반복하여 이루어진다.

첫 번째 단계에서는 현 배치 상태에 대해서 각 트랜지스터의 배치 상태를 평가한다. 본 방법은 시뮬레이티드 이볼루션을 사용하여 배치를 점차 개선시키는 방법이므로, 현재 배치 상태에 대한 평가가 올바르게 수행되어야 효과적으로 배치를 개선할 수 있다.

두 번째 단계에서는 평가된 트랜지스터들 중 현재의 배치 상태가 나쁜 트랜지스터들을 선택하여 레이아웃에서 제거한다. 이 과정은 일부 트랜지스터들과 그 트랜지스터들에 연결된 네트들을 레이아웃에서 제거함으로써, 현재의 배치 상태에 변화를 주어 새로운 배치 상태를 얻기 위한 과정이다.

```
Read data;
Find an initial solution;
While ( generation ≤ NUM_TOTAL_GENERATIONS ) {
        /* create a new generation */
        Evaluate cost of each transistor;
        Select "bad" transistors and remove them from the layout;
        Replace removed transistors and reroute affected nets;

        if ( total cost has been reduced )
          Accept the new generation;
        else
          Keep the original generation;
}
```

그림 3. 전체 알고리듬
Fig. 3. Overall algorithm.

세 번째 단계에서는 제거된 트랜지스터들을 재배치하고 재배선한다. 면적과 성능이 최적화된 모듈을 합성하기 위해서는 제거된 트랜지스터들의 재배치와 재배선 단계는 중요한 과정이다. 제거된 트랜지스터들을 재배치하는 경우, 디퓨젼 공유를 최대화 할 수 있는 위치를 우선적으로 탐색한다. 본 방법은 트랜지스터의 방향이 90° 의 배수로 배치될 수 있도록 하였으므로, 디퓨젼 공유가 가능한 모든 위치에 대해서 모든 방향을 탐색한 후 가장 좋은 위치에 재배치한다. 그러나, 디퓨젼 공유가 불가능한 트랜지스터는 레이아웃에서 재배치 가능한 모든 위치와 방향을 탐색하여 가장 좋은 위치에 재배치한다. 위의 두 가지 탐색 과정을 완료한 후, 영향을 받은 네트들에 대해서 재배선을 한다.

트랜지스터의 재배치 후에 완전한 재배선이 불가능할 때가 있다. 이러한 문제점을 해결하기 위해서 면적은 다소 증가하더라도 배선 경로가 없는 네트들에 대해서는 배선 영역을 확장하여 배선이 완료되도록 하였다. 따라서 재배치 및 재배선이 수행된 후의 해는 일부 트랜지스터가 재배치되고 배선이 완료된 모든 설계 규칙을 만족하는 새로운 상태가 된다.

배선이 완료된 후 여분의 배선 영역이 존재한다면 여분의 배선 영역을 제거하여 레이아웃의 면적을 줄일 수 있다. 여분의 배선 영역은 트랜지스터나 네트들에 의해서 사용하지 않은 행이나 열을 뜻한다. 임의의 배치 상태로부터 새로운 배치 상태를 형성하게 되는 과정을 한 세대(one generation)라고 한다. 본 방법에서는 이러한 세대를 NUM_TOTAL_GENERATIONS 만큼 반복하여 배치를 향상시키게 된다.

마지막 단계에서는 한 세대를 통하여 얻은 새로운

상태를 다음 세대(next generation)를 위한 해로 받아들일 것인가를 결정하게 된다. 다음 세대를 위한 해를 결정하는 경우, hill climbing을 사용하면 보다 전역적인 최적해를 얻을 가능성은 증가하지만, 각 세대에서 트랜지스터들을 재배치하고 재배선하기 때문에 많은 CPU 시간을 필요로 하게 된다. 따라서 수행 시간을 줄이기 위해서, hill climbing은 사용하지 않고 원래의 상태와 한 세대를 완료한 새로운 상태를 비교하여 새로운 상태가 원래의 상태보다 좋은 배치 상태(적은 면적)를 가지는 경우에는 새로운 상태를 다음 세대를 위한 해로 받아들이고, 새로운 상태가 원래의 상태보다 나쁜 배치 상태 (큰 면적)인 경우에는 원래의 상태를 다음 세대를 위한 해로 사용하였다.

Ⅲ. 모듈 합성기의 주요 알고리듬

본 장에서는 모듈 합성기의 주요 알고리듬을 단계별로 설명한다. 본 방법은 시뮬레이티드 이볼루션을 사용하여 점진적으로 배치를 개선시키는 방법이므로, 주어진 상태에 대한 평가가 올바르게 수행되어져야 한다. 이를 위해서는 적절한 비용 함수를 정해야 하고, 전체 트랜지스터들 중에서 배치 상태를 변화시킬 트랜지스터들을 결정하는 기준이 필요하다.

선택된 트랜지스터들을 재배치하고 모든 네트들의 배선을 완료하기 위해서는 경우에 따라 열이나 행의 수를 증가시킬 필요가 있다. 마지막으로 새로운 배치의 선택 기준이 필요하다.

1. 비 용 함 수

일반적으로 한 트랜지스터가 현재의 위치를 계속 유지할 수 있다는 것은 현 배치 상태가 좋은 것을 뜻하고, 현재의 위치를 계속 유지할 수 없다는 것은 현 배치 상태가 나쁜 것을 뜻한다. 초기에 주어진 레이아웃으로부터 적은 면적을 가진 레이아웃을 얻기 위해서는 각 트랜지스터에 대해서 배선 길이, 디퓨젼 공유의 개수, 레이아웃의 행이나 열의 사용 정도 등을 고려한 비용 함수를 사용하여, 현 배치 상태가 나쁜 트랜지스터에는 큰 비용 값을 주어 현재의 위치에서 제거될 가능성을 증가시키고, 배치 상태가 좋은 트랜지스터에는 적은 비용 값을 주어 제거될 가능성을 감소시켰다.

모듈의 면적을 최소화 함과 동시에 레이아웃의 가로 세로 비를 고려하기 위해서 다음과 같은 비용 함수를 정의하였다.

비용 1 : 트랜지스터의 세개의 시그널에 대한 배선 길이의 예측값.

비용 2 : 트랜지스터의 디퓨젼 공유의 개수에 대한 비용.

비용 3 : 트랜지스터가 사용한 행과 열의 비용.

비용 4 : 재배치시 violation 을 일으키는 네트에 대한 비용.

전체 비용 함수는 이러한 4가지의 비용에 가중치를 곱하여 합한 식이다. 각 비용에 대한 가중치를 W1, W2, W3, W4 라고 하면, 전체 비용 함수는 다음과 같다.

$$\text{전체 비용 함수} = W1 * \text{비용 } 1 + W2 * \text{비용 } 2 + W3 * \text{비용 } 3 + W4 * \text{비용 } 4.$$

각각의 비용을 자세히 설명하면 다음과 같다.

비용 1은 트랜지스터의 세개의 시그널에 대하여 각 시그널의 무게 중심(center of mass)에서 트랜지스터의 핀 위치까지의 거리를 모두 합한 비용으로 다음과 같이 정의된다.

$$\text{비용 } 1 = \text{dist(gate)} + \text{dist(source)} + \text{dist(drain)}.$$

단, 핀이 전원/접지 (power/ground) 핀일 때에는

$$\text{dist(pin)} = \text{(distance from the pin to a power/ground track)}.$$

핀이 일반 신호 핀일 때에는

$$\text{dist(pin)} = \text{(distance from the center of mass of the net to the pin)}.$$

시그널의 무게 중심은 비용 1을 구하려는 핀의 위치를 제외한 나머지 핀들의 위치의 무게 중심이다. 비용 1은 합성된 모듈에서 가능한 한 배선 길이를 줄이기 위한 것으로, 한가지 더 고려한 점은 전원/접지 시그널은 일반 시그널에 비해 미리 배치된 전원/접지 선에 배선이 보다 용이하므로, 일반 시그널에 주어지는 가중치 보다 적은 가중치를 주었다. 현재의 구현에서는 전원/접지 선은 설계자가 지정한 수만큼 미리 배치하고, 전원/접지 시그널에 대해서는 일반 시그널에 비해 1/2의 가중치를 주었다.

트랜지스터의 세개의 시그널 중 게이트 시그널을 제외한 드레인과 소오스 시그널은 인접한 트랜지스터의 드레인 소오스 시그널과 같은 시그널일 경우 디퓨젼 공유가 가능하여, 합성되는 모듈의 면적을 줄일 수 있다. 비용 2는 이러한 점을 고려하여 트랜지스터의 디퓨젼 공유갯수에 대하여 다음과 같이 정의하였다.

비용 2 = 한 트랜지스터의 시그널 개수
- 인접한 트랜지스터와 디퓨젼을 공유하는 시그널 개수.

한 트랜지스터의 시그널 개수인 3에서 이 트랜지스터와 인접한 트랜지스터와 디퓨젼을 공유하는 시그널의 개수를 감산하여, 디퓨젼 공유가 많을수록 비용 값이 적어지도록 하였다.

비용 3은 레이아웃의 행이나 열에 트랜지스터들과 배선된 네트들이 사용한 분포를 고려하여, 합성될 모듈의 면적을 최소화하고, 원하는 가로 세로 비로 모듈을 합성하기 위해 사용되어진다. 즉 레이아웃에서 비교적 사용 정도가 적은 행이나 열상의 트랜지스터나 네트들을 다른 행이나 열에 재배치하여, 열이나 행의 수를 줄이기 위해 사용된다. 비용 3을 예를 들어 설명하면 다음과 같다. 그림 4와 같이 트랜지스터와 네트들이 배치되었을 때, 트랜지스터와 네트를 각각 하나의 세그먼트로 가정하고, 각각의 행과 열에 대해서 그 행이나 열을 사용한 트랜지스터와 네트의 개수를 계산한다.

네트의 수를 계산할 때에는 행에 대해서는 수평 방향으로 배선된 네트만을 계산하고, 열에 대해서는 수직 방향으로 배선된 네트만을 계산한다. 이러한 이유는 수직으로 지나는 네트들은 행을 증가시키거나 제거하는 경우 영향을 받지 않기 때문이다. 그림 4의 #usage는 각각의 행과 열에 위치한 트랜지스터와 네트의 개수를 나타낸다.

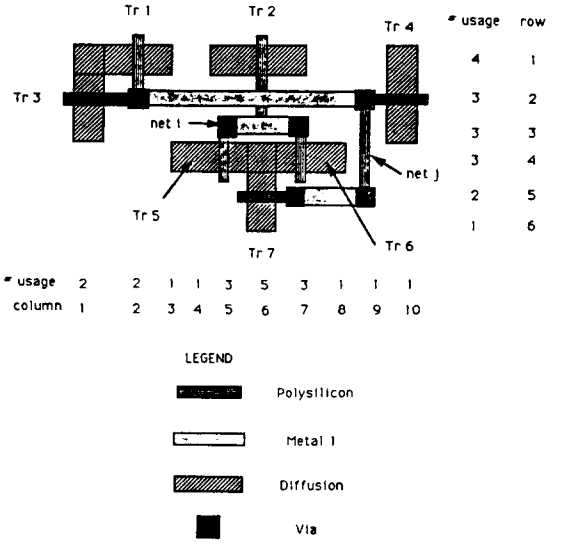


그림 4. 트랜지스터와 네트의 배치
Fig. 4. Placement of transistors and nets.

그림 4의 1번째 행은 Tr1, Tr2, Tr3, Tr4등 모두 4개의 트랜지스터가 사용하였다. 2번째 행은 Tr1과 Tr4, 그리고 네트 j등 모두 3개의 세그먼트가 사용하였음을 알 수 있다. 또한 6번째 열의 경우는 Tr2, Tr5, Tr6, Tr7과 네트 i등 모두 5개의 세그먼트가 사용하였다. 이러한 방법으로, 레이아웃 전체의 행과 열에 대해서 그 위에 위치한 세그먼트의 개수를 계산하면 현 레이아웃에서 트랜지스터나 네트가 배치된 분포를 얻을 수 있으며, 비교적 작은 분포를 가지는 행이나 열상의 트랜지스터를 제거하기 위하여, 트랜지스터 각각에 대해서, 다음과 같이 분포 비용을 구한다. Tr1은 그림 4의 레이아웃에서 1번째 행을 사용하였고 1, 2, 3번째 열을 사용하였다. 먼저 Tr1에 대한 행의 비용값(row_cost)을 구해 보면, 그림 4의 레이아웃에서 1번째 행은 모두 4개의 세그먼트가 공유하고 있으므로, Tr1에 대한 행의 비용 값은 1 / 4 = 0.25 가된다. 또한 그림 4의 레이아웃에서 열 1, 2, 3에는 각각 2, 2, 1개의 세그먼트가 사용되었고, 그 중에 Tr1이 사용한 비용은 열 1, 2, 3에 대하여 구한 비용값중 최대값을 열의 비용값(column_cost)으로 사용하였다. 즉, Tr1에 대한 열의 비용 값은 MAX (1 / 2 = 0.5, 1 / 2 = 0.5, 1 / 1 = 1.0) = 1.0 이 된다.

열에 대한 비용이 1이라는 것은 이 트랜지스터를 다른 곳으로 옮겨서 배선 가능하면, 열을 하나 이상 감소시킬 수 있다는 것을 뜻한다. 이와 같은 방법으로 모든 트랜지스터에 대해서 구한 비용을 표 1에 나타내었다.

행의 비용 값에 대한 가중치를 Wr 이라고 정의하고 열의 비용 값에 대한 가중치를 Wc 라고 정의하면 비용 3은 다음과 같다.

비용 3 = Wr * 행의 비용값 + Wc * 열의 비용값.

표 1. 그림 4의 각 트랜지스터의 행과 열에 대한 비용
Table 1. The row and column costs for each transistor in Fig. 4.

트랜지스터 번호	행에 대한 비용값 (row_cost)	열에 대한 비용값 (column_cost)
Tr 1	0.25	1.0
Tr 2	0.25	0.33
Tr 3	0.5	0.33
Tr 4	0.33	1.0
Tr 5	0.33	1.0
Tr 6	0.33	1.0
Tr 7	1.0	0.25

원하는 가로 세로 비를 얻기 위하여 현재 레이아웃의 폭이 원하는 폭보다 크다면 폭을 우선적으로 줄여야한다. 예를 들어 폭을 줄이는 것이 바람직할 때에는, 모든 트랜지스터에 대해서 열의 비용 값에 대한 가중치를 Wc = 1 으로 주고, 행의 비용 값에 대한 가중치를 Wr = 0.1 의 값을 주어, 폭을 우선적으로 줄일 수 있도록 하였다.

비용 4 는 선택 단계에서는 사용되지 않고, 선택된 트랜지스터를 재배치하는 경우에만 고려한다. 선택된 트랜지스터를 재배치하는 경우, 남아있는 네트들 때문에 배치할 수 있는 영역이 크게 제한될 수 있다. 따라서 트랜지스터를 재배치하는 경우, 전역적인 최적화(global optimization)를 위해서 트랜지스터와 네트의 중첩(overlap violation)을 부분적으로 허용하였다. 비용 4는 이러한 점을 고려하기 위하여 다음과 같이 정의하였다.

비용 4 = 중첩을 유발하는 네트의 개수.

배치 개선 초기에는 전역적인 최적화를 위하여 비용 4의 가중치 W4의 값을 적게 주어 네트의 중첩을 허용하였고, 배치 개선이 진행되면서 점차 큰 가중치 값을 주어, 배치 개선 후반에는 네트의 중첩이 허용되지 않도록 하였다.

한 트랜지스터의 전체 비용을 구하기 위한 가중치는 실험에 의하여 다음과 같이 결정하였다.

W1 = 1, W2 = 5, W3 = 50.

2. 트랜지스터의 선택과 제거

모든 트랜지스터에 대해서 비용을 구한 후, 현재의 배치 상태를 개선시키기 위해서 재배치 해야 할 트랜지스터들을 선택하는 과정이 수행된다.

배치 개선 초기에는 전역적인 범위에서 배치 개선 문제를 해결하기 위하여 전체 트랜지스터 개수의 20% 정도를 선택하였다. 배치 개선이 진행되면서, 선택할 트랜지스터의 개수를 점차 감소시켜서 지엽적인 범위에서 배치 개선이 이루어 지도록 하였다. 선택할 트랜지스터 개수를 5-20%로 동적으로 조정함으로써, 초기 배치에 크게 영향을 받지않고, 면적과 성능을 최적화한 유연한 구조의 모듈을 합성할 수 있다.

배치 상태가 나쁜 트랜지스터를 선택하기 위해서, 앞에서 정의한 비용 함수를 이용하여, 각 트랜지스터의 현 배치에 대한 비용을 구하였다. 그러나, 배치 상태에 대한 비용으로만 트랜지스터를 선택 (즉 가장 큰 비용을 가지는 순으로 선택) 하게 되면, 같은 트랜지스터들이 계속해서 선택될 가능성이 존재한다. 다시 말하면, 비교적 큰 비용을 가진 트랜지스터가 보다 좋은 위치에 재배치되지 않는다면, 배치 개선이 진행되는 동안 계속 선택되어, 더 이상의 최적화가 불가능하게 된다. 이 문제를 해결하기 위하여, 각 트랜지스터의 비용에 난수(random number)를 합산하여 그 합계가 큰 순서로 트랜지스터를 선택하였다. 본 방법에서는 트랜지스터의 비용에 대한 난수 발생 범위를 균등하게 설정하여 그 범위 내에서 난수가 발생되도록 하였다.

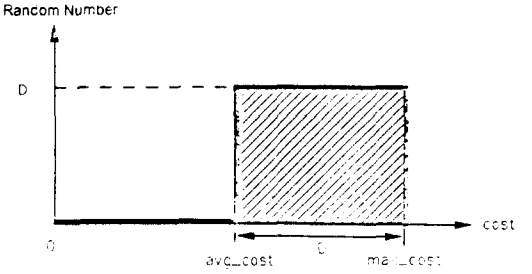


그림 5. 트랜지스터의 비용에 대한 난수 발생 범위
Fig. 5. Range of the random number generated.

그림 5에서 cost는 트랜지스터의 비용을 의미하며 avg_cost는 모든 트랜지스터의 비용을 평균한 값을 의미한다. 그리고 max_cost는 모든 트랜지스터의 비용 중 최대 비용을 뜻하며, D는 최대 비용과 평균 비용의 차이값을 의미한다. 모든 트랜지스터에 대해서 그림 5의 난수 발생 범위 내에서 난수를 발생하여 트랜지스터의 비용에 합산한다. 예를 들어, 트랜지스터의 비용이 평균 비용과 같거나 작은 값을 가지는 경우, 이 트랜지스터에 대한 난수는 영(zero)이 되며, 트랜지스터의 비용이 평균 비용보다 큰 값을 가지는 경우에는 난수 발생 범위 내에서 균등 분포를 갖는 난수를 발생하여 트랜지스터의 비용에 합산한다. 이러한 방법으로 모든 트랜지스터의 비용에 난수를 합산한 뒤, 그 합계가 큰 순서로 정해진 수의 트랜지스터를 선택하게 된다.

난수 발생 범위를 균등하게 설정하는 경우, 현재의 위치에서 평가한 비용에 관계없이, 평균 비용보다 큰 비용을 갖는 모든 트랜지스터들에 대해서 동일한 난수 발생 범위를 사용하여 트랜지스터를 선택하게 되므로, 상당히 넓은 범위에서 배치 개선을 수행할 수 있다.

트랜지스터의 선택 과정이 끝난 뒤, 선택된 트랜지스터와 그 트랜지스터의 세개의 시그널을 레이아웃에서 제거하게 된다. 트랜지스터의 선택과 제거 과정이 완료된 후의 레이아웃은 비교적 좋은 배치 상태를 가지는 트랜지스터들과 이들을 연결하는 네트들 만 포함하고 있으며, 일부 트랜지스터가 제거되었기 때문에 레이아웃의 일부분이 빈 영역으로 남아있게 된다. 이러한 빈

영역은 재배치와 재배선에 사용되어 진다.

3. 트랜지스터의 재배치와 재배선

배치 상태가 나쁜 트랜지스터를 선택한 후, 선택된 트랜지스터를 제거되어진 위치보다 더 좋은 위치에 재배치하기 위하여 배치할 위치를 탐색하게 된다. 재배치하는 순서는 제거되기 전의 비용이 가장 컸던 트랜지스터부터 비용이 감소하는 순으로 한다. 재배치를 위한 탐색은 배치 개선 초기에는 레이아웃 상에 이미 존재하는 네트와 중첩(violation)을 허용하면서 탐색하였고, 배치 개선 후기에는 이러한 네트와 중첩을 허용하지 않았다. 탐색 과정은 크게 두 가지로 나누어 볼 수 있다.

첫 번째 탐색 과정은 디퓨젼의 공유가 가능한 경우로 모든 디퓨젼 공유가 가능한 위치에 대해서 탐색을 하며, 두 번째 탐색 과정은 디퓨젼 공유가 불가능한 경우로서 배치 가능한 모든 위치를 탐색한다. 탐색을 수행할 때, 트랜지스터의 방향을 90° 의 배수로 배치 가능한 모든 방향 (0°, 90°, 180°, 270°)에 대해서 비용을 계산한다. 비용의 계산은 앞에서 설명한 비용 함수를 사용한다. 선택된 모든 트랜지스터들에 대해서 재배치 가능한 모든 위치를 탐색하여 최소의 비용을 갖는 위치에 트랜지스터들을 재배치한다. 재배치가 완료된 후, 배선되지 않은 네트들에 대해서 배선을 수행한다.

본 방법에서는 3개의 배선 층(polysilicon, metal 1, metal 2)을 사용하여 배선하였다. 그러나 폴리 와이어는 다른 와이어에 비하여 저항 성분이 크기 때문에 가능한 한 짧은 연결에만 사용하는 것이 모듈의 성능 향상에 좋다. 또한 메탈 2도 전체 칩 레벨(chip level)배선을 위해서 제한적으로 사용할 수도 있다. 이러한 배선 최적화는 각 층에서의 단위 길이 당의 비용을 적절히 조정하여 이루어진다.

재배치 후에는 제거된 신호를 재배선 하는데, 경우에 따라서는 경로를 찾지 못하는 네트가 존재할 수 있다. 네트의 경로가 존재하지 않을 경우에는, 배선 영역에 행이나 열의 수를 증가시켜서 새로운 경로를 만들어 주도록 하였다.

원하는 가로 세로 비를 갖는 레이아웃을 얻기 위하여, 레이아웃의 가로 세로 비를 고려하여 새로운 행이나 열을 추가하였다. 예를 들어, 현 레이아웃의 가로 세로 비가 원하는 가로 세로 비에 비하여 클 경우에는, 새로운 행을 우선적으로 추가하였다.

경로가 없는 네트의 배선을 위해서 새로운 행이나 열을 추가하는 알고리듬을 예를 들어 설명하면 다음과 같다. 그림 6은 네트의 경로가 존재하지 않는 배선 영역의 예를 보여준다. 그림 6에서 네트 N1으로 연결되어야 할 두핀을 PN1으로 표시하였고, 장애물은 X 로 표시하였다. N2로 표시된 점들은 이미 존재하는 다른 네트의 경로를 뜻하며 ri, cj (i = 1, 2, ..., num_rows : j = 1, 2, ..., num_cols)는 각각의 행이나 열을 나타낸다.

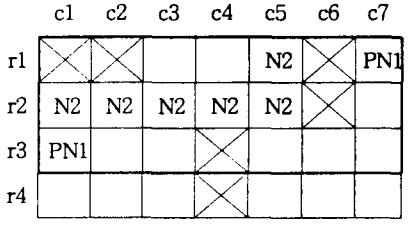
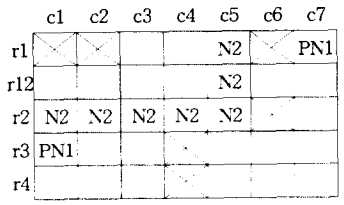


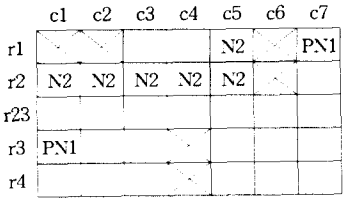
그림 6. 네트의 경로가 존재하지 않는 경우
Fig. 6. An example where the path of a net does not exist.

새로운 경로를 추가할 위치를 찾기 위해서, 연결되지 않은 핀들을 포함하는 직사각형의 탐색 영역(search window)을 구한다. 그림 6의 굵은 실선은 이러한 탐색 영역을 표시한다. 다음은 현 레이아웃의 가로 세로 비를 고려하여 새로운 행을 추가할 것인지 또는 열을 추가할 것인지를 결정한다. 그림 6에서는 행을 추가하는 것으로 가정한다. 그러면, 탐색 영역 내의 모든 행과 행사이에 가상 그리드 행을 하나씩 삽입하여 원래의 행과 가상의 행을 모두 이용하여 배선할 때의 배선 비용을 구한 뒤, 최소 비용을 사용한 가상 행의 위치에 새로운 행을 추가한다.

그림 7의 배선 영역에서는 r1과 r2사이 또는 r2와 r3사이에 가상 행을 사용할 수 있다. 먼저 그림 7의 (a)와 같이 가상의 행 r12를 사용하면, 가상 그리드 r12를 지나가는 N2의 길이는 행 간격만큼 증가하며 이는 비용 계산에 더해진다. 그러나 가상 그리드를 사용한다고 반드시 네트의 경로가 형성되는 것은 아니다. 그림 7의 (a)는 이러한 예로써, 가상의 행을 사용하였지만 네트 N1을 연결할 경로를 형성하지 못한다. 다음은 r2와 r3사이에 위치한 가상 그리드 r23을 사용하여 배선 비용을 구한다. 가상 그리드 r23을 사용함으로써, 네트 N1의 경로를 형성할 수 있다. 따라서 네트 N1을 연결하기 위해서 새로운 그리드를 r23의 위치에 행을 추가하여 배선을 완료한다. 일반적으로 한 레이어(layer)를 수직 방향의 배선에 사용하고 또 다른 레이어를 수평 방향의 배선에 사용하면, 하나의 행과 하나의 열을 추가함으로써 항상 한 네트의 배선을 완료할 수 있다.



(a)



(b)

그림 7. 레이아웃의 행사이에 위치한 가상 그리드
(a) r1과 r2사이에 위치한 가상 그리드 (r12)
(b) r2와 r3사이에 위치한 가상 그리드 (r23)

Fig. 7. Virtual grids located between rows in the layout.
(a) A virtual grid located between r1 and r2.
(b) A virtual grid located between r2 and r3.

위에서 설명한 방법으로 선택된 트랜지스터를 재배치하고 재배선하여 새로운 레이아웃을 얻은 후, 트랜지스터나 네트가 사용하지 않은 행이나 열은 제거하여 레이아웃의 면적을 감소시키도록 한다.

Ⅳ. 실험 결과

지금까지 기술한 알고리듬에 기초한 유연한 구조의 모듈 합성기를 C 언어를 사용하여 SPARC10 워크스테이션에서 구현하였다.

표 2. 간단한 예제 회로
Table 2. Example circuits.

예제 회로	트랜지스터의 개수
add1	28 개
D flip-flop	40 개

본 장에서는 새로운 유연한 구조로 합성된 모듈의 예를 보이고, 기존의 모듈 합성기의 결과와 비교한다. 표 2는 실험에 사용된 예제 회로를 보여준다. 예제 회로 add1은 [12]에서 인용한 회로로서, 1-bit full adder이고, D flip-flop은 [8]에서 사용한 회로이다. 본 방법은 임의의 배치 상태에서 반복적인 배치 개선 과정을 통하여 최종 결과를 얻는 방법이므로, 전체 배치 개선 회수에 대한 레이아웃 결과의 개선과 CPU 시간을 고찰해 볼 필요가 있다.

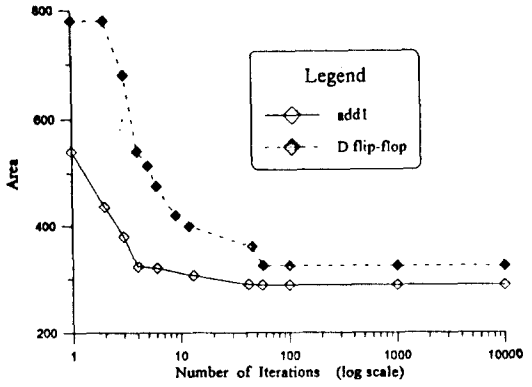


그림 8. 전체 배치 개선 회수 증가에 대한 레이아웃 결과

Fig. 8. Layout results to number of iterations.

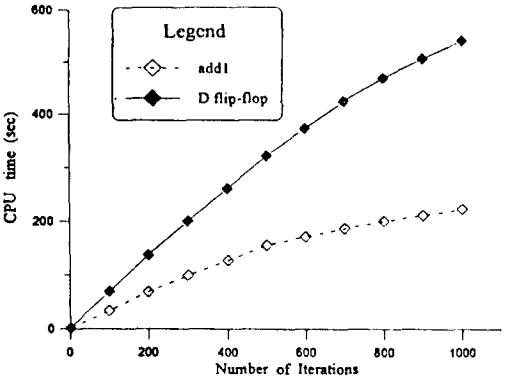


그림 9. 전체 배치 개선 회수와 CPU 시간과의 관계

Fig. 9. CPU time to number of iterations.

그림 8은 D flip-flop 과 add1 예제 회로에 대해서 전체 배치 개선 회수가 10000회까지 증가할 때의 면적의 변화를 보여준다. D flip-flop의 경우 최소의 면적을 얻은 최초의 반복 회수는 57로써, 전체 배치 개선 회수를 더 이상 증가시켜도 결과는 더 이상 개선되지

않았다. 그림 9에는 전체 배치 개선의 회수와 CPU 시간과의 관계를 그래프로 보였다. 일반적으로 시뮬레이티드 이볼루션 방법은 많은 수행 시간을 필요로 하므로, 본 방법은 트랜지스터 수가 200개 정도 이내의 회로 합성에 적합하다. 따라서 보다 큰 규모의 모듈은 계층적으로 분할하여 합성할 필요가 있다.

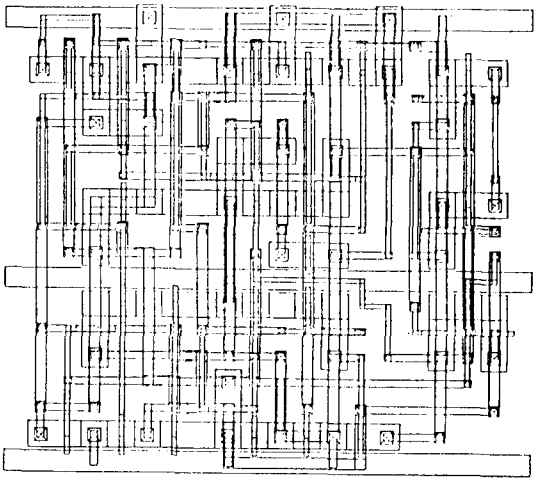


그림 10. D flip-flop 예제 회로의 레이아웃 결과 (18×18 grids)

Fig. 10. Layout result of D Flip-Flop. (18 ×18 grids)

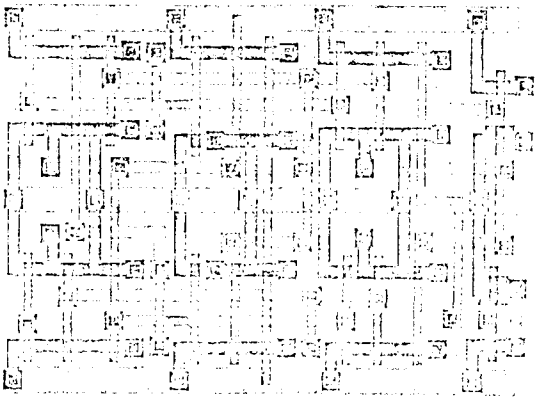


그림 11. [8]에서 제안한 레이아웃 결과 (28× 13 grids)

Fig. 11. Layout result in [8] (28×13 grids).

그림 10은 본 논문에서 제안한 방법으로 합성한 D flip-flop 예제 회로에 대한 최종 결과를 보여주고, 그림 11에는 동일한 예제 회로를 [8]에서 합성한 결과를 보여준다. D flip-flop 예제 회로에 대해 본 방법으로 합성된 최종 레이아웃은 18 X 18 grids를 사용하였고, [8]의 결과는 28 X 13 grids를 사용하였다.

표 3. 여러 원하는 가로 세로 비에 대한 모듈 합성 결과

Table 3. Layout results when desired aspect ratios are given.

예제 회로	원하는 가로 세로비		
	2 : 1	1 : 1	1 : 2
D flip-flop (x, y grids)	1.73 : 1 (26 X 15)	1 : 1 (18 X 18)	1 : 1.73 (15 X 26)
add1 (x, y grids)	1.9 : 1 (21 X 11)	1 : 1 (16 X 16)	1 : 1.5 (14 X 21)

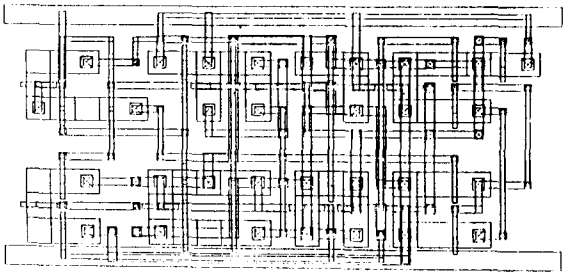


그림 12. add1 예제 회로의 레이아웃 결과 (21 × 11 grids)

Fig. 12. Layout result of add1 (21 × 11 grids).

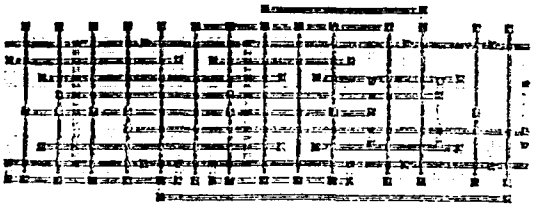


그림 13. [12]에서 제안한 레이아웃 결과 (31 × 12 grids)

Fig. 13. Layout result in [12] (31 × 12 grids).

[8]에서 제안한 결과와 본 방법으로 합성된 모듈을 비교하여 보면, [8]에서는 2개의 층을 배선에 사용하였고, 본 방법에서는 3개의 층을 배선에 사용하여 직접적인 결과 비교는 할 수 없지만, 본 방법으로 합성

된 모듈은 대부분의 트랜지스터가 디퓨젼을 공유하였고 레이아웃의 면적을 매우 효율적으로 사용한 것을 볼 수 있다. 그림 12는 본 방법으로 합성한 add1 예제에 대한 결과를 보여주며, 그림 13은 [12]에서 제안한 방법으로 합성한 모듈의 레이아웃을 보여준다.

또한 표 3은 D flip-flop 과 add1 예제 회로에 대하여 다양한 가로 세로 비가 주어졌을 경우, 합성된 모듈의 가로 세로 비를 보여준다. 이 실험에서는 전체 배치 개선 회수를 100으로 제한하였다. 표 3에서 보는 바와 같이, 본 방법으로 합성된 모듈은 이러한 제약 조건하에서 비교적 좋은 결과를 보여준다.

V. 결 론

기존의 방법에 비하여 유연한 구조의 모듈을 합성하기 위한 새로운 트랜지스터 배치 및 배선 방법을 개발하였다. 본 방법에서는 트랜지스터의 네트 리스트를 입력으로 받아들여 초기 배치한 후, 주어진 배치 상태에 대한 평가를 하고, 현 배치 상태를 향상시키기 위하여 배치가 나쁜 트랜지스터들을 선택한 후 다시 재배치 함으로써, 트랜지스터의 배치 상태를 최적화 하였다. 배치를 향상시키기 위해 적절한 비용 함수를 정의하여 사용하였으며, 시뮬레이티드 이볼루션 기법을 이용하였다. 이러한 방법을 C 언어로 SPARC10 워크스테이션에서 구현하였다. 실험 결과, 본 방법으로 합성된 모듈은 소규모 모듈에서는 만족할 만한 결과를 보였으며, 원하는 가로 세로 비에 가까운 결과를 얻을 수 있었다. 앞으로는 큰 규모의 모듈을 효율적으로 합성하기 위한 연구를 수행할 예정이다. 본 모듈 합성기는 전문 설계자가 설계한 것과 비슷한 형태의 유연한 구조로 모듈을 합성하기 위한 시도라는 점에서 의미가 있다.

참 고 문 헌

[1] D. Baltus and J. Allen,"SOLO:A Generator of Efficient Layouts from Optimized MOS Circuit Schematics," Proc. ACM/IEEE DAC, pp445-452, 1988.

[2] B. S, Carlson, C. Y. R. Chen and U. Singh,"Optimal Cell Generation for Dual Independent Layout Styles," IEEE Trans. on CAD, vol. 10, no. 6, pp770-782, June 1991.

[3] Chi-Yi Hwang, Yung-Ching Hsieh, Youn-Long Lin, Yu-Chin Hsu,"An Efficient Layout Style For 2-Metal CMOS Leaf Cells And Their Automatic Generation," DAC, pp481-486, Jun. 1991.

[4] K. Ho and S. Sastry,"Flexible Transistor Matrix (FTM)," Proc. ACM/IEEE DAC, pp475-480, 1991.

[5] S. M. Kang,"Metal-Metal Matrix(M^3) for High-Speed MOS VLSI Layout," IEEE Trans. on CAD, vol CAD-6, no. 5, pp886-891, Sep. 1987.

[6] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi,"Optimization by simulated annealing," Sci., vol. 220, pp671-680, May 1983.

[7] R. M. Kling and P. Banerjee,"Empirical and Theoretical Studies of the Simulated Evolution Method Applied to Standard Cell Placement, "IEEE Trans. on CAD, vol 10, no. 10, pp1303-1315, 1991.

[8] Y. Lin and D. Gajski,"LES:A Layout Expert System,"IEEE Trans. on CAD, vol 7, no. 8, pp868-876, 1988.

[9] A. D. Lopez and H. S. Law,"A Dense Gate Matrix Layout Method for MOS VLSI," IEEE Electron Devices, vol ED-27, pp1671-1675, 1980.

[10] T. Uehara and W. VanCleemput," Optimal Layout of CMOS Functional Arrays,"IEEE Trans. on Computers, vol C-30, pp305-314, May 1981.

[11] O. Wing, S. Huang, and R. Wang,"Gate Matrix Layout," IEEE Trans. on CAD, vol CA-4 , no. 3, pp220-231, July 1985.

[12] H. Zhang and K. Asada,"A General and Efficient Mask Pattern Generator for non-series-parallel CMOS Transistor Network," IFIP Workshop, pp132-138, 1992.

[13] H. Zhang and K. Asada,"An Improved Algorithm of Transistors Pairing for Compact Layout of non-series-parallel CMOS Networks," CICC, 1993.

저 자 소 개

吳 命 燮(正會員) 제 32권 A편 제1호 참조.
현재 한양대학교 대학원 석사과정

申 鉉 哲(正會員) 제 32권 A편 제 1호 참조.
현재 한양대학교 전자공학과 부교수

權 成 勳 (正會員) 제 32권 A편 제 1호 참조.
현재 한양대학교 대학원 박사과정