최소 면적의 CMOS 기능셀 설계도면을 찾는 휴리스틱 알고리즘

°권 용준, 경 종 민

한국과학기술원 전기 및 전자공학과

A Heuristic Algorithm for Minimal Area CMOS Cell Layout

Yong Joon Kwon and Chong Min Kyung
Dept. of Electrical Engineering, KAIST

Abstract

The problem of generating minimal area CMOS functional cell layout can be converted to that of decomposing the transistor connection graph into a minimum number of subgraphs, each having a pair of Euler paths with the same sequence of input labels on the N-graph and P-graph, which are portions of the graph corresponding to NMOS and PMOS parts, respectively. This paper proposes a heuristic algorithm which yields a nearly minimal number of Euler paths from the path representation formula which represents the given logic function.
Subpath merging is done through a list processing scheme where the pair of paths which results in the lowest cost is successively merged from all candidate merge pairs until no further path merging and further reduction of number of subgraphs are possible.
Two examples were shown where we were able to further reduce the number of interlaces, i.e., the number of non-butting diffusion islands, from 3 to 2, and from 2 to 1, compared to the earlier work [1].

I. 서론

VLSI의 계층적 설계를 위하여 사용되어지는 기본 구성 블럭 라이브러리를 만드는데 있어서 CMOS 기능셀 설계도면의 자동생성은 필수 불가결하다. CMOS 기능셀 설계도면을 자동생성케 하는 방법이 [1]에 서술 되어져 있다. [1]에서 Interlaecs의 수, 즉 분리된 diffusion islands의 수의 감소를 국부적 permutation, mirroring 및 path 결합을 통하여 이루어졌다. 여기서의 path는 트랜지스터들이 직렬 및 병렬로 연결된 것을 의미한다. Fig.1. (a)에 논리함수 f = 1 + 2 * 3 + 4 * 5 의 그래프 모델을 나타내었다. Fig.1. (a) 에서 실선은 N-channel MOSFET's의 연결선도를 나타내는 N-graph라 하고 파선은 P-channel MOSFET's 의 연결도를 나타내는 p-graph라 한다. Fig.1. (b)에는 이에 상당하는 설계 도면을 나타내었다. Fig.1의 경우에서는 N-graph 및 P-graph에 공통되는 Euler path 순열(2-3-1-4-5)가 존재하여 하나의 diffusion island만으로써 주어진 논리함수를 구현할 수 있다.

그러나 [1]의 알고리즘은 뒤에 보이겠지만 어떤 기능셀에 대해서는 최소수의 interlaces를 갖는 설계도면을 찾지 못한다. 그 이유는 [1]의 알고리즘이 트랜지스터들이 직열 및 병렬 연결도를 tree로 나타내었을때 같은 level내에서만 path 결합을 할 수 있기 때문이다. 본 고에서는 trans-level merging이 interlace의 수를 줄이는데 있어서 중대한 역할을 한다는 것을 뒤에 보였다.

또한 [2]에서는 모든 path 종류들을 나타내는 graphs를 18가지 대표적인 형태로 분류 및 정의하고 이들을 path 결합을 하고 Euler paths를 찾는데 사용하였다. 그러나 하나의 직렬 연결된 짝수개의 edges를 두 직렬연결된 홀수개의 edges로 분리하거나 혹은 하나의 병렬 연결된 짝수개의 edges를 두 병렬 연결된 홀수개의 edges로 분리한다면 더 나쁜 결과를 초래할 수 도 있다는 사실을 고려할때 각 edge마다 모든 가능한 결합 경우들을 고려하는 [2]의 초기 단계는 불필요하게 긴 계산 시간을 초래한다.

본 고에서는 위에서 기술한 단점들이 없는, CMOS 기능셀 설계도면을 자동생성케 하는 새로운 방법을 제안하였다. N-graph와 P-graph상에서 공통되는 입력라벨 순열을 갖는 최소수의 Euler paths를 찾기 위하여 path의 종단결합 특성을 갖는 path-hand 개념을 도입하였다. 모든 path-hands는 종단결합 특성에 따라 10가지 종류로 분류된다. 또한 결합형태에 따라 모든 path 결합을 series merging, parallel merging, 그리고 trans-level merging으로 구분하였다.

여러 결합가능한 path쌍들 중 결합시 드는 비용이 최소인 한 쌍을 선택하여 먼저 결합시키는 방식을 사용하였다. list processing에 의해 주어진 paths로 부터 거의 최소수의 Euler paths를 선택적이고 거시적인 전략을 사용하여 찾아 내었다. 원리적으로 최소비용이 한쌍의 paths를 선택하여 결합하는 전략은 greedy방법에 기초를 두고 있으나 [1]의 알고리즘을 사용하여 얻은 결과보다 더 낳은 결과를 얻을수 있었다.

II. Path의 표현 및 Path-hands의 분류

한 path의 다른 한 path와의 결합은 각 path의 결합특성에 의존한다. 여기서는 각 path의 모든 종단 결합특성을 갖고있는 path-hand 개념을 제안하여 두 path의 결합시 직접적으로 각 path를 판찰하지 않고도 단지 각 path-hand만을 가지고 가능케 하였다.

차후 기술될 list processing시 표기의 편의를 위하여 다음과 같은 정의를 내렸다.

정의 1) 하나의 path는 결합시 사용될 수 있는 path-hands라 불리우는 두 종단을 갖는다.

정의 2) 하나의 path-hand는 그것을 한 종단으로서 갖는 path의 종단 결합특성을 나타낸다.

정의 3) 트랜지스터 그래프에서 한 edge의 라벨을 atom이라하고 이것을 한 종류의 expression에 속한다고 한다.

정의 4) 두 operators가 존재하는데 그들은 *와 +이다.

정의 5) 한 expression은 (operator paths or expressions operator)로 나타낸다

정의 6) 한 path는 | path-hand expression path-hand | 로 나타낸다.

path결합시 내부의 expression은 무시하고 path-handa만을 고려한다.

본 고에서는 모든 path-hand를 종단 결합 특성에 따라 10가지 종류로 분류하였다. 이들을 열거해 보면 odd series/parallel path-hands (Fig. 2. (a) 및 (b)), even series path-hands (Fig. 2. (c)), even parallel path-hands (Fig. 2. (d)), series inside path-hands (Fig. 2. (e)), series outside path-hands (Fig. 2. (f)), series anyside path-hands (Fig. 2. (g)), parallel inside path-hands (Fig. 2. (h)), parallel outside path-hands (Fig. 2. (i)), parallel anyside path-hands (Fig. 2. (j)), 그리고 terminated path-hands (Fig. 2. (k))가 있다. 각 path-hand 종류에 대한 표기는 Table 1에 나타낸 바와 같다. 각 path-hand 종류는 그것의 고유한 종단 결합 특성을 갖고 있어서 두 path-hands가 같은 path-hand 종류에 속해 있으면 그 두 path-hands는 등가라고 말한다. 예를 들어보면 Fig. 2. (a)의 odd series/parallel (os/p) path-hand는 직렬 연결 된 홀수개의 edges로 이루어져 있는데 edge의 수가 홀수인 한 그 수가 크고 작음이 종단 결합 특성에는 전혀 영향을 끼치지 못한다. Fig. 2. (b), (c), (d)의 경우도 같은 방식으로 설명되어 질 수 있다. Fig. 2. (e)에 나타낸 series inside (Si) path-hand는 H-graph상 (실선 표시) 에서는 직렬 연결 되어져 있고 P-graph상 (파선 표시)에서는 차우 path결합 시 Pi vertex에서만 결합 가능하다. Fig. 2. (f), (g), (i), (j)의 경우도 같은 방식으로 설명되어 질 수 있다. Fig. 2. (k)에 나타낸 terminated (x) path-hand는 더 이상 결합에 참여할 수 없다.

지금부터는 Fig. 1. (a)의 그래프 모델을 한 예로 들어서 path 표현을 설명하여 본다.

예: Fig. 1. (a)에 대한 입력 list가 다음과 같이 주어진다.

(* (+ 2 3 +) 1 (+ 4 5 +) *)

그러면 차우 기술된 preprocessing에 의해 다음과 같이 초기 paths를 표현한다.

(* | ep (+ 2 3 +) ep | | os/p (+ (* 1 *) +) os/p | | ep (+ 4 5 +) ep | *)

이때 | ep 혹은 ep | 는 even parallel (ep) path-hand의 종류에 속하는 한 path-hand를 나타내고 | os/p 혹은 os/p | 도 같은 방식으로 설명되어 진다.

III. path 결합

앞서 언급한 것처럼 path결합에는 3가지 방법이 있다. 나열하면 series merging (sm), parallel merging (pm) 그리고 trans-level merging (tlm)이다. Table 2에는 각각 두 path-hands를 갖는 두 path가 결합하였을때 생성되는 결과 path의 두 path-hands를 발생 가능한 모든 경우에 대해 나타내었다.

지금부터는 한 예를 들어서 각 path결합 방법에 대해 설명하고자 한다. 주어진 예의 path 표현이 다음과 같다고 하자.

(* | os/p (* 1 2 3 +) os/p | | os/p (+ (* 4 5 6 *) +) os/p |

(+ | os/p (* 7 8 9 *) os/p | | os/p (* (+ 10 11 12 +) *) os/p | +) *)

1) Series Merging (sm)

path | os/p (+ 1 2 3 +) os/p | 와 path | os/p (+ (* 4 5 6 *) + os/p | 가 결합했을 때 series merging 이 일어났다고 하고 생성된 path는 | es (+ 1 2 3 +) (+ (* 4 5 6 *) +) es | 와 같다.

2) Parallel Merging (pm)

path | os/p (* 7 8 9 *) os/p | 와 path | os/p (* (+ 10 11 12 +) *) os/p| 가 결합했을 때 parallel merging 이 일어났다고 하고 생성된 path는 | ei (+ (* 7 8 9 *) (* (+ 10 11 12 +) *) +) ep | 와 같다.

3) Trans - Level Merging (tlm)

Series merging과 parallel merging이 path 표현에서 같은 레벨에 있는 path간의 결합인데 반해 trans-level merging은 path 표현에서 다른 레벨에 있는 path간의 결합이다. path | os/p (* ... *) os/p | 가 path | os/p (+ (* 4 5 6 *) +)와 결합하면 trans-level merging이 일어났다고 하고 생성된 path는 | ni (+ (* 4 5 6 *) +) (+ ((* 7 8 9 *)) po | 와 같다.

IV. Mergeability Check와 결합 알고리즘

각 path-hand는 다른 path-hand와 연결되기전 연결의 가능성을 검토하게 된다. 즉 두 paths간의 mergeability를 check 하여 결합 가능하면 후에 기술된 바와 같이 비용을 계산하게 되고 결합 불능이면 차후부터 두 paths간의 결합을 고려하지 않게 된다. (+ ... +) 혹은 (* ... *) 형태의 expression 내에 존재하는 paths는 같은 형태의 expression 내에 존재하는 paths와 최고 4번까지 결합할 수 있다는 사실을 상기할 때 list processing시 어떤 path-hands가 차후 path 결합시 참여할 수 있는가를 나타내기 위해 약간의 표기 및 규칙을 만들었다.

Table 3과 Table 4에서 double merging은 (+ ... +) 형태의 expression내의 두 expressions가 양끝에서 두번 표기하는데 한 일반적인 예는 다음과 같다.

예) (+ ... [(* ... | ... | ... *) (* ... | ... | ... *)] ... +)

각 경우마다 mergeability를 check 하기 위하여 두 tables를 사용한다. Table 2는 sa, si, so, pa, pi, 및 po 종류의 path-hands 중의 하나와 os/p, es 및 ep 종류의 path-hands 중의 하나가 결합시 사용하고 Table 3는 sa, si, so, pa, pi 및 po 종류의 path-hands 중의 둘의 결합시 사용된다.

여러 결합 가능한 path 쌍 후보들 중 결합비용이 최소인 path 쌍을 선택 하는데 결합 비용은 결합에 의해 생성되는 terminated path-hands의 수와 terminated path-hands는 아니나 연결되어 질 수 없는 path-hands의 수의 합으로 정의된다. 최소 결합 비용을 가진 path 쌍이 여러개일 경우 결합전 최소수의 terminated path-hands를 갖는 path 쌍이 선택된다. 이때 선택된 path 쌍도 여러개일 경우 두 path간의 level 차가 최소인 path 쌍이 선택된다. 이때 선택된 path 쌍도 역시 여러개일 경우 임의의 하나를 선정하여 결합시킨다.

V. 검토 및 결론

본 고에서 제안된 알고리즘의 정당성을 확인하기 위해 Fig.3. (a) 및 (b)에 나타낸 두 예제들을 풀어 보였다. Fig.3. (a)의 예제를 풀어 나아가는 list processing 절차는 다음과 같다.

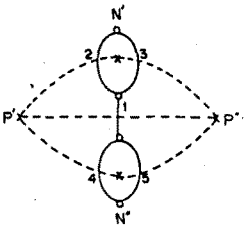
```
(+ (* 1 2 *) (* (+ 3 4 +) (+ 5 6 +) *) (* 7 8 *) +)
          |
          |    preprocessing
          V
(+ | es (* 1 2 *) es | (* |ep (+ 3 4 +) ep | | ep (+ 5 6 +) ep | *)
   | es (* 7 8 *) es | +)
          |
          |    calculation of merging cost
          |    es-tlm-ep : 0 --> selected
          |    es-pm-es : 2
          |    ep-sm-ep : 2
          V
(+ | -#1%1 pi (* 1 2 *) (* (+ 3 4 +) *) po &1%1 |
   | ep | 1%1 (* (+ 5 6 +) *) 1%1 | ep+ | es (* 7 8 *) es | +)
          |
          |    calculation of merging cost
          |    pi-tlm-ep : 1
          |    po-tlm-es | 0 --> selected
          |    ep-tlm-es | 1
          V
(+ | -#1%1 pi (* 1 2 *) (* (+ 3 4 +) *) (* 7 8 *) pi #1%1 |
   | ep | 1%1 (* (+ 5 6 +) *) 1%1 | ep | +)
          |
          |    calculation of merging cost
          |    pi-tlm-ep : 1 --> selected
          V
| -#1%1 pi (+ (* 1 2 *) (* (+ 3 4 +) *) (* 7 8 *)
             | <1> (* (+ 5 6 +) *) <1> | +) po -!1%1 |
          |
          |    output
          V
( 1 - 2 - 3 - 4 - 7 - 8 - 5 - 6 )
```

list processing에 의해 최종적으로 얻어진 edge sequence는 (1-2-3-4-7-8-5-6)인데 한 expression 내의 subexpression 간의 등가관계를 이용하여 재 배열하면 (1-2-5-6-8-7-4-3)이 된다. 이 경우 찾아진 Euler path의 수는 1인데 [1]에 기술된 알고리즘으로 부터 얻은 Euler path의 수는 2이므로 그 수가 감소 되었음을 보았다. 같은 방식으로 Fig.3. (b)도 풀어보았다. 두 예제를 본 고에서 제안된 알고리즘과 [1]에서 기술된 알고리즘으로 풀어보고 그 결과를 간의 비교를 Table 5에 나타내었다.

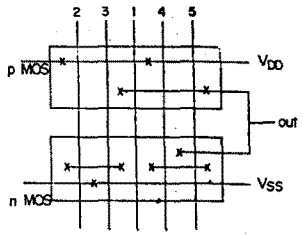
REFERENCES

[1] T. Uehara and W. M. van Cleemput, " Optimal Layout of CMOS Functional Arrays ", IEEE Trans. Computers, vol. C-30, NO.5, May, 1981, pp 305-312.

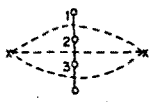
[2] R. Nair, A. Bruss, and J. Reif, " Linear Time Algorithms for Optimal CMOS Layout ", in VLSI : Algorithms and Architectures, edited by P. Bertolazzi and F. Luccio, North-Holland, 1985, pp 327-338.



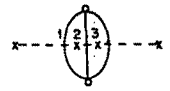
(a) graph model
(solid line for N-graph and dashed line for ⊥-graph)



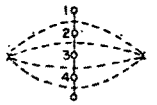
(b) layout for Euler path (2-3-1-4-5)
Fig.1. Graph Model and Layout for out = 1 + 2 * 3 + 4 * 5



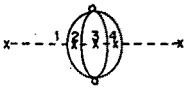
| os/p (* 1 2 3 *) os/p |
(a) odd series path



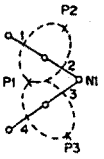
| os/p (+ 1 2 3 +) os/p |
(b) odd parallel path



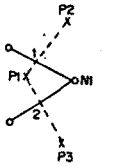
| es (* 1 2 3 4 *) es |
(c) even series path



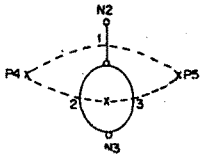
| ep (+ 1 2 3 4 +) ep |
(d) even parallel path



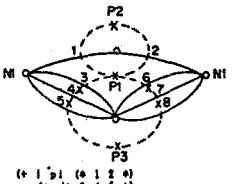
(+ (* --- | s1 (+ (* 1 2 *) +) *)
(* (+ (* 3 4 *) +) s1 | --- *) +)
(e) a path having two series inside path-hands



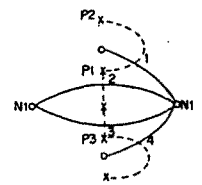
(+ (* --- | s0 (+ (* 1 *) +) *)
(* (+ (* 2 *) +) s0 | --- *) +)
(f) a path having two series outside path-hands



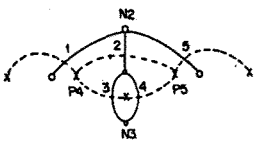
(* --- | ss (+ (* 1 *) +)
(+ 2 3 +) x | --- *)
(g) a path having a series anyside path-hand



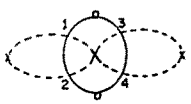
(+ | pi (* 1 2 *)
(* (+ 3 4 5 +) ss
| os/p (+ 6 7 8 +) os/p | *) +)
(h) a path having a parallel inside path-hand



(+ (* --- | ss (+ (* 1 *) +) *)
(* (+ 2 3 +) *) ps |
(* | os/p (+ (* 4 *) +) os/p
| --- *) +) |
(i) a path having a parallel outside path-hand

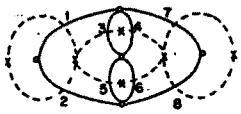


(+ (* --- | os/p (+ 1 +) os/p | *)
| ps (* (+ 2 *) (+ 3 4 +) *) x |
(* | os/p (+ 5 +) os/p | --- *) +)
(j) a path having a parallel anyside path-hand



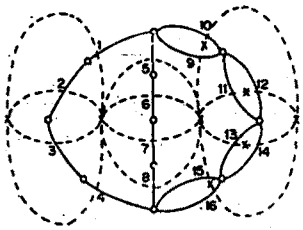
| x (+ (* 1 2 *) (* 3 4 *) +) x |
(k) a path having two terminated path-hands

Fig.2. Ten Different Kinds of Path-hands



(a)

(a) a graph model for Example 1



(b)

(b) a graph model for Example 2

Fig.3. Graph Models for Example 1 and 2

path-heads	abbreviated notation
odd series/parallel	os/p
even series	es
even parallel	ep
series inside	si
series outside	so
series anyside	ss
parallel inside	pi
parallel outside	po
parallel anyside	ps
terminated	x

Table 1. Notation of Merging Characteristics of Each Path-Heads

#	path 1		path 2		kind of merging	merged path	
	the other path-head	merging path-head	merging path-head	the other path-head		path-head 1	path-head 2
1	os/p	os/p	os/p	os/p	sm	es	es
2	os/p	os/p	os/p	os/p	pm	ep	ep
3	os/p	os/p	os/p	os/p	tlm	so	so
4	os/p	os/p	es	es	sm	os/p	os/p
5	os/p	os/p	es	es	pm	po	x
6	os/p	os/p	es	es	tlm	so	si
7	os/p	os/p	ep	ep	sm	ss	x
8	os/p	os/p	ep	ep	pm	os/p	os/p
9	os/p	os/p	ep	ep	tlm	so	po
10	os/p	os/p	ss,si,so	-	sm	ss,so,si	no change
11	os/p	os/p	ss,si,so	-	tlm	so	no change
12	os/p	os/p	ps,pi,po	-	pm	po	no change
13	os/p	os/p	ps,pi,po	-	tlm	so	no change
14	es	es	es	es	sm	es	es
15	es	es	es	es	pm	x	x
16	es	es	es	es	tlm	si	si
17	es	es	ep	ep	sm	ss	x
18	es	es	ep	ep	pm	x	po
19	es	es	ep	ep	tlm	si	po
20	es	es	ss,si,so	-	sm	ss,si,so	no change
21	es	es	ss,si,so	-	tlm	si	no change
22	es	es	ps,pi,po	-	pm	x	no change
23	es	es	ps,pi,po	-	tlm	si	no change
24	ep	ep	ep	ep	sm	x	x
25	ep	ep	ep	ep	pm	ep	ep
26	ep	ep	ep	ep	tlm	po	po
27	ep	ep	ss,si,so	-	sm	x	no change
28	ep	ep	ss,si,so	-	tlm	po	no change
29	ep	ep	ps,pi,po	-	pm	po	no change
30	ep	ep	ps,pi,po	-	tlm	po	no change
31	-	ss,si,so	ss,si,so	-	sm,pm,tlm	no change	no change
32	-	ss,si,so	ps,pi,po	-	sm,pm,tlm	no change	no change
33	-	ps,pi,po	ss,si,so	-	sm,pm,tlm	no change	no change
34	-	ps,p	ps,pi,po	-	sm,pm,tlm	no change	no change

sm : series merging
pm : parallel merging
tlm : translevel-merging
- : a path-hand to be ignored in merging

Table 2. Merging Table

Kind of Merging / Path-Hand	Serial Merging	Parallel Merging case 1	Parallel Merging case 2	Trans level Merging case 1	Trans level Merging case 2
ss	o	-		o*	
si	o	-		x	o*
so	o	-		o*	x
ps	-	o**		o	
pi	-	x	o*	x	o
po	-	o**	x	o	x

case 1 : the case when double merging does not have occurred.
case 2 : the case when doubly merging has occurred.
o : can merge
x : cannot merge
- : cannot occur

* being mergeable if the path having a path-hand | ss —, | si —, or | so — covers only one expression in a (+ — +)
otherwise not mergeable

** being mergeable if the path having a path-hand | ps —, | pi —, or | po — completely covers one or more than one expression in a (+ — +)
otherwise not mergeable

Table 3. Mergibably Check Table I

The One Path-Hand	The Other Path-Hand	Series Merging case 1	Series Merging case2	Parallel Merging case 1	Parallel Merging case 2	Trans-Level Merging case 1	Trans-Level Merging case 2
ss	ss	o		-		o*	
	si	o		-		x	o*
	so	o		-		o*	x
	ps	-		-		o*	
	pi	-		-		x	o*
	po	-		-		o*	x
si	si	x	o	-		x	o*
	so	o	x	-		x	
	ps	-		-		x	o*
	pi	-		-		x	o*
	po	-		-		x	
so	so	x	o	-		o*	x
	ps	-		-		o*	x
	pi	-		-		x	
	po	-		-		o*	x
ps	ps	-		o		o*	
	pi	-		x	o	x	o*
	po	-		o	x	o*	x
pi	pi	-		x	o	x	o*
	po	-		x		x	
po	po	-		o	x	o*	x

case 1 : the case when doubly merging does not occurred.
case 2 : the case when doubly merging has occurred.
o : can merge
x : cannot merge
- : cannot occur

* being mergeable if the path having a path-hand | ss —, | si —, or | so — covers only one expression in a (+ — +)
otherwise not mergeable

Table 4. Mergeability Check Table II

	the number of Euler paths found	
	the algorithm described in [1]	the new general algorithm
Example 1	2	1
Example 2	3	2

Table 5. Comparison of the New General Algorithm with the Algorithm described in [1]