최종 배선을 고려한 연산 회로 합성

엄준형, 김태환
한국과학기술원 전산학과 첨단정보기술 연구센터(AITrc)
jhum@vlsisyn.kaist.ac.kr tkim@cs.kaist.ac.kr

Layout-Aware Synthesis of Arithmetic Circuits

Junhyung Um Taewhan Kim
Dept. of EECS, Korea Advanced Institute of Science and Technology
and Advanced Information Technology Research Center(AITrc)

요 약

현대의 Deep-Submicron Technology(DSM)에서 배선은 논리 구성요소들보다 더욱 중요한 위치를 차지 하게 되었다. 최근에, [2]는 연산 회로를 합성하기 위해 비트 단위의 최적 지연시간의 partial product reduction tree(PPRT)를 생성하는 방법을 제시하였고, 이는 현재의 최적 지연시간을 갖는 회로를 능가한다. 그러나, [2]를 포함하는 기존의 합성방법에서는, 합성의 복잡함이나, 배선에서 발생하는 여러가지 예상치 못하는 문제등으로 인하여 최종 배선을 고려하지 못하는 회로를 생성하며, 이는 길고 복잡하며, 특정한 부분에 밀집 되어 있는 배선을 형성하는 결과를 낳게 된다. 이러한 제한점을 극복하기 위하여, 우리는 carry-save-adder(CSA)를 이용한 새로운 모듈 합성 알고리즘을 제시한다. 이는 단지 상위 단계에서의 회로의 지연시간만을 고려한 알고리즘이 아니라, 이후의 배선을 고려하여 최종 배선에서 규칙적인 배선 토폴로지를 생성한다.

1 서론

연산식의 최적화는 회로 합성의 여러 단계에서 주요하게 연구되어지는 분야 중에 하나이다. 연산식을 최적으로 합성하기 위한 여러 가지 방법들이 제시되어 있다. [1, 2]. 캐리-세이브-가산기(CSA)는 회로 면적의 손해 없이 회로 지연시간을 향상시킬 수 있는 장점을 가지고 있는 연산자이기 때문에 가장 자주 사용되는 연산자 중 하나이다.

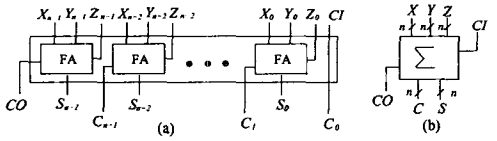

그림 1: n-비트 CSA의 구조

그림 1(a)는 n-비트 CSA의 구조를 보여준다.(그림 1(b)의 블록 심볼은 CSA 모듈을 나타낸다.) VLSI 회로 합성에 있어 CSA를 이용하여 효율적인 allocation이나 변환을 수행하고자 하는 여러 가지 연구가 그 동안 진행되어 왔다. [1]은 CSA 변환을 다음과 같은 세 단계로 수행하였다: (1) 변환될 연산 트리 정의; (2) 선택된 트리를 변환하여 덧셈 연산으로 변환; (3) 선택된 덧셈 연산 트리를 CSA 트리로 변환. 주어진 연산 그래프에서, 위의 알고리즘은 더 이상 CSA 변환이 적용될 수 있는 연산 트리가 발견되지 않을때 까지 반복된다.

또한, 이러한 단어 단계의 CSA를 이용한 연산회로 최적화와는 반대로, *비트-단계의 FA*를 기본 연산자로 사용하여 빠른 곱셈 회로[4]의 설계를 위한 많은 연구가 수행되어 왔다. 비트 단계의 연산식 최적화의 가장 중요한 과제는 FA를 이용하여 빠른 지연시간을 갖는 Partial Product Reduction 트리(PPRTs) 합성하는 것이다. 최근에, [2]은 *two-greedy*라는 알고리즘을 제안하였다. 이는 PPRT를 합성하는데 있어 새로운 기법을 도입하여 *three-greedy approach*를 향상시켰다.

지연시간의 측면에서 보면 FA를 이용한 비트 단계의 합성이 CSA를 이용한 단어 단계의 합성보다 명백하게 우위에 있다. 그와 반대로, 배선의 효율을 위한 레이아웃의 관점에서 보면, CSA를 이용한 회로 합성에서는 배선이 버스로 그룹 단위로 들어가기 때문에 보다 규칙적인 구조를 갖는 장점을 가지고 있다. 본 논문에서, 우리는 CSA를 이용한 최종 배선을 고려한 합성 알고리즘을 제안한다.

2 CSA트리와 지연시간 모델

우리는 *일반적인 CSA 지연시간* 모델을 일반적인 FA지연시간 모델[2]을 확장하여 정의한다. X, Y, Z ($D(X) \le D(Y) \le D(Z)$)를 CSA의 일반적인 세 개의 입력 포트에 할당되는 입력이라 하고, s를 캐리 입력 포트에 할당되는 시그널이라 하자. 그러면, CSA의 *sum*과 *carry*로의 지연시간은 $D(sum) = max\{D(Y)+D_{s1}, D(Z)+D_{s2}\}$, $D(carry) = \{D(Z)+D_c, D(s)\}$로 정의된다. 우리는 이제 일반적 CSA 지연시간 모델에서 최적 지연시간을 갖는 CSA 트리를 생성하는 새로운 CSA 할당 알고리즘을 제안한다.

먼저, 우리는 제안된 방법이 기존의 방법과 어떻게 다른지를 보이기 위한 연산회로 합성의 예를 제시한다. 각 입력에 대한 정보는 그림 2(a)의 표에서 주어진다. 그림 2(b)는 [3]의 알고리즘에 의해 생성된 상수 지연시간 모델 하의 최적 지연 시간의 CSA-트리 구조를 보여주며, 우리는 이를 *circuit1*이라 부른다. 그와 반면에, 그림 2(c)는 [2]에 의해 생성된 일반적인 FA 지연시간 모델에서의 최적인 FA-트리의 구조를 보인다. 우리는 이를 *circuit2*라 한다. *circuit1*와 *circuit2*의 비교에서, 우리는 *circuit1*의 배선이 좀더 규칙적인 구조를 갖는 것을 볼 수 있으며(버스 단위로 그루핑 됨) 이는 *circuit1*이 좀더 상위 단계의(단어 단계의) 모듈 생성방법에 의해 생성되었기 때문이다. 그러나, *circuit2*는 지연시간에 있어 *circuit1* 보다 빠른데, 이는 *circuit2*가 좀더 하위 단계의(비트 단위) 최적화 방법으로 생성되었기 때문이다.

우리의 접근 방법은, [3]과 [2]에서 제시된 방법의 장점을 동시에 이용하는 것이다. 그림 2(d)는 일반적인 지연시간 모델을 이용하여 지연시간이 향상된 CSA-트리의 구조를 보이고 있다. 우리는 이 회로를 *circuit3*이라 부른다. 결과적으로, *circuit3*의 지연시간(=11)은 *circuit1*(=13)의 지연시간보다 짧으며, *circuit3*의 배선은 *circuit1*과 같이 규칙적이다. 우리는 이러한 *circuit3*의 지연시간을 FA-단계의 배선 재배치를 이용하여 더욱 향상시킨다. 그림 2(e)의 *circuit4*는*circuit3*에서 더욱 최적화된 CSA트리의 결과

를 보인다. 여기서, 점선 안의 FA는 입력 배선이 재배치된 FA이다. 예를 들어, FA_9에서는 첫번째와 세번째 입력이 출력 지연시간을 감소시키기 위해 재배치 되었다. 원래의 CSA-단계의 구조는 FA-단계의 입력 재배치때에 그대로 유지 되기 때문에, *circuit4*의 배선의 규칙성은 광범위한 시각에서 보았을때 그대로 유지된다. 그러나, *circuit4*의 주요 경로의 지연시간(=10)은 감소 되었으며, *circuit2*의 비트 단계의 최적 지연시간(=9)에 더욱 가깝게 된다.

또한, 그림 2(f)와 (g)는 *circuit2*와 *circuit4*의 FPGA 방식의 mapping결과를 보인다. 두꺼운 선은 주요 경로를 나타내며, 우리의 배선을 고려한 합성이 그렇지 않은 방식보다 최종 배선에 있어 더 좋은 결과를 생성함이 보여진다.

3 배선을 고려한 합성

본문에서 제시되는 배선을 고려한 합성은 두 단계로 진행된다. 실행되어야 하는 주어진 연산식에 대해, 우선, 우리는 일반적인 CSA 지연시간 모델에서 최적의 지연시간을 갖는 *부분-Greedy-CSA*알고리즘을 수행하며, 이후의 FA의 입력의 재배열등의 FA단계의 최적화를 통해(그림 2참조) 회로의 지연시간을 향상시킴과 동시에 생성된 CSA의 블록-단위의 배선 구조를 유지시킨다.

CSA 할당 문제는 다음과 같이 표현된다: *(문제 1) 주어진 연산식*[1] *연산식*

$$F = X_1 + X_2 + \cdots + X_m + s_1 + s_2 + \cdots + s_n + c \quad (1)$$

과 멀티-비트 피연산자 $X_1, \cdots X_m$ 싱글-비트 피연산자, $s_1, \cdots, s_n$, 그리고 상수 c와 각 피연산자의 지연시간이 주어졌을때, 일반적인 CSA 지연시간 모델에 대해, F를 위한 최적 지연시간을 갖는 CSA-트리 구조를 만든다. 먼저, 문제 1의 제한적인 경우를 생각해보자.

부분문제 1 : 문제 1에서 $c = 0, m - 1 \geq n$.

부등식$m-1 \geq n$는 멀티-비트 피연산자의 개수가 충분히 커서 모든 싱글-비트 입력이 CSA의 캐리 입력으로 사용이 가능한 것을 의미한다. R을 부분 문제 1에 대한 모든 가능한 CSA-트리의 집합이라 하자. 이때 모든 싱글 비트 입력은 반드시 CSA의 캐리 입력에 할당 되어야 한다. 이제, 다음과 같은 CSA-트리 생성방법을 고려해보자. 우리는 이 알고리즘을 *부분-greedy-CSA*알고리즘이라 부른다: 우리는 각 반복에 있어 피연산자를 합하는 CSA를 하나씩 할당한다. 세개의 멀티 비트 입력을 새로운 CSA의 입력으로 할당하며, 그중 첫번째와 두번째로 빠른 입력을 멀티비트로, 가장 빠른 싱글 비트 입력을 싱글 비트 입력으로 할당한다. 이때, 이러한 *부분-greedy-CSA*알고리즘은 세번째의 입력의 지연시간에 다른 지연시간과 구조를 가지게 된다(그림 3). A를 *부분-greedy-CSA*알고리즘에 의해 생성되는 CSA트리의 집합이라 하자. 우리는 논의의 편의를 위해 다음과 같은 기호를 정의한다: CSA_i는 *부분-greedy-CSA*방법의 i번째 반복에서 생성되는 CSA라 하고, a_i, b_i, d_i $(D(a_i) \leq D(b_i) \leq D(d_i))$를 CSA_i의 세 개의 일반적인 입력에 할당 되는 멀티 비트 입력이라 하며(그림 1의 왼쪽에서부터 X, Y, Z), s_i를 CSA_i의 *carry-in*포트에 할당되는 싱글비트 입력이라 하자. 그리고, $carry_i$ sum_i는 각각 CSA_i의 *carry*와 *sum* 시그널을 의미한다. 이제, T를 *부분-greedy CSA*방법에 있어 CSA를 n번 할당함으로써 생성되는 회로라 하자. 우리는 CSA-트리 T의 시그널x의 도달시간을 나타내기 위해 $D(x,T)$의 기호를 사용하며, $D(T)$를 $max\{D(x_1,T),\cdots D(x_n,T)),D(y_1,T),\cdots D(y_n,T)\}$ 로 놓는다. 이때, x_i와 $y_i(i=1,\cdots n)$는 트리 T의 CSA_i의 *carry*, *sum* 시그널을 각각 의미한다.[2]

정리 1 *R의 어떤 T에 대해서도, $D(T') \leq D(T)$를 만족하는 CSA-트리 $T' \in A$가 존재한다.*

[1]연산식은 뺄셈이나 곱셈등을 포함할 수 있다. 이러한 연산식을 덧셈 연산식으로 변환하는 방법은, [1]에 자세히 나타나 있다.

[2]우리는 $D(v,T)$의 트리 T의 의미가 명확할 경우, $D(v,T)$대신 $D(v)$를 간단히 사용한다.

Word	Bit	Delay(Word)	Delay(Bit)
X_1	$x_{1,5}\ x_{1,4}\ x_{1,3}\ x_{1,2}\ x_{1,1}\ x_{1,0}$	2	0, 0, 2
X_2	$x_{2,5}\ x_{2,4}\ x_{2,3}\ x_{2,2}\ x_{2,1}\ x_{2,0}$	2	1, 0, 2
X_3	$x_{3,5}\ x_{3,4}\ x_{3,3}\ x_{3,2}\ x_{3,1}\ x_{3,0}$	4	3, 0, 4
X_4	$x_{4,5}\ x_{4,4}\ x_{4,3}\ x_{4,2}\ x_{4,1}\ x_{4,0}$	7	7, 1, 1
X_5	$x_{5,5}\ x_{5,4}\ x_{5,3}\ x_{5,2}\ x_{5,1}\ x_{5,0}$	5	4, 3, 5
X_6	$x_{6,5}\ x_{6,4}\ x_{6,3}\ x_{6,2}\ x_{6,1}\ x_{6,0}$	6	0, 6, 1
X_7	$x_{7,5}\ x_{7,4}\ x_{7,3}\ x_{7,2}\ x_{7,1}\ x_{7,0}$	9	8, 8, 9

(a) Operands to be added

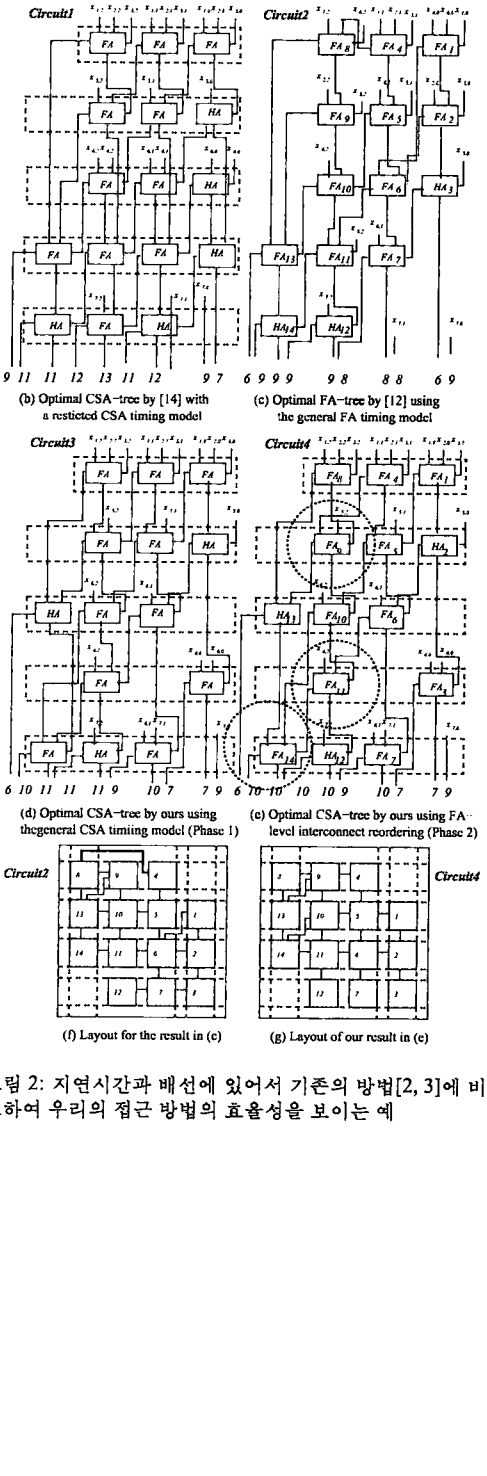

그림 2: 지연시간과 배선에 있어서 기존의 방법[2, 3]에 비교하여 우리의 접근 방법의 효율성을 보이는 예

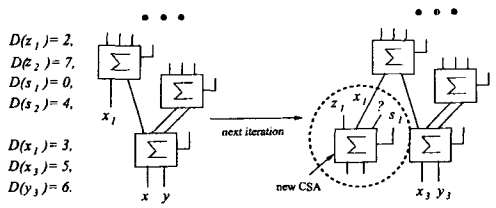

그림 3: *부분-greedy-CSA* 알고리즘을 보이는 예

*정리 1*는 *부분-greedy-CSA*를 이용한 방법이 method R에서 생성된 최적 지연시간을 갖는 CSA-트리를 찾는데 있어 search space를 줄일 수 있음을 말하고 있다. 나머지 싱글비트 입력이나 상수를 다루는 방법은 [3]의 방법과 동일하며, 그러한 과정이 일반적인 CSA 지연시간 모델에 대해 최적의 CSA-tree를 생성한다.

*부분-greedy-CSA*할당 방법을 이용하여 최적 지연시간의 CSA-트리 구조를 찾기 위해서 우리는 branch and bound search를 이용한다. [3]의 CSA 할당알고리즘은 $D_{s1} = D_{s2}$일 경우 최적의 해를 찾아내기 때문에, 우리는 D_{s2}와 D_{s1} ($D_{s2} \le D_{s1}$)가 다르게 모델링 되어 있는 우리의 경우에, [3]으로 찾아진 CSA트리의 지연시간을 upper bound로 이용한다. 그리고, 우리는 다음과 같은 pruning을 위한 여러 방법들을 이용해서 search의 수행시간을 줄인다. 이제, T를 *부분-greedy-CSA* 방법에 의해 생성된, 즉, $T \in A$를 만족하는 CSA 트리라 하자.

성질 1 *트리 T의 두 개의 CSA CSA_i, $CSA_j(i < j)$가 다음과 같은 조건,*[3] $D(d_i) \le D(b_j)$, $D(d_i) > D(a_j)$,*을 만족할 때,* $D(T') \le D(T)$*를 만족하는 CSA-트리 T'가 A에 존재한다.*

성질 2 *만약 트리 T의 CSA_i, $CSA_j(i < j)$가 다음과 같은 조건 $D(b_j) < D(d_i) \le D(d_j)$, $D(d_j) \ge D(d_i) + D_{s1} - D_{s2}$을 만족시킨다면, 다음과 같은 CSA-트리 $T' \in A$ such that $D(T') \le D(T)$가 존재한다.*

성질 3 *트리 T의 CSA_i, $CSA_j \in T$ $(i < j)$가 $D(d_i) > D(d_j)$을 만족시키면, $D(T') \le D(T)$을 만족시키는 CSA-트리 T'가 A에 존재한다.*

4 실험 결과

우리는 *부분-Greedy-CSA*과 이후의 FA 재배열로 이루어진 일련의 배선을 고려한 합성 방법의 효율을 보이기 위해 실험을 수행하였다. 주어진 연산식에 대해 본문에 제시된 알고리즘을 이용하여 CSA-트리를 만들고, CSA트리의 규칙적인 구조를 유지시키며 FA의 입력을 그림 2에서 주어지는 바와 같이 재배열하였다. 우리의 알고리즘은 C++로 작성되었으며, Pentium-3 500MHz Linux machine에서 측정되었으며, Stelling의 알고리즘에 의해 최적의 지연시간을 갖도록 생성된 결과와 우리의 결과를 비교하였다.[4] 우리의 알고리즘과 Stelling의 알고리즘을 여러 test-case에 적용한 후, 우리는 placemet를 Fiduccia and Mattheyses's partitioning 알고리즘을 사용하여 적용한 후,[5] 라우팅을 위해[6] Lee's maze routing 알고리즘을 사용하였다. 우리는, CSA와 FA의 여러 가지 지연시간 상수를 측정하기 위해 Synopsys Design Compiler(*lcbg10pv*(0.35*u*) technology library [6])를 사용하였다. 또한, FA 셀간의 지연시간 측정을 위해, L이 라우팅 경로 위의 블록의 텝개수라 할때, $\alpha \cdot L^2$를 지연시간을 측정하기 위해 사용하였다. α는 게이트와 배선의 적절한 지연시간 차이를 반영하기 위한 상수이며, 배선의 스타일과 target library에 따라 다르게 적용될 수 있다. 지연시간 측정을 위해, L이 라우팅 경로 위의 블록의 텝개수라 할때, $\alpha \cdot L^2$를 지연시간을 측정하기 위해 사용하였다. α는 게이트와 배선의 적절한 지연시간 차이를 반영하기 위한 상수이며, 배선의 스타일과 target library에 따라 다르게 적용될 수 있다.

표 1: 임의의 연산식에 대한 결과 Exp.1 : $X_1 + \cdots + X_6$; Exp.2 : $X_1 \cdot X_2 + X_3 + X_4$; Exp.3 : $X_1 + X_2 + X_3 - X_4 + X_5 - X_6 - X_7$; Exp.4 : $X_1 + \cdots + X_8$; Exp.5 : $X_1 \cdot X_2 + X_3 \cdot X_4 + X_5 \cdot X_6$; Exp.6 : $X_1 \cdot X_2 + X_3 \cdot X_4 - X_5 \cdot X_6 - X_7 \cdot X_8$;

Exp.	α	Stelling[2] crit. path delay/ long. net delay	Ours crit. path delay/ long. net delay	Improvement crit. path delay/ long. net delay
Exp. 1	0.04	11.27 / 3.24	8.87 / 1.96	21% / 40%
	0.02	6.78 / 1.62	5.59 / 0.98	18% / 40%
	0.01	4.54 / 0.81	3.95 / 0.49	13% / 40%
Exp. 2	0.04	14.69 / 6.76	8.14 / 3.24	45% / 52%
	0.02	7.89 / 3.38	4.86 / 1.62	38% / 52%
	0.01	4.49 / 1.69	3.22 / 0.81	28% / 52%
Exp. 3	0.04	46.43 / 17.64	18.22 / 9.00	61% / 49%
	0.02	23.79 / 8.82	9.50 / 4.50	60% / 49%
	0.01	12.48 / 4.42	5.14 / 2.25	59% / 49%
Exp. 4	0.04	21.92 / 9.00	14.43 / 6.76	34% / 25%
	0.02	12.16 / 4.50	8.49 / 3.38	30% / 25%
	0.01	7.28 / 2.25	5.52 / 1.69	24% / 25%
Exp. 5	0.04	18.42 / 9.00	13.41 / 6.76	27% / 25%
	0.02	10.42 / 4.50	7.96 / 3.38	24% / 25%
	0.01	6.42 / 2.25	5.24 / 1.69	18% / 25%
Exp. 6	0.04	19.27 / 9.00	11.88 / 3.24	38% / 64%
	0.02	10.87 / 4.50	7.48 / 1.62	31% / 64%
	0.01	6.67 / 2.25	5.28 / 0.81	21% / 64%
Average	0.04	29.06 / 9.11	12.49 / 5.16	38% / 43%
	0.02	11.99 / 4.56	7.31 / 2.58	34% / 43%
	0.01	6.98 / 2.28	4.73 / 1.29	27% / 43%

5 결론

본 논문에서, 우리는 연산 회로를 위한 새로운 RT-단계의 접근방법을 다음과 같은 두 가지의 목적, 빠른 지연시간과 바람직한 배선이라는 목적을 가지고 제시하였다. DSM 기술에서는, architectural/logical 합성간의 관계가 점점 중요시 되며, 결과적으로 합성의 초기 단계에서 마지막 배선을 고려하는 것이 필요하게 되었다. 이러한 관점에서, 우리는 본 논문에서 제시된 알고리즘이 연산식의 데이터 경로를 최종 배선을 고려하여 합성하는데 있어서 효율적으로 사용될 것이라 생각한다.

감사의 글 : 본 논문은 첨단정보기술 연구센터(AITrc)를 통하여 과학재단의 지원을 받았음.

참조 서적

[1] T. Kim, W. Jao, and S. Tjiang, "Circuit Optimization using Carry-Save-Adder Cells", *IEEE Tran. on Computer-Aided Design of Circuits and Systems*, Vol.17, No.10, pp.974-984, October 1998.

[2] P Stelling, C.U. Martel, V.G. Oklobdzija, and R. Ravi, "Optimal Circuits for Parallel Multipliers", *IEEE Trans. on Computers*, Vol. 47, No. 3, pp. 273-285, March 1998.

[3] J Um and T. Kim, "An Optimal Allocation of Carry-Save-Adders in Arithmetic Circuits", *IEEE Trans. on Computers*, Vol. 50, No. 3, pp. 215-232, March 2001.

[4] C.S. Wallace, "A Suggestion for a Fast Multiplier", *IEEE Trans. on Computers*, Vol. 13, pp. 14-17, 1964.

[5] C. Lee, "An Algorithm for Path Connections and its Applications", *IRE Trans. on Electronic Computers*, VEC-10. pp. 346-365, 1961. New York, 1979.

[6] LSI Logic Inc., *G10-p Cell-Based ASIC Products Databook*, 1996.

[3] CSA_i는 *부분-greedy-CSA* 알고리즘의 i번째 반복수행에 생성된 CSA이다.

[4] [2]의 알고리즘에 최적이라 함은 지연시간만을 고려한 결과이다.

[5] 우리가 목적한 레이아웃은 격자 구조이다. 하지만 우리의 알고리즘은 어떤 경우의 standcell이나 FPGA도 적용이 가능하다.

[6] 우리는 레이아웃의 the channel capacity를 4로 놓았다.