연산회로 최적화를 위한 배선의 재배열

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

A Reorering of Interconnection for Arithmetic Circuit Optimization

Junhyung Um and 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)에선 배선에 관련된 문제, 예를 들어 crosstalk이나 노이즈 등이 큰 문제가 된다. 그리하여, 배선은 논리 구성요소들보다 더욱 중요한 위치를 차지 하게 되었다. 우리는 이러한 배선을 고려하여 연산식을 최적화 하기 위해 carry-save-adder(CSA)를 이용한 모듈 합성 알고리즘을 제시한다. 즉, 상위 단계에서 생성된 규칙적인 배선 토폴로지를 유지하며 CSA간의 배선을 좀더 향상시키는 최적의 알고리즘을 제안한다. 우리는 우리의 이러한 방법으로 생성된 지연시간이 [1]에 가깝거나 거의 근접하는 것을 많은 testcase에서 보이며(배선을 포함하지 않은 상태에서), 그리고 그와 동시에 최종 배선의 길이가 짧고 규칙적인 구조를 갖는것을 보인다.

1 서론

연산식의 최적화는 회로 합성의 여러 단계에서 주요하게 연구되어지는 분야 중에 하나이다. 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.

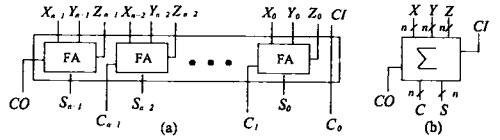


그림 1: n-비트 CSA의 구조

n-비트 CSA는 서로 독립적인 n개의 full adder(FA)들로 구성되어 있다. 일반적인 가산기, 예를 들어 리플 캐리 가산기나 캐리-lookahead 가산기와는 다르게, CSA내부의 FA간에는 캐리 지연시간이 없기 때문에, 충분히 큰 n값에 대해, CSA를 이용한 회로 연산은 일반적인 가산기를 사용한 경우보다 지연시간에 있어 월등히 빠르며, 그와 동시에 비교적 적은 회로 면적을 갖는다.

VLSI 회로 합성에 있어 CSA를 이용하여 효율적인 allocation이나 변환을 수행하고자 하는 여러 가지 연구가 그 동안 진행되어 왔다. CSA를 이용한 회로 합성에서는 배선이 버스로 그룹 단위로 들어가기 때문에 보다 규칙적인 구조를 갖는 장점을 가지고 있다. 그러므로, 우리는 이미 기존의 전체적인 CSA-트리 구조를 유지하며 *CSA 모듈간의 배선을 최적으로 재 합성* 한다.

실험에서, 우리는 위의 방법으로 생성된 회로의 지연시간이 [1]에 의해 생성된 비트 단계의 최적의 회로의 지연시간에 근소함을 보였으며 (회로의 지연시간을 고려하지 않으며), 또한 동시에 배선에 있어 규칙적이고 보다 짧은 배선을 갖는 것을 보였다.

2 FA 지연시간 모델

p, q, r을 FA의 세 개의 입력 포트라 하고, FA의 캐리 출력, sum 출력 포트를 각각 *carry*, *sum*이라 하자. x, y, z ($D(x) \le D(y) \le D(z)$)는 각각 p, q, r 에 할당되는 비트 입력이라 하자. FA의 기본적인 논리 구조에서 볼 때(그림 2), FA가 *sum*를 생성하는데는 입력으로부터 두 단계의 XOR을 지나는 데에 걸리는 시간이 소요된다. 그러므로, 우리는 *sum*을 모델링하기 위해 다음과 같은 두 개의 상수 D_{s1}와 D_{s2}를 사용한다. D_{s1}은 q로부터 *sum*포트까지의 지연시간을 의미하며, D_{s2}는 r로부터 *sum*포트까지의 지연시간을 의미한다. 또한, 우리는 D_c(2-level NAND delay)를 FA의 *carry* 지연시간을 나타내기 위해 사용한다. (그림 2.)

우리는 위의 FA 지연시간 모델을 *FA의 일반적인 지연시간 모델*이라 한다. 결과적으로, 입력으로부터 *sum*, *carry* 시그널까지의 지연시간은 Consequently, the arrival times for *sum* and *carry* signals from inputs are $D(sum) = max\{D(y) + D_{s1}, D(z) + D_{s2}\}$, $D(carry) = \{D(z) + D_c\}$이 된다. [1]의 PPRT 알고리즘에서는 이와 같은 일반적인 FA 지연시간 모델을 사용하여 최적의 FA-트리 할당 알고리즘을 제안하였다.

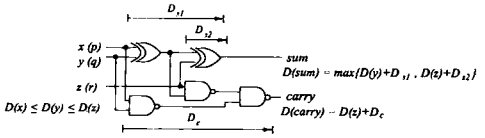


그림 2: FA의 지연시간 모델

3 배선 재배열을 통한 지연시간 최소화

본 단락의 목적은, 단어 단위의 CSA-트리의 구조를 유지시키면서 비트 단위의 배선 구조를 재조정 함으로써 CSA-트리의 지연시간을 좀더 향상시키는 것이다. 우리의 알고리즘은 두 단계의 최적화 과정으로 구성되어 있다: (단계 1) 배선을 고려한 *HA (half-adder) 합병* 으로써, 이 단계에서는 CSA-트리의 HA를 합하여 하나의 FA로 만들고, (단계 2) *FA 재조절 단계*에서는 FA의 입력을 재조절 한다.

단계 1 (HA-합병 기법): 할당된 CSA-트리에서, 어떤 CSA의 입력으로 사용되는 세 개의 일반적인 입력의 비트-넓이가 같지 않

거나[1], 아니면 어떤 입력이 상수 입력으로 주어진 경우(즉, logic-0혹은 1), CSA 내부의 어떤 FA는 HA로 대치된다. 예를 들어, 그림 3(b)는 그림 3(a)에서 보이는 단계 1에서 생성된 CSA-트리를 보이며, 이 경우 비트의 넓이는 일정하지 않다. 결과적으로, 두 개의 HA가 비트열 0에 생성되었다(점선으로 표시됨). 이 두 개의 HA는 하나의 FA로 그림 3(c)에서와 같이 합해질 수 있고, CSA-트리의 지연시간을 9에서 8로 줄인다.

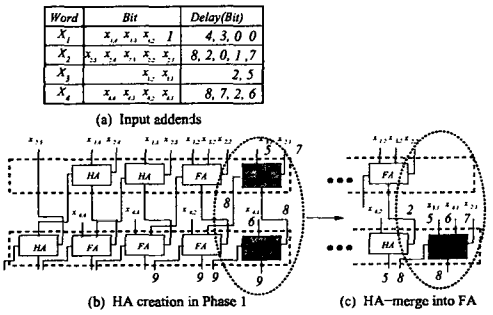


그림 3: CSA-트리 할당에 있어서 HA가 생성되는 것과, HA-합병을 보이는 예

각 비트열에 대해, 가장 왼쪽 비트열부터 오른쪽 비트열로 *HA-합병*을 수행한다. 하나의 열에서는, 위에서부터 아래로, 즉, leaf HA가 먼저 수행되고 root HA가 가장 마지막에 수행된다. 이러한 top-down HA-합병 과정에서, 만약에 HA의 sum 시그널이 다른 HA의 입력으로 주어진 경우, 즉, 밀접히 연결 되어 있는 경우에는, 우리는 두 개의 HA를 하나의 FA로 합병할 수 있다. 이는 그림 4(a)의 case 1에서 보여진다. 이때, 새로이 생성된 FA의 캐리 시그널을 잇는 방법에는 그림 4(a)에서 보이듯 두 가지 방법(i.e., trans1와 trans2)가 있다. 우리는 경우 1의 HA-병합에 대해 중요한 성질을 제시한다.

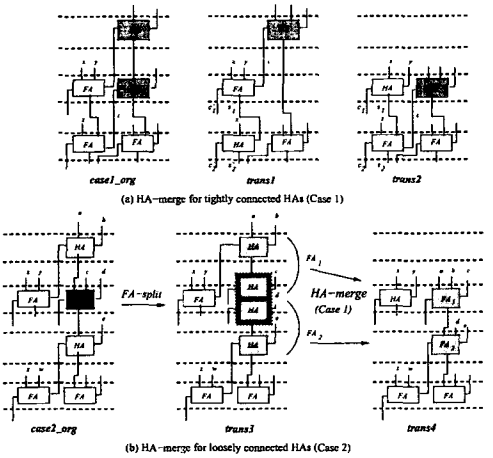


그림 4: 밀접히 연관되어 있는 HA의 병합(Case 1)과 느슨히 연결되어 있는 HA의 병합(Case 2)의 두가지 예

[1]이러한 경우는 곱셈이 shift와 덧셈 연산으로 분해된 경우에 많이 발생한다.

성질 1 그림 4(a)에서 보이는 바와 같이 HA-병합이 경우 1에 대해 수행되었을 경우, 일반적인 지연시간 모델이 HA에 사용되었을 경우에는(FA의 지연시간 모델과 비슷함) trans2*의 지연 시간은 항상* trans1*의 지연 시간보다 작거나 같다. 또한,* trans2*의 지연시간은* case1_org*의 지연시간보다 항상 빠르거나 같다.*

결과적으로, 성질 1에 따르면 우리가 밀접히 연관되어 있는 HA들에 대해(즉, 경우 1) HA-병합을 trans2의 형태로 적용하여 그림 4(a)에 보이는 FA를 생성한다. 반면에, 만약 두 개의 HA가 그림 4(b)의 case2_org에 보이듯이 밀접히 연관되어 있지 않은 경우, *경우 1처럼 HA를 병합하는 것은 전체적인 CSA-트리의 구조를 깨뜨리기 때문에 그대로 사용하지 않는다.* 대신, 우리는 FA를 나누어, 그 중 하나의 입력은 위쪽의 HA의 입력으로 할당하고, 두 개의 HA를 이어서, 위쪽 HA는 FA가 분해된 두 개의 HA 중 위쪽의 HA와 병합한다. 즉, 우리는 경우 2를 위한 HA-병합 과정을 다음과 같이 진행한다: (단계 1) *FA-분해* 경우 2를 경우 1로 바꾸는 단계이며, (단계 2) *HA-병합* 경우 1에서와 같이 진행된다. 그림 4(b)은 두 단계의 예를 보여주며, 우리의 HA-병합 과정에 대한 다음의 property를 제시한다.

성질 2 그림 4(b)에서 보이는 바와 같이 경우 2에 대한 HA-병합과정이 수행될 경우, 일반적인 HA 지연시간 모델이 사용될 때 trans4*의 지연시간은* case2_org*의 지연시간보다 항상 빠르다.*

성질 1와 2에 따르면, 제시된 HA-병합을 반복적으로 top CSA부터 bottom CSA로 순차적으로 진행할 때, 각 반복에서 CSA-트리의 지연시간을 지속적으로 감소시키는 것을 알 수 있다.

단계 2 (FA-재배열 기법): 이 과정에서는, 단계 1에서 생성된 전체적인 단어 단위의 배선 구조를 유지하며 비트-단위의 재배열로 지연시간을 향상시킨다. 단계 1에서는, CSA-트리 배치를 단어단위의 지연시간을 사용했었다. 즉, 만약 n비트 피연산자 $X = (x_{n-1}, x_{n-2} \cdots x_0)$에 대해, 단계 1은 $D(X)$로 $max\{D(x_{n-1}), D(x_{n-2}) \cdots D(x_0)\}$의 값을 사용한다.

우리는 이 단계에서 단어 단위의 CSA-트리의 구조를 보존하려 하기 때문에, 해당 CSA 블록의 *내부의* FA-단위의 배선을 재배열 한다. 예를 들어, 단계 1에 의해 생성된 CSA-트리의 부분이 그림 5(a)에 주어졌다. 여기서, 단어 단위의 CSA의 입력 Y_1, Y_2, Y_3의 지연시간은 3(=max{0,0,1,3}), 3(=max{2,3,2,2}), 5(=max{1,2,5,4})과 같이 각각 주어져 있다.[2] 결과적으로, Y_1, Y_2, Y_3의 지연시간은 CSA_i의 입력의 순서를 결정하고, 결국 이는 CSA_i의 모든 FA의 입력의 순서를 그림 5(b)에 보이듯 결정한다. 단계 2에서는, 그림 5(c)에 보이듯 각 FA의 입력을 재배열하여 FA들의 출력시간을 줄인다.

우리는 이러한 CSA-트리의 각 FA의 입력 재배열을 수행하는데 있어, leaf CSA의 FA에 대해 먼저 수행하고, root의 CSA에 대해 마지막으로 수행한다. 우리는 이러한 과정을 *FA-재배열*이라 부른다. 우리는 이러한 *FA-재배열*이 최적임을 다음의 정리에서 증명한다.

정리 CSA-트리의 비트 단계의 배선 재배열만이 허용되었을 때, FA-재배열에 의해 생성된 지연시간이 최적이다.

4 실험결과

우리는 *HA-병합*과 *FA-재배열*로 이루어진 일련의 배선을 고려한 합성 방법의 효율을 보이기 위해 실험을 수행하였다. [2]에 의해 생성된 최적지연시간의 CSA-트리의 구조를 유지하며 제안된 알고리즘을 이용하여 배선을 재배열하였다. 알고리즘은 C++로 작성되었으며, Pentium-3 500MHz Linux machine에서 측정되었다. 우리는 [3]의 여러 예제에 대해 우리의 알고리즘을 테스트하

[2]그림 5(b)에서
$D(Y_i)=max\{D(y_{i,3}), D(y_{i,2}), D(y_{i,1}), D(y_{i,0})\}, i = 1,2,3$

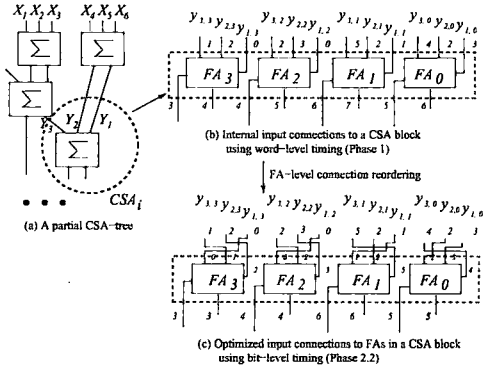
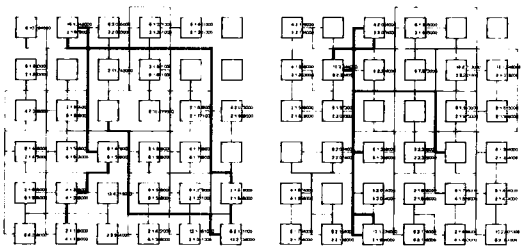


그림 5: FA-재배열의 예

였다. 그리고, 우리는 Stelling의 알고리즘에 의해 최적의 지연시간을 갖도록 생성된 결과와 우리의 결과를 비교하였다.[3]

우리의 알고리즘과 Stelling의 알고리즘을 여러 testcase에 적용한 후, 우리는 placemet를 Fiduccia and Mattheyses's partitioning 알고리즘[4]을 사용하여 적용한 후,[4] 라우팅을 위해[5] Lee's maze routing 알고리즘 [5]을 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 개수라 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. 우리는 모든 입력의 도달시간을 0으로 가정하였다. 결과에서, 우리는 우리의 알고리즘이 주요 경로의 지연시간을 평균적으로 22%를 감소시키는 것을 보였다. 더우기, 이러한 감소는 일관됨을 보였다. 즉, 합성의 초기단계에 배열을 고려하는 것이 마지막 회로의 지연시간을 상당히 향상시키는 것을 보였으며, 우리의 제시된 최종 배선을 고려한 합성 알고리즘이 잘 동작하는 것을 보였다.



(a) Stelling's [1] (b) Ours

그림 6: Exp.3에 대한 배선의 비교. (굵은 선은 주요 경로를 나타낸다..)

[3][1]의 알고리즘에 최적이라 함은 지연시간만을 고려한 결과이다.

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

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

Filter Design	α	Stelling's[1] crit. path delay/ long. net delay	Ours crit. path delay/ long. net delay	Improvement crit. path delay long. net delay
Lowpass	0.04	16.11 / 6.76	12.08 / 4.84	25% / 28%
	0.02	8.52 / 3.38	6.48 / 2.42	24% / 28%
	0.01	4.72 / 1.69	3.68 / 1.21	22% / 28%
Laplace	0.04	12.26 / 4.84	12.08 / 4.84	4% / 0%
	0.02	6.74 / 2.42	6.48 / 2.42	4% / 0%
	0.01	3.98 / 1.21	3.78 / 1.21	5% / 0%
Wavelet	0.04	46.43 / 17.64	18.22 / 9.00	61% / 49%
	0.02	23.79 / 8.82	9.59 / 4.50	60% / 49%
	0.01	12.48 / 11.86	5.14 / 2.25	59% / 49%
IIR	0.04	21.38 / 14.44	20.02 / 6.76	6% / 53%
	0.02	11.23 / 7.22	10.50 / 3.38	6% / 53%
	0.01	6.15 / 3.61	5.74 / 1.69	7% / 53%
Complex	0.04	18.68 / 11.56	16.53 / 11.56	12% / 0%
	0.02	9.99 / 5.78	8.69 / 5.78	6% / 0%
	0.01	5.61 / 2.89	4.77 / 2.89	7% / 0%
Average	0.04	22.97 / 11.05	15.79 / 7.40	22% / 26%
	0.02	10.05 / 6.13	8.26/3.70	21% / 26%
	0.01	5.49 / 4.60	5.49 / 4.60	21% / 26%

표 1: 벤치마크 디자인에 대한 결과.

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 두 단계의 접근 방법, *CSA 회로를 HA-병합과 FA-입력 재배열을 통해 최종 배선을 고려하며 지연시간을 향상시키는 알고리즘* 이 연산식의 데이터 경로를 최종 배선을 고려하여 합성하는데 있어서 효율적으로 사용될 것이라 생각한다. 여러 가지 bnnchmark filter design에 대해, 우리는 최적의 비트 단위의 지연시간을 갖는 [1]의 방법에 비해 배선의 복잡도를 줄이는 데 있어서 큰 역할을 하는 것을 볼 수 있었으며, 결국 주요 경로의 지연시간을 짧게 하는 것을 보였다.

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

참조 서적

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

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

[3] N. D. Dutt, "High-Level Synthesis Design Repositories", *http://www.ics.uci.edu/ dutt.*

[4] C. Fiduccia, R. Matteyses, "A Linear-time Heuristic for Improving Network Partitions", *Proc. of ACM/IEEE Design Automation Conference*, 1982.

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

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