저전력 소모를 위한 상위 수준의 효과적인 바인딩 알고리즘

최윤서 김태환
한국과학기술원 전산학과
yschoi@vlsisyn.kaist.ac.kr tkim@cs.kaist.ac.kr

An Efficient Low-Power Binding Algorithm in High-Level Synthesis

Yoonseo Choi Taewhan Kim
Dept. of EECS, Korea Advanced Institute of Science and Technology

요 약

우리는 저전력 소모를 위한 상위 수준(high-level)에서의 효과적인 바인딩(binding) 알고리즘을 제안한다. 이전 연구들에 의해서 저전력 소모를 위한 몇몇의 바인딩 알고리즘들은 멀티-코모도티 플로우(multi-commodity flow) 문제로 모델링 될 수 있음이 밝혀졌다. 그러나 멀티-코모도티 플로우 문제는 NP-hard이기 때문에 작은 크기의 설계에만 적용될 수 있다. 이러한 제약을 극복하기 위해 우리는 네트워크 상의 플로우를 잘 이용해서 효과적으로 빠른 시간 안에 최적에 가까운 결과를 낼 수 있는 방법을 제안하여 크기가 큰 설계에도 적용할 수 있도록 한다. 이를 위해 우리는 첫번째 단계에서는 네트워크에서 최소 비용의 최대 플로우 (maximum flow - minimum cost) 를 구하는 방법을 부분적으로 이용해서 유효한 결과를 구하고 두 번째 단계에서는 이를 반복적으로 개선시켜나가는 2 단계의 알고리즘을 제안한다. 벤치마크를 이용한 실험 결과는 제안된 알고리즘이 실제적인 설계에 적용되었을 때, 충분히 빠른 시간 안에 최적에 가까운 결과를 생성함을 보여준다.

1 소개

노트북, 무선 통신 기기 등의 휴대용, 고밀도의 마이크로(micro) 장치들의 출현과 함께 VLSI(very large scale integrated) 회로의 전력소모는 아주 중요한 문제가 되었다. 따라서 전력 소모를 모델링하고 측정하며 최적화 하는 것은, 상위 수준에서부터 게이트(gate)와 레이아웃(layout) 단계에 이르기까지 회로 제작의 전 과정에서 고려되어야 한다. 이 논문은 상위 수준에서 전력 소모를 최소화 하기위한 연구의 범주에 속한다.

저전력 소모를 위한 하드웨어의 바인딩에 관한 연구들이 많이 있었다. Dasgupta와 Karri[1]는 데이터 버스 상에서의 전력 소모를 최소화하기위한 스케쥴링과 바인딩 알고리즘을 제안하였다. 그들의 알고리즘은 시뮬레이티드 어닐링(simulated annealing)에 기초하고 있다. Hong과 Kim[2]은 최소 비용의 최대 플로우 계산을 반복적으로 수행함으로써 데이터 버스에서의 전력소모를 최소화 하는 스케쥴링과 바인딩의 통합알고리즘을 제안하였다. Chang과 Pedram[3]은 전력소모를 줄이기 위한 레지스터 할당과 바인딩 알고리즘을 제안하였다. 그들은 그 문제를 최소 비용의 클리크 커버링(clique covering) 문제로 모델링하고 최대 비용 플로우 알고리즘을 이용하여 최적의 해를 구했다. 그들은 또한 저전력 소모를 위한 기능 단위(functional unit)를 바인딩 하는 방법을 제안하였다[4]. 그 문제의 최적의 해를 구하는 방법으로 최소 비용의 멀티-코모도티 문제로 모델링 되었다. 그러나 멀티-코모도티 문제는 NP-hard하기 때문에 그들은 이 문제를 작은 크기의 짧은 수행 주기를 가지는 파이프라인(pipeline) 디자인에만 적용하였다. [3, 4]의 방식들이 전력 소모를 최소화 하기위한 최적의 해를 구하기는 하지만, 이들은 작은 크기의 특정한 디자인에만 적용될 수 있다는 제약이 있다. 따라서 우리는 네트워크에서 최소 비용의 최대 플로우의 특성을 잘 이용해서, 실제적인 크기의 디자인에도 적용가능하면서 최적에 가까운 결과를 내는 방식을 제안한다.

우리는 BIND-lp라고 불리는 첫 단계에서 유효한 결과를 내고, 두 번째 단계에서 이를 점차적으로 개선시키는 알고리즘을 제안한다. 이 논문에서 우리는 저전력 소모를 위한 기능 단위를 바인딩 하는 방식만 설명한다. 그러나 우리의 알고리즘은 약간의 수정만 가하면 다른 상위 수준의 최적화 문제(예: 레지스터, 메모리 포트, 버스 바인딩)에도 적용될 수 있음이 자명하다.

2 서문

상위 수준의 합성 단계에서 하나의 기능 단위(functional unit,FU)에 의해 소모되는 전력을 빠르게 측정하기 위해서, 하나의 FU에 의해 소모되는 전력은 선택된 입력과 출력의 통계로 모델링 된다. 우리는 [3, 4]와 유사한 모델링을 이용한다. 즉, 하나의 FU에 의한 소모 전력은 그 FU의 입력 비트(bit)들의 스위칭 액티비티들(switching activities)에 의해 정해진다. 입력 데이터의 정확한 패턴들은 대부분 정해져 있지 않으므로, 확률적인 모델을 이용한다.

$SW(x, y)$는 한 FU에서 오퍼레이션 x와 y가 차례로 수행될 때의 입력 단자에서의 비트들의 평균적인 천이(transition) 회수, 즉 해밍 디스턴스(hamming distance) 를 나타낸다. 모든 오퍼레이션 쌍의 $SW(\cdot)$ 는 반복적으로 CDFG를 시뮬레이션 함으로써 구해진다. 그림 1(a)는 스케쥴이 된 데이터 플로우 그래프의 예이다. 오퍼레이션 a, b, c는 첫 번째 수행 단계에, d, e는 두 번째 수행 단계에, f, g, h는 세 번째 수행 단계에 수행된

다. 표 1은 그림 1(a)의 오퍼레이션들의 $SW(\cdot)$의 값의 예이다. $SW(a,b) = 7.99$는 16(8개의 오른쪽 입력과 8개의 왼쪽 입력) 개의 비트들 중에 평균 7.99개의 비트들이 천이한다는 것을 나타낸다. $SW^{fu_i}(x,y)$는 fu_i상에서 x와 y가 차례로 수행되었을 때 입력 단자에서 일어나는 비트 천이 수의 기대값을 나타낸다고 하자. 이 때 우리가 풀고자 하는 문제는 다음의 값이 최소가 되도록 CDFG의 오퍼레이션들을 FU들에 바인딩하는 것이다.

$$SW_{tot} = \sum_{\forall fu_i \ of \ functional \ units} SW^{fu_i}. \quad (1)$$

그림 1(b)는 CDFG의 스케줄된 오퍼레이션들을 세 개의 기능 단위, fu_1, fu_2, fu_3에 바인딩한 한 예를 나타낸다. 오퍼레이션 a, d, f는 fu_1에서 각각 수행 단계 1, 2, 3에서 , b, e, g는 fu_2에서 각각 수행 단계 1, 2, 3에, c, h는 fu_3에서 각각 1,3에 수행된다. 이러한 순서로 오퍼레이션들이 반복적으로 수행된다. 따라서, 표 1에서 SW^{tot}은 $SW(a,d) + SW(d,f) + SW(f,a) + SW(b,e) + SW(e,g) + SW(g,b) + SW(c,h) + SW(h,c)$ 로서 $5.75 + 13.67 + \cdots + 6.44 = 52.15$이다.

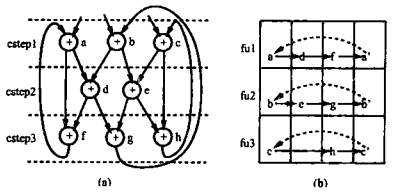

Figure 1: (a) 스케줄된 CDFG, (b) 스케줄(a)에 대한 바인딩.

	a	b	c	d	e	f	g	h
a	0.00	7.99	1.32	5.79	4.33	9.17	6.24	5.14
b	7.99	0.00	7.32	7.33	4.77	9.76	3.96	11.23
c	1.32	7.32	0.00	4.48	4.42	9.97	6.25	6.44
d	5.79	7.33	4.48	0.00	7.35	13.67	8.58	10.92
e	4.33	4.77	4.42	7.35	0.00	6.40	1.91	6.47
f	9.17	9.76	9.97	13.67	6.40	0.00	5.93	6.80
g	6.24	3.96	6.25	8.58	1.91	5.93	0.00	7.86
h	5.14	11.23	6.44	10.92	6.47	6.80	7.86	0.00

Table 1: 그림1(a)의 CDFG의 $SW(\cdot)$ 값.

3 저전력 소모를 위한 바인딩 알고리즘

3.1 플로우 문제를 위한 네트워크 모델링

우리의 알고리즘을 설명하기 위해, [4]의 네트워크 구성을 간단히 살펴본다. 그림 2는 그림 1(a)의 스케줄에 해당하는 네트워크 $G(V,A)$를 나타낸다. CDFG의 각 오퍼레이션은 그림 2에서 점선으로 둘러싸인 한 쌍의 노드로 표현된다. 노드들은 수행되는 단계에 따라서 세로로 배열된다. 스케줄상에서 연속적으로 수행될 수 있는 두 노드들 간에는 아크(arc)가 존재하게 된다. 예를 들어 그림 1에서 오퍼레이션 a는 실행 단계 1에, d는 실행 단계 2에 수행되는 것으로 스케줄 되어있다. 따라서 그림 2에서는 a로부터 d로의 아크가 존재한다. 또한 서로 다른 실행 단계 i 와 j에서($i < j$) 수행되는 두개의 오퍼레이션 사이에도, 만약 실행 단계 $k(= i+1, i+2, \cdots, j-1)$ 에서 오퍼레이션이 수행되지 않는 FU가 존재한다면, i로부터 j로의 아크가 존재한다. 예를 들어 실행단계 1의 a로부터 실행단계 3의 f로의 아크가 존재하는 데, 이는 실행단계 2에서 두개의 오퍼레이션만이 수행되므로 3개의 FU 중 하나는 사용되지 않기 때문이다. G의 모든 아크는 비용(cost)를 가지고 있다. 자세한 것은 [4, 2]에 나와 있다. 모든 아크의 용량(capacity)는 1이다.

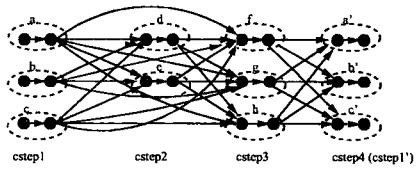

Figure 2: 그림 1의 스케줄된 오퍼레이션들에 해당하는 네트워크.

3.2 바인딩 알고리즘

SW_{tot}를 최소화 하는 바인딩은 네트워크 상의 모든 노드들을 커버하는 최소 비용의 서로 나뉘어진 패스(path)들을 구하면 된다(예, 그림 2에서 G의 $a \rightarrow \cdots \rightarrow a'$, $b \rightarrow \cdots \rightarrow b'$, $c \rightarrow \cdots \rightarrow c'$). 이는 멀티-코모도티 문제로서, NP-hard가 되는 이유는 다음의 조건에서 비롯된다.

(시작-끝 제한조건) *문제의 해의 각각의 플로우 패스들의 시작하는 노드와 끝나는 노드는 서로 동일해야 한다.*

위 조건이 없다면, [5]의 minimum cost path augmentation 알고리즘을 G에 적용함으로써 주어진 플로우 문제의 최적의 해를 효과적으로 구할 수 있다. 따라서 원래의 멀티-코모디티 문제를 두개의 하위 문제로 나누어서 두 단계로 푸는 것이 우리의 핵심 착상이다. 우리의 알고리즘이 최적의 해를 구하는 것은 아니지만, 최적에 가까운 해를 구하며, 큰 크기의 실제적인 디자인에 적용될 수 있고, 빠른 수행으로 여러 가지 가능한 스케줄에 바인딩을 적용해 볼 수 있다는 장점이 있다.

Step 1 (*유효한 초기 해를 구하기*) 먼저 minimum cost path augmentation 알고리즘[5]은 시작 노드에서 끝 노드로의 최소 비용 패스 p를 찾고, [1] 만약 패스가 찾아지지 않으면 끝내고(*old-1*), 패스 p에 속한 모든 아크들의 방향을 반대로 하고 1의 용량을 가지도록 하는(*old-2*), 두 단계를 반복함으로써 구해진다. (우리의 알고리즘은 여기에 약간의 수정을 가하여 구해진다.) 그림 3(a)-(c)는 path augmentation알고리즘의 각 수행 단계에서 생성된 augmenting path 를, 그림 3(d)는 구해진 해의 3개의 플로우 패스를 나타낸다. 즉, 패스 $a \rightarrow h \rightarrow a'$ 는 하나의 FU에 실행 단계 1에서는 a가, 실행 단계 2에서는 h가 실행 단계 4에서는 a가 바인딩됨을 나타낸다. 그러나 $b \rightarrow e \rightarrow f \rightarrow c'$ 과 $c \rightarrow d \rightarrow g \rightarrow b'$ 바인딩은 시작 노드와 끝 노드가 동일하지 않음으로 유효하지 않은 패스들이다. 이러한 유효하지 않은 패스들이 생성되는 이유는 (*old-2*)에서 용량이 1인 반대 방향의 아크들을 만들어 주어서 다음 번 수행에서 패스를 찾을 때 기존에 구해진 패스가 바뀔 수 있기 때문이다. 따라서 우리는 아크의 방향을 반대로 만들 때 그 아크의 용량을 0 으로 만들어 줌으로써 유효하지 않은 패스가 생성되는 것을 막아준다.

우리의 알고리즘의 첫번째 단계는 원래의 path augmentation알고리즘을 수정하여 다음의 두 과정을 반복함으로써 유효한 해를 구한다. (*new-1*)각각의 시작 노드에서 그와 동일한 끝 노드로의 최소 비용의 패스를 구한다. 만약 패스가 구해지지 않으면 끝내고, 그렇지 않으면 구해진 패스들 중에 최소의 비용을 가지는 패스를 선택한다; (*new-2*) 위에서 구해진 패스에 속한 아크들의 방향을 반대로 하고 비용을 0으로 만들어서 G를 갱신한다.

그림 4는 그림 2의 스케줄에 우리의 알고리즘을 적용한 것

[1] p는 *augmenting path*라고 불린다.

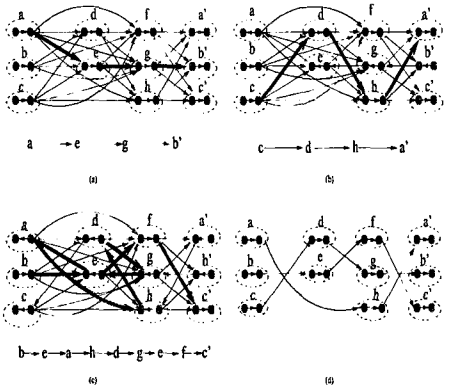

Figure 3: 그림2(a)의 스케줄에 대한 min-cost path augmentation 알고리즘의 예

이다. 처음 G에서 *시작-끝 제한조건*을 만족하는 최소 비용의 패스들인 $a \rightarrow \cdots \rightarrow a$, $b \rightarrow \cdots \rightarrow b$, $c \rightarrow \cdots \rightarrow c$를 구한다. 이중에서 가장 적은 비용을 가지는 패스인 $b \rightarrow \cdots \rightarrow b$가 하나의 바인딩으로 선택되고, 네트워크는 그림 4(b)와 같이 갱신된다. 이 네트워크에서 두 개의 유효한 최소 비용 패스인 $a \rightarrow \cdots \rightarrow a$와 $c \rightarrow \cdots \rightarrow c$를 구한다. $c \rightarrow \cdots \rightarrow c$가 $a \rightarrow \cdots \rightarrow a$보다 적은 비용을 가지므로, $c \rightarrow \cdots \rightarrow c$가 또 다른 패스로 선택된다. 마지막으로 a로부터 a'로의 패스가 구해진다 (그림4(c)). 그림 4(d) 이 마지막 바인딩 결과이다.

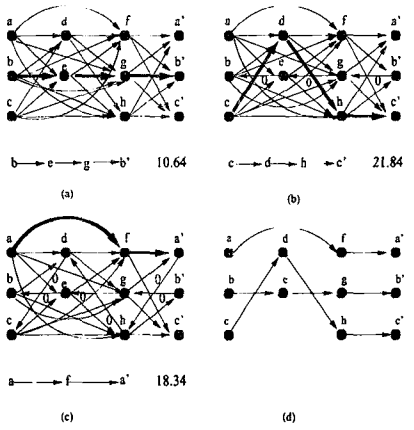

Figure 4: 그림 2의 스케줄에 제안된 알고리즘의 Step 1을 적용한 예.

Step 2 (*초기 바인딩의 개선*) 이 단계에서는 *시작-끝 제한조건*을 만족시키면서, 초기에 구해진 바인딩을 개선시켜 나간다. 두 개의 플로우 패스가 있을 때, 두 패스의 아크들이 실행 단계 i와 $i+1$사이에서 서로 바뀌었을 때 발생하는 SW_{tot}의 차이를 $switch_cost(i)$로 정의한다. 그림 5(a)의 두 개의 플로우 패스에서 $switch_cost(1) = 2.11$는, 아크 $a \rightarrow f$과 $c \rightarrow d$가 서로 바뀜으로써 SW_{tot}가 2.11만큼 증가한다는 것을 나타낸다.
모든 플로우 패스 쌍에 대해서 $switch_cost(\cdot)$을 구한 다음 짝수 쌍의 아크를 교환함으로써 $switch_cost(\cdot)$의 합, 즉

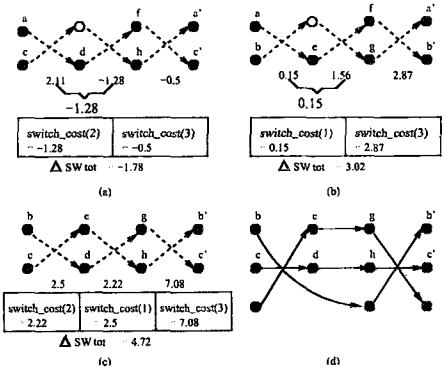

Figure 5: 그림 4의 바인딩에 Step 2를 적용하는 예.

ΔSW_{tot}을 최소로 증가시키는 플로우 패스 쌍을 골라 아크들을 교환시킨다. $switch_cost(\cdot)$가 더 이상 개선되지 않을 때 까지 이 과정을 반복한다. 그림 5(a)-(c)는 각각의 플로우 패스에 대한 ΔSW_{tot}를 나타낸다. 그 중 최소인 (a)가 선택되어 결과적으로 (d)가 구해진다.

4 실험 결과

제안된 알고리즘은 C++로 Sun Sparc20에서 구현되었다. 상위 수준 합성의 벤치마크를 이용하여 FU 바인딩을 수행한 결과는 표 2과 같다. 수치는 모두 SW_{tot}를 나타낸다. *BIND-opt*는 멀티-코모디티를 이용하여 최적의 해를 구한 것이다. 메모리 부족으로 인하여 몇몇의 결과를 구할 수 없었다(- 표시). 반면에 제안된 알고리즘인 *BIND-lp*는 1초 이내에 최적의 결과와 크게 다르지 않은 좋은 결과를 생성해 낸다.

	total transitions		
design	*BIND-opt [4]*	*BIND_lp*	diff.(%)
COMPLEX	947.6	947.6	0.00
KALMAN	549.8	549.8	0.00
KALMAN.2	819.8	820.6	0.09
DIFF	1191.2	1191.2	0.00
DIFF.2	2371.3	2542.4	7.20
IDCT	1510.9	1513.8	0.19
IDCT.2	1413.4	1418.5	0.36
EWF	-	(1697.9)	-
EWF.2	-	(3787.5)	-
average			1.12

Table 2: 상위 수준 합성 벤치마크를 이용한 바인딩 결과.

References

[1] A. Dasgupta and R. Karri, "High-Reliability, Low-Energy Microarchitecture Synthesis," *TCAD*, 1998.

[2] S. Hong and T. Kim, "Bus Optimization for Low Power Data Path Synthesis based on Network Flow Method," *ICCAD*, 2000.

[3] J.-M. Chang and M. Pedram, "Register Allocation and Binding for Low Power," *DAC*, 1995.

[4] J.-M. Chang and M. Pedram, "Module Assignment for Low Power," *EDAC*, 1996.

[5] R. E. Tarjan, *Data Structures and Network Algorithms*, Society for Industrial and Applied Mathematics, 1983.