논문 2007-44SD-12-4

처리량 기반 평면계획을 위한 처리량 계산 방법

(A Throughput Computation Method for Throughput Driven Floorplan)

강 민 성*, 임 종 석**

(Min-Sung Kang and Chong-Suck Rim)

요 약

반도체 공정 기술의 발전으로 인한 개략 배선 지연시간의 증가는 고성능 시스템의 설계를 어렵게 하고 있다. 이 문제를 해결하기 위해 배선에 파이프라인 요소를 삽입하는 방법이 있으나 시스템의 타이밍을 변화시켜 시스템의 기능성을 보장할 수 없다. LIP(Latency Insensitive Protocol)는 임의의 파이프라인 요소의 삽입에 대해 기능성을 보장하지만 처리량이 저하된다. 처리량 저하를 줄이기 위해서는 평면계획 단계에서 처리량을 고려하여 블록을 배치하여야 한다. 이러한 평면계획을 가능하게 하기 위해서 새로운 처리량 계산 방법을 제안하고 평면계획의 비용함수에 적용하였다. 실험 결과, 기존의 휴리스틱 처리량 평가 방법을 적용한 평면계획에 비해 처리량이 평균 16.97% 향상되었다.

Abstract

As VLSI technology scales to nano-meter order, relatively increasing global wire-delay has added complexity to system design. Global wire-delay could be reduced by inserting pipeline-elements onto wire but it should be coupled with LIP(Latency Intensive Protocol) to have correct system timing. This combination however, drops the throughput although it ensures system functionality. In this paper, we propose a computation method useful for minimizing throughput deterioration when pipeline-elements are inserted to reduce global wire-delay. We apply this method while placing blocks in the floorplanning stage. When the necessary for this computation is reflected on the floorplanning cost function, the throughput increases by 16.97% on the average when compared with the floorplanning that uses the conventional heuristic throughput-evaluation-method.

Keywords: LIP, wire delay, throughput, floorplan, cost function

Ⅰ. 서 론

VLSI 공정 기술의 발전에 따라 집적도와 시스템의 동작 주파수 증가로 인해 개략 배선(global wire)의 지연시간(delay)은 상대적으로 증가하고 있다. 이는 시스템 성능을 결정하는 주요한 요소로 대두되고 있다[1].

이러한 개략 배선 지연시간 문제는 배선(wire)에 파이프라인 요소(pipeline-element)를 삽입하여 해결할 수 있으나 시스템의 타이밍을 변경시켜 시스템의 기능성(functionality)을 보장 할 수는 없다. 이에 임의의 파이프라인 요소들의 삽입에 대해 기능성을 보장하는 LIP(Latency Insensitive Protocol)[2]가 제안 되었지만, LIP를 적용한 시스템은 사이클(cycle)에 파이프라인 요소가 삽입되면 처리량(throughput) 저하의 문제가 발생할 수 있다.

처리량 저하는 파이프라인 요소의 삽입을 최소화하여 줄일 수 있다. 이를 위해 시스템에서 사이클을 구성하는 블록을 가까이 배치해야 한다. 이러한 배치를 위해서는 평면계획 단계에서 처리량을 평가할 수 있는 비용함수(cost function)가 필요하다. 기존의 정확한 처리량 계산 방법은 시간 복잡도가 높아 비용함수로 사용할 수 없다[3]. 이를 개선한 휴리스틱 처리량 평가 방법은 처리량을 정확히 계산하지 못하는 문제를 갖는다[4].

이에 본 논문은 처리량 저하를 줄일 수 있는 새로운 처리량 계산 방법을 제안하고 이를 평면계획의 비용함

* 정회원, 서강대학교 컴퓨터공학과,
(주) 하이닉스 반도체 선행설계팀
(Department of Computer Science and Engineering, Sogang University; Advanced Design Team, HYNIX Co., Ltd)
** 평생회원, 서강대학교 컴퓨터공학과
(Department of Computer Science and Engineering, Sogang University)
접수일자: 2007년5월3일, 수정완료일: 2007년11월23일

수에 포함시켰다. 실험 결과, 제안한 방법을 적용한 평면계획은 기존의 휴리스틱 처리량 평가 방법을 적용한 평면계획보다 처리량이 평균 16.97% 향상되었다.

본 논문의 구성은 다음과 같다. 먼저 Ⅱ장에서는 LIP와 처리량에 관련된 기존의 방법들에 대하여 살펴본다. Ⅲ장에서는 제안한 처리량 계산 방법을 설명한다. Ⅳ장에서는 기존의 평면계획과 제안한 평면계획의 결과를 비교한다. 마지막으로 Ⅴ장에서는 결론을 내린다.

Ⅱ. 기본정리

1. LIP(Latency Insensitive Protocol)

LIP는 모든 블록이 임의의 시간동안 블록 내부의 상태 변화 없이 블록의 동작을 정지시킬 수 있는 stalling capacity를 가지기 때문에 파이프라인 요소들의 삽입으로 인한 타이밍 변경에 대해 시스템의 기능성을 보장하는 프로토콜이다[2].

LIP는 event와 stop 신호로 동작한다. event 신호는 블록의 입력이 유효한 경우의 informative event와 그렇지 않은 경우의 stall event로 나뉜다. stop 신호는 블록이 stall된 경우에 앞 단의 블록의 동작을 정지시켜 informative event의 전송을 방지한다. 블록의 모든 입력이 informative event인 경우에만 informative event를 출력하고 블록과 파이프라인 요소의 초기 값은 각각 informative event와 stall event이다.

LIS(Latency Insensitive System)는 LIP를 적용한 시스템이다. LIS의 처리량 T_L은 informative event를 이용하여 식 (1)과 같이 정의할 수 있다.

$$T_L = \frac{\#\ informative\ event}{total\ time} \tag{1}$$

처리량 T_L은 LIS에 존재하는 사이클에 파이프라인 요소가 삽입되면 저하될 수 있다. 그림 1에서 보듯이 사이클의 특성상 파이프라인 요소가 삽입되면 stall event는 사이클에 계속 존재한다. 모든 블록의 입력은 3클록 마다 stall event이므로 처리량은 2/3로 저하된다.

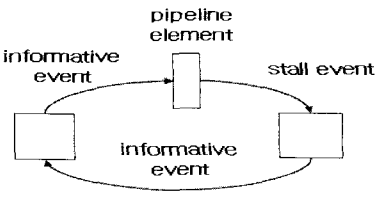


그림 1. 사이클에서의 처리량 저하
Fig. 1. Throughput degradation at a cycle.

2. 정확한 처리량 계산 방법

lis-graph는 LIS의 그래프 표현이다[3]. lis-graph의 정점(vertex)은 블록, 호(edge)는 블록 간의 배선, 호의 가중치(weight)는 배선에 삽입된 파이프라인 요소의 개수이다.

lis-graph G가 주어졌을 때 처리량 $\vartheta(G)$는 식 (2)을 수행하여 구할 수 있다.

$$\vartheta(G) = \min_{\forall C \in G} \left(\frac{|C|}{w(C)+|C|} \right) \tag{2}$$

C는 G에 존재하는 사이클이고 $|C|$는 C의 크기로 정점의 수를 의미한다. $w(C)$는 C에 존재하는 호의 가중치 합이다. 처리량은 시스템에 존재하는 모든 사이클 중 식 (1)을 만족하는 사이클에 의해 결정된다. 이러한 사이클을 L_c라 한다. 반면에 식 (1)의 시간 복잡도는 $O(EV^2)$로 시스템에 존재하는 모든 사이클에 대한 $w(C)$와 $|C|$을 알아야 하기 때문에 계산 시간이 오래 걸려 평면계획의 비용함수로 사용하기에 부적절하다[4]. E와 V는 각각 lis-graph의 호와 노드의 개수이다.

3. 기존의 처리량 저하 해결 방법

가. 처리량을 고려한 평면계획

평면계획은 직사각형의 칩 상에 각 블록을 적절히 배치하는 것이다. 개략 배선 지연시간은 블록 간의 위치에 따라 대부분 결정되기 때문에 처리량 저하를 줄이기

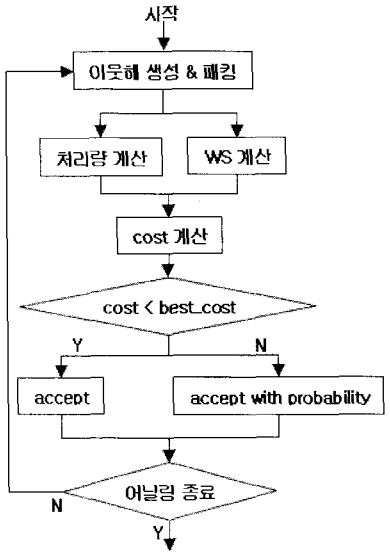


그림 2. 처리량이 고려된 평면계획의 흐름도
Fig. 2. The flow of throughput-aware floorplan.

위해서는 처리량을 고려한 평면계획이 필요하다.

그림 2는 처리량이 고려된 평면계획을 보이고 있다. 초기 해에서 시작하여 이웃 해를 생성하고 패킹을 통해 각 블록의 위치를 구한다. 그 위치를 통해 이웃 해에 대한 처리량과 WS(White Space)를 계산한다. WS는 작고 처리량은 크면 좋으므로 처리량은 (1-처리량)로 바꾼다. 계산한 WS와 처리량에 각각 가중치를 곱하고 이들을 더하여 하나의 cost로 만든다. 이 cost가 best_cost보다 작으면 이웃 해를 받아들이고 그렇지 않으면 확률적으로 받아들인다. 이 과정을 어닐링이 종료할 때까지 반복하여 WS와 처리량이 동시에 최적화된 평면계획 결과를 얻을 수 있다.

나. 휴리스틱 처리량 평가 방법

휴리스틱 처리량 평가 방법은 식 (2)의 시간 복잡도를 개선한 것으로 최근에 Casu가 제안 하였다[4]. 휴리스틱 처리량은 다음과 같은 과정을 통해 평가할 수 있다.

1. 하나 이상의 사이클에 포함된 배선은 가장 작은 사이클 크기의 역수를 가중치로 준다.
2. 각 배선의 Manhattan distance를 계산한다. 이를 한 클록에 도달할 수 있는 최대 거리로 나누어 필요한 파이프라인 요소의 개수를 계산한다.
3. 1과 2에서 구한 각 배선의 가중치와 파이프라인 개수를 곱한다.
4. 3에서 구한 모든 값을 더하여 처리량을 예측한다.

이 평가 방법은 사이클의 크기가 작을수록 같은 수의 파이프라인 요소 삽입에 대해 처리량 저하가 크다는 특성을 이용한 것으로 4에서 구한 값이 작을수록 좋은 결과를 얻을 수 있다. 반면에 이는 정확한 처리량을 평면계획에 반영하지 못할 수도 있다.

예를 들어 그림 3는 IP 블록의 배치에 따라 파이프라인 요소가 삽입된 두 가지 결과를 보이고 있다. 이 시스템은 IP_1과 IP_2로 이뤄진 사이클 C_1과 IP_1, IP_2, IP_3으로 이뤄진 사이클 C_2를 가지고 있다. 식 (2)에 의해 정확한 처리량을 구하면 그림 3(a)의 경우는 $|C_1|=2$, $w(C_1)=2$, $|C_2|=3$, $w(C_2)=3$이므로 $\min(2/(2+2), 3/(3+3))=1/2$이고 그림 3(b)의 경우는 $|C_1|=2$, $w(C_1)=1$, $|C_2|=3$, $w(C_2)=4$이므로 $\min(2/(1+2), 3/(4+3))=3/7$이 되어 그림 3(a)가 더 좋은 처리량을 갖는다. 반면에 휴리스틱 처리량 평가 방법을 적용하면 C_1에 속한 배선의 가중치 (IP_1, IP_2)=1/2, (IP_2,IP_1)=1/2와 C_2에 속한 배선의 가중치 (IP_2, IP_3)=1/3, (IP_3,IP_1)=1/3에 의해 그림 3(a)의 처리량 평가는 (1/2*1)+(1/2*1)+(1/3*1)+(1/3*1)=5/3이고 그림 3(b)의 처리량 평가는 (1/2*1)+(1/2*0)+(1/3*2)+(1/3*1)=3/2이 되어 그림 3(b)가 더 좋은 처리량을 갖는 것으로 평가한다. 이러한 잘못된 평가는 평면계획 단계에서 처리량 저하의 원인이 된다.

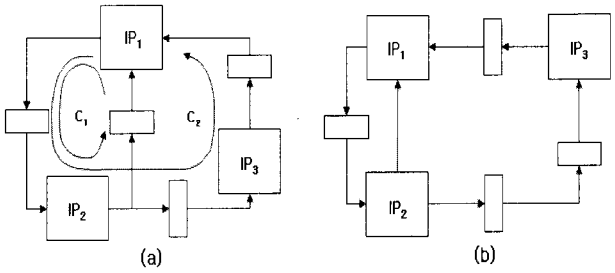


그림 3. IP블록의 배치에 따른 파이프라인 요소의 삽입
Fig. 3. IP-block-dependent pipeline-element insertion.

III. 출력 시간 기반 처리량 계산 방법

본 장에서는 기존의 휴리스틱 처리량 평가 방법의 처리량 저하를 개선하기 위해서 주어진 lis-graph G를 SCC(Strongly Connected Component)[5]로 분할하고 각 SCC에 대해 블록의 출력 시간을 기반으로 처리량을 계산하는 새로운 방법을 제안한다.

1. SCC를 이용한 lis-graph의 분할

SCC는 임의의 그래프 H에 포함된 서로 다른 노드 v_i, v_j에 대하여 v_i에서 v_j로의 방향 경로가 존재하고 또한 v_j에서 v_i로의 방향 경로가 존재하는 H의 부 그래프이다[5]. 즉, H에 존재하는 모든 사이클은 H에 속하는 임의의 SCC에 속한다.

임의의 lis-graph G의 처리량은 식(2)을 만족하는 사이클에 의해서 결정되고 모든 사이클은 G의 임의의 SCC에 속한다. 이에 G를 SCC로 분할한다. 분할된 각각의 SCC에 대해서 식 (2)을 만족하는 값을 구하고 이들의 최소를 선택하여 정확한 처리량을 계산할 수 있다. G의 SCC 분할과 이 분할에 의해서 사이클이 아닌 부분이 제거되므로 정확한 처리량 계산 방법에 비해서 계산 시간을 단축할 수 있다.

2. 출력 시간 테이블과 출력 시간 간격 테이블

임의의 SCC에 속한 블록의 집합을 B_{scc}라고 하자. B_{scc}에 속하는 임의의 블록 b는 이 블록의 모든 입력이

informative event일 때 informative event를 출력한다. 이에 따라 블록 b의 i번째 informative event 출력 시간은 가장 마지막 informative event의 입력 시간에 의해 결정된다. 즉, B_{scc}에 속하는 임의의 블록 b의 출력 시간 함수 $t_b(i)$는 식 (3)을 통하여 구할 수 있다.

$$t_b(i)=\begin{cases}1, & \text{if } i=1.\\ \max(t_v(i-1)+w(v,b)+1), & \text{if } i>1.\\ where,\ v\neq b\in B_{scc}\end{cases} \quad (3)$$

v는 블록 b의 입력에 연결된 블록들이고 $w(v,b)$는 호 (v,b)의 가중치로 파이프라인 요소에 의한 지연시간을 의미한다. 각 블록의 지연 시간은 1로 가정한다.

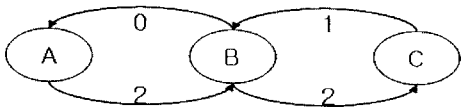
출력 시간 테이블은 B_{scc}에 속하는 임의의 블록 b의 $t_b(i)$을 기록한 테이블로 정의한다. i는 출력 단계이다.

출력 시간 간격 $d_b(i)$은 i번째와 $i-1$번째 출력 시간의 차로 식 (4)과 같이 구한다.

$$d_b(i)=\begin{cases}0, & \text{if } i=1.\\ t_b(i)-t_b(i-1), & otherwise.\end{cases} \quad (4)$$

출력 시간 간격 테이블은 B_{scc}에 속하는 임의의 블록 b의 $d_b(i)$을 기록한 테이블로 정의한다. 두 테이블의 행과 열은 각각 B_{scc}에 속하는 블록과 출력 순서를 의미한다. B_{scc}에 속하는 모든 블록의 출력 시간은 정리 1을 만족한다.

정리 1. B_{scc}에 속하는 임의의 블록 b의 출력 시간이 L_c에 속하는 블록 c에 의해 결정되는 출력 단계 i가 존재한다. 단, $i>1$이다.



(a) lis-graph G의 SCC

출력	1	2	3	4	5	6	7
A	1	2	5	7	10	12	15
B	1	4	6	9	11	14	16
C	1	4	7	9	12	14	17

(b) 출력 시간 테이블

간격	1	2	3	4	5	6	7
A	0	1	3	2	3	2	3
B	0	3	2	3	2	3	2
C	0	3	3	2	3	2	3

(c) 출력 시간 간격 테이블

그림 4. 출력 시간과 출력 시간 간격 테이블의 생성 예
Fig. 4. The table of output time and output time distance.

증명) 만일 블록 b의 출력 시간이 L_c가 아닌 사이클에 속하는 블록 d에 의해 결정된다고 가정하자. 이때 출력 단계는 $j(\geq i)$이다. 블록 d에서 블록 b까지의 경로를 P_1, 블록 c에서 블록 b까지의 경로를 P_2라 하면 블록 b의 출력 시간은 식 (3)에 의해 다음과 같다.

$$\begin{aligned}t_b(j)=max\ &\{t_d(j-|P_1|)+w(P_1)+|P_1|,\\ &t_c(j-|P_2|)+w(P_2)+|P_2|\}\\ =\ &t_d(j-|P_1|)+w(P_1)+|P_1|\end{aligned}$$

그러면 처리량은 식 (1)에 의해 다음과 같다.

$$\frac{j}{t_d(j-|P_1|)+w(P_1)+|P_1|}$$

그러므로 처리량은 L_c가 아닌 사이클에 의해 결정된다. 이는 식 (2)에 의해서 L_c가 처리량을 결정한다는 사실에 모순이다.□

이제, 정리 1의 출력 단계 i를 출력 시간 안정화 시점 S_t라 한다. S_t이후 B_{scc}에 속하는 모든 블록의 출력 시간 간격은 정리 2를 만족한다.

정리 2. B_{scc}에 속하는 모든 블록의 출력 시간 간격은 S_t 부터 패턴이 반복되며 패턴의 최대 길이는 $|L_c|$이다.

증명) 정리 1에 의해 B_{scc}에 속하는 임의의 블록 b는 자기 자신의 출력 시간에 영향을 받는다. 따라서 식 (3)에 의해 다음과 같은 출력 시간이 성립한다.

$$\begin{aligned}t_b(S_t+|L_c|)&=t_b(S_t)+w(L_c)+|L_c|\\ t_b(S_t+|L_c|+1)&=t_b(S_t+1)+w(L_c)+|L_c|\end{aligned}$$

이때 두 식의 우변에서 $t_b(S_t)$와 $t_b(S_t+1)$을 각각 좌변으로 이항하면 다음과 같은 식이 성립한다.

$$\begin{aligned}t_b(S_t+|L_c|+1)-t_b(S_t+1)&=t_b(S_t+|L_c|)-t_b(S_t)\\ t_b(S_t+|L_c|+1)-t_b(S_t+|L_c|)&=t_b(S_t+1)-t_b(S_t)\end{aligned}$$

식 (4)에 의해서 출력 시간 간격은 $|L_c|$마다 반복된다. 즉, 출력 시간 간격 패턴의 최대 길이는 $|L_c|$이다.□

그림 4는 임의의 lis-graph G의 SCC에 대한 출력 시간과 출력 시간 간격 테이블의 간단한 생성 예를 보이고 있다. 먼저 그림 4(b)의 블록 B를 보자. 출력 시간의 초기 값은 식 (3)에 의해 1이다. 2번째 단계 출력 시간은 $\max((t_a(1)+w(A,B)+1),(t_c(1)+w(C,B)+1))=\max(4,3)=4$이다. 3번째 단계 출력 시간은 $\max(t_a(2)$

$+w(A,B)+1$, $t_c(2)+w(C,B)+1)=\max(2+2+1,$ $4+1+1)=6$이다. 이와 같은 방식으로 블록 B의 7번째 출력 시간은 16이 된다. 그림 4(a)의 모든 블록의 출력 시간은 그림 4(b)와 같다. 출력 시간 간격 테이블은 출력 단계마다 출력 시간을 구하고 식 (4)을 수행하여 그림 4(c)와 같이 얻을 수 있다.

3. 새로운 처리량 계산 방법

임의의 SCC에 속한 블록의 집합 B_{scc}의 처리량은 정리 3과 같다.

정리 3. B_{scc}의 처리량은

$$\frac{|L_c|}{t_b(S_t+|L_c|+j)-t_b(S_t+j)}$$

이다. 단, $j \geq 0$, $b \in B_{scc}$.

증명) 정리 2에 의해 S_t부터 출력 시간 간격은 $|L_c|$마다 패턴을 가지기 때문에 다음과 같은 식이 성립한다.

$$\begin{aligned}
&t_b(S_t+k|L_c|+j)-t_b(S_t+j)\\
&=\{t_b(S_t+k|L_c|+j)-t_b(S_t+k|L_c|+j-1)\}\\
&\quad+\{t_b(S_t+k|L_c|+j-1)-t_b(S_t+k|L_c|+j-2)\}\\
&\quad\vdots\\
&\quad+\{t_b(S_t+(k-1)|L_c|+j)-t_b(S_t+(k-1)|L_c|+j)\}\\
&\quad\vdots\\
&\quad+\{t_b(S_t+|L_c|+j)-t_b(S_t+j)\}\\
&=k[\{t_b(S_t+|L_c|+j)-t_b(S_t+|L_c|+j-1)\}\\
&\quad\cdots+\{t_b(S_t+j+1)-t_b(S_t+j)\}],\ where\ k\geq 1
\end{aligned}$$

위 식과 식 (1)에 의해 처리량은 다음과 같다.

$$\begin{aligned}
&\frac{k|L_c|}{t_b(S_t+k|L_c|+j)-t_b(S_t+j)}\\
&=\frac{k|L_c|}{k[\{t_b(S_t+|L_c|+j)-t_b(S_t+j-1)\}\\
&\quad\cdots+\{t_b(S_t+k|L_c|+j)-t_b(S_t+j)\}]}\\
&=\frac{|L_c|}{t_b(S_t+|L_c|+j)-t_b(S_t+j)}
\end{aligned}$$

□

정리 3에서 출력 시간 기반으로 정확한 처리량을 계산할 수 있다는 사실을 알 수 있다. 반면에 정리 3에서 S_t와 $|L_c|$을 알지 못하므로 실제 처리량을 계산할 수는 없다. 이에 현재 출력 단계 n에서 정확한 처리량을 계산하는 방법을 정리 4에 보인다.

정리 4. 현재 출력 단계 n에서 B_{scc}에 속하는 모든 블록 쌍 (b_i, b_j)이 $t_{b_i}(n)-t_{b_i}(n-l)=t_{b_j}(n)-t_{b_j}(n-l)$을 만족하면 정확한 처리량은

$$\frac{l}{t_{b_i}(n)-t_{b_i}(n-l)},\ l\geq 1$$

이다. 단, n은 현재 출력 단계, l은 패턴의 길이.

증명) 블록 b_i의 $n-l+1$번째 출력시간이 블록 b_j의 $n-l$번째 출력 시간에 의해 결정된다고 하면 다음과 같은 식이 성립한다.

$$\begin{aligned}
t_{b_i}(n-l+1)&=max\ \{t_v(n-l)+w(v,b_i)+1\}\\
&=t_{b_j}(n-l)+w(b_j,b_i)+1
\end{aligned}$$

조건 $t_{b_i}(n)-t_{b_i}(n-l)=t_{b_j}(n)-t_{b_j}(n-l)=\alpha$라 하자. 가정에 의해 $n>S_t$이므로 식 (3)에 의해 다음과 같은 식이 성립한다.

$$\begin{aligned}
t_{b_i}(n+1)&=max\ \{t_v(n)+w(v,b_i)+1\}\\
&=\max\{t_v(n-l)+\alpha+w(v,b_i)+1\}\\
&=t_{b_j}(n-l)+\alpha+w(b_j,b_i)+1\\
&=t_{b_j}(n)+w(b_j,b_i)+1
\end{aligned}$$

즉, 패턴 길이 l마다 블록 b_i의 출력 시간은 블록 b_j에 의해 결정된다. 위의 두 식을 빼면 다음과 같다.

$$t_{b_i}(n+1)-t_{b_i}(n-l+1)=t_{b_i}(n)-t_{b_i}(n-l)$$

식 (1)의 처리량 정의에 따라 패턴의 길이 l마다 출력 시간은 $t_{b_i}(n)-t_{b_i}(n-l)$와 동일하므로 정리 4가 성립한다.□

정리 4는 출력 시간으로 처리량을 계산하기 때문에 전체 그래프의 구조를 알 필요가 없다. 이에 식 (2)의 처리량 계산 방법에 비해 계산 속도의 향상을 기대할 수 있다. 또한 정리 4는 기존의 휴리스틱 처리량 평가 방법과 달리 정확한 처리량을 계산한다.

정리 2에 의해 패턴의 길이 l는 $|L_c|$을 넘지 못하므로 이 비교 연산은 정점의 수 V을 넘지 못한다. 따라서 최대 출력 시간 간격 패턴을 찾는데 걸리는 시간 복잡도는 $O(V)$이다.

그림 5는 제안한 처리량 계산 방법의 전체 흐름을 보여주고 있다. lis-graph를 입력받아 SCC로 분할하고 각 SCC에 대해 처리량을 계산한다. 모든 SCC에 대한 처리량을 구한 후 가장 작은 값을 lis-graph의 처리량으로 선택한다.

정확한 처리량이 가능한 출력 단계를 k라 하자. 출력

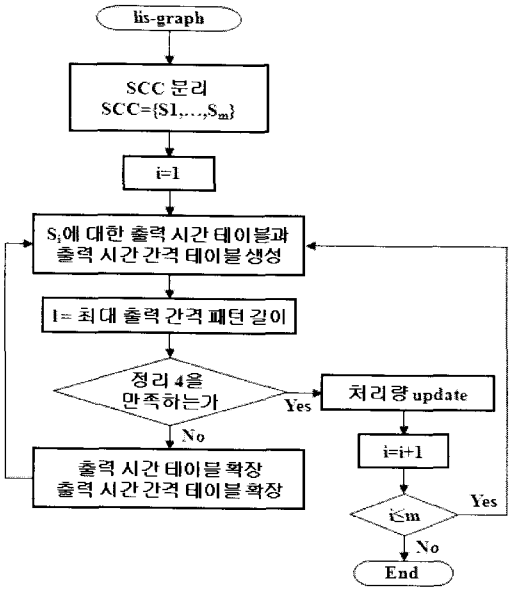


그림 5. 제안한 처리량 계산 방법의 흐름도
Fig. 5. The flow of proposed throughput computation.

시간 테이블의 확장 시간은 $O(E)$이고 최대 출력 시간 간격 패턴 길이를 구하는 시간과 정리 4의 만족을 확인하는데 걸리는 시간은 모두 $O(V)$이므로 제안한 처리량 계산 방법의 복잡도는 $O(k(E+V))=O(kE)$이다. 여기서 k의 복잡도는 최악의 경우 $O(V^2)$이지만 E가 많을수록 출력 시간 안정화 시점이 빨라지는 경향이 생기므로 기존의 정확한 처리량 계산 방법의 $O(EV^2)$보다 실제적으로 빠르다.

Ⅳ. 실 험

본 논문은 2.8GHZ Pentium 4, 1GB RAM환경에서 실험하였다. 평면계획 전략으로 Simulated Annealing을 사용했으며 평면계획 모델로 Sequence pair를 사용했다. 실험 데이터는 MCNC와 GSRC 벤치마크 회로를 사용했다[6]. 처리량을 계산하기 위해서는 각 네트에 방향성이 필요하여 벤치마크 회로의 각 네트의 마지막 핀을 source pin으로 설정했다[7]. 반면에 MCNC 벤치마크는 네트의 마지막 핀을 source pin로 설정하면 사이클이 생기지 않으므로 랜덤하게 설정했다.

표 1은 휴리스틱과 제안한 방법을 적용한 평면계획 결과를 비교한 것이다. 비용함수로 면적과 처리량의 가중치를 각각 0.05와 0.95를 주어 결합하였으며 10번의 평면계획 결과에 대한 (평균/최소)를 기록했다. Area는 WS를 기록했으며 처리량 계산을 위해 1클록 당 이동할 수 있는 거리를 $0.3\times\sqrt{\text{전체 블록의 면적 합}}$ 으로

표 1. 휴리스틱과 제안한 방법의 평면계획 결과 비교
Table 1. Comparison of different floorplan results.

bench	Opt. Thr (heuristic method)		Opt. Thr (proposed method)		Thr 평균 향상률 (%)
	Area	Thr	Area	Thr	
ami33	12.30/10.00	0.44/0.50	13.40/10.30	0.46/0.50	4.55
ami49	9.87/7.50	0.31/0.35	19.80/9.82	0.35/0.45	12.90
apte	7.88/3.80	0.25/0.27	6.71/1.09	0.47/0.50	88.00
hp	14.80/3.73	0.30/0.38	15.50/9.60	0.31/0.33	3.33
xerox	9.44/5.21	0.20/0.30	11.00/4.96	0.26/0.30	30.00
n10	11.20/6.16	0.31/0.36	14.40/8.44	0.46/0.50	48.39
n30	10.40/8.49	0.32/0.36	14.40/11.80	0.35/0.37	9.38
n50	9.48/7.23	0.33/0.36	21.80/10.30	0.24/0.25	-27.27
n100	11.40/9.73	0.43/0.50	37.30/15.60	0.32/0.33	-25.58
Avg.	10.80/6.90	0.32/0.38	17.15/9.10	0.36/0.39	16.97

표 2. 세 처리량 비용함수의 계산 속도 비교
Table 2. Comparison speed of different throughput cost functions.

bench.	CPU time(ms)				
	exact	speed	heuristic	speed	proposed method
ami33	39.31	23.12	0.40	0.24	1.70
ami49	264.29	46.37	1.00	0.18	5.70
apte	1.24	2.07	0.20	0.33	0.60
hp	2.04	2.55	0.30	0.38	0.80
xerox	3.43	1.43	0.60	0.25	2.40
n10	0.97	1.08	0.20	0.22	0.90
n30	20.80	5.94	0.60	0.17	3.50
n50	78.61	6.50	1.00	0.08	12.10
n100	54.47	1.92	1.80	0.06	28.30
Avg.		10.11		0.21	

설정하고 각 핀 사이의 거리를 HPWL로 정했다. 실험 결과, WS가 평균 6.35% 증가했으나 처리량은 평균 16.97% 향상되었음을 확인할 수 있다.

표 2는 세 처리량 계산 방법의 계산 시간을 비교한 결과를 보이고 있다. 100개의 평면계획의 처리량을 계산하는데 걸린 시간을 100으로 나눈 평균을 기록했다. 기존의 방법들을 각각 제안한 처리량 계산 방법으로 나누어 계산 속도를 비교하였다. 제안한 처리량 계산 방법은 휴리스틱 평가 방법에 비해 평균 4.76배 느려졌지만 정확한 처리량 계산 방법에 비해 평균 10.11배 빨라졌다. 계산 시간은 ami49와 같이 계산 시간이 오래 걸리는 회로에서 더욱 향상되는 것으로 나타나고 있다. 이처럼 제안한 처리량 계산 방법은 평면계획에 사용하기에 적합한 방법이다.

V. 결 론

본 논문에서는 새 처리량 계산 방법을 제안했다. 제안한 방법은 기존의 정확한 처리량 계산 방법과 동일한 처리량을 계산하면서 실질적인 계산 시간이 향상 되었다. 그 결과, 제안한 방법은 평면계획의 비용함수로 사용할 수 있으며 기존의 휴리스틱 평가 방법을 적용한 평면계획보다 처리량이 평균 16.95% 향상되었다.

평면계획에서 처리량과 면적을 더 향상시킬 수 있는 조합과 처리량과 관련된 적절한 이웃 해를 생성하는 방법이 앞으로의 연구 대상이다. 또한 본 논문에서 사용한 Sequence Pair보다 더 빠르고 좋은 결과를 낸다고 알려진 B*-Tree 평면계획 모델을 평면계획에 적용한다면 더 좋은 결과를 얻을 수 있으리라 기대된다.

참 고 문 헌

[1] J. Cong, "Challenges and Opportunities for Design Innovations in Nanometer Technologies," in SRC Design Sciences Concept Paper, December 1997.

[2] L. P. Carloni, K. L. McMillan, and A. L. Sangiovanni-Vincentelli, "Theory of Latency Insensitive Design," *IEEE Trans. on CAD*, vol. 20, no. 9, pp.1059-1076, Sept. 2001.

[3] L. P. Carloni and A. L. Sangiovanni-Vincentelli, "Performance analysis and optimization of latency insensitive protocols," *in Proc. DAC*, pp. 361-367, 2000.

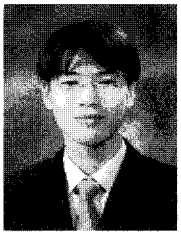
[4] M. R. Casu and L. Macchiarulo, "Throughput-Driven Floorplanning with Wire Pipelining," *IEEE Trans. on CAD*, vol. 24, no. 5, pp.663-675, May 2005.

[5] A. Gibbons, "Algorithmic graph theory," Cambridge University Press, pp. 7-33, 1985.

[6] http://www.cse.ucsc.edu/research/surf/GSRC/

[7] Y. Ma, X. Hong, S. Dong, S. Chen, Y. Cai, C. K. Cheng, J. Gu, "An Integrated Floorplanning with an Efficient Buffer Planning Algorithm", *in Proc. ISPD*, pp. 136-142, 2003.

저 자 소 개



강 민 성(정회원)

2004년 서강대학교 컴퓨터공학과 학사 졸업.

2006년 서강대학교 컴퓨터공학과 석사 졸업.

2007년 현재 (주) 하이닉스 연구원

<주관심분야 : VLSI 설계, CAD 알고리즘, SoC 설계>

임 종 석(평생회원)

1981년 서강대학교 전자공학과 학사 졸업.

1983년 한국과학기술원 전기 및 전자공학과 석사 졸업.

1989년 University of Maryland, College Park 전기공학과 박사 졸업.

1983년~1990년 8월 한국전자통신연구소 연구원.

1990년 9월~현재 서강대학교 컴퓨터공학과 교수.

<주관심분야 : 알고리즘, CAD, VLSI 설계>