논문 2006-43SD-9-1

싱글 페이즈 클락드 래치를 이용한 SoC 리타이밍

(Retiming for SoC Using Single-Phase Clocked Latches)

김 문 수*, 임 종 석**

(Moon-Su Kim and Chong-Suck Rim)

요 약

System-On-Chip(SoC) 설계에서 글로벌 와이어는 성능에 큰 영향을 끼친다. 이 때문에 플립플롭이나 래치를 사용한 와이어 파이프라이닝이 필요하게 되었다. 래치는 플립플롭에 비해 타이밍 제약이 유연하므로 래치 파이프라이닝이 플립플롭에 비해 클락 주기를 더 작게 할 수 있다. 리타이밍은 회로의 메모리 요소를 이동시켜 최적화된 클락 주기를 얻는 방법이다. SoC 리타이밍은 기존의 게이트 레벨 리타이밍과 달리 SoC 회로를 대상으로 한다. 본 논문에서는 기존의 플립플롭을 사용한 SoC 리타이밍 방법을 래치를 사용한 경우에도 적용할 수 있게 확장 시켰다. 본 논문에서는 래치를 사용한 SoC 리타이밍 문제를 해결하기 위해 MILP로 식을 세우고, 이를 고정점 계산을 통해 효과적으로 해결 하였다. 실험 결과 본 논문의 방법을 적용할 경우 플립플롭 SoC 리타이밍에 비해 평균적으로 클락 주기를 10% 감소시킬 수 있었다.

Abstract

In the System-on-Chip(SoC) design, the global wires are critical parts for the performance. Therefore, the global wires need to be pipelined using flip-flops or latches. Since the timing constraint of the latch is more flexible than it of the flip-flop, the latch-based design can provide a better solution for the clock period. Retiming is an optimizing technique which repositions memory elements in the circuits to reduce the clock period. Traditionally, retiming is used on gate-level netlist, but retiming for SoC is used on macro-level netlist. In this paper, we extend the previous work of retiming for SoC using flip-flops to retiming for SoC using single-phase clocked latches. In this paper we propose a MILP for retiming for SoC using single-phase clocked latches, and apply the fixpoint computation to solve it. Experimental results show that retiming for SoC using latches reduces the clock period of circuits by average 10 percent compared with retiming for SoC using flip-flops.

Keywords : SoC, retiming, wire pipelining, timing analysis, fixpoint computation

I. 서 론

최근 고성능, 고집적 회로를 빠른 시간 안에 설계하기 위하여 System-on-Chip(SoC) 설계 방식이 널리 사용된다. 또한 Intellectual Property(IP)를 연결하는 글로벌 와이어의 지연시간이 길어지고 클락 스피드가 향상됨에 따라 와이어 파이프라이닝이 필요하게 되었다. 이러한 와이어 파이프라이닝은 대부분 플립플롭(FF)을 사용했으나 최근에는 래치를 사용한 방법이 제시되었다[1]. 래치를 사용할 경우 유연한 타이밍 제약조건 때문에 FF에 비해 더 빠른 클락 주기로 회로를 동작시킬 수 있다. 리타이밍은 회로의 기능을 유지하면서 FF나 래치와 같은 메모리 요소를 이동시켜 클락 주기를 줄이는 방법이다. 전통적으로 리타이밍은 게이트 레벨 네트리스트에 적용했고 와이어의 지연시간은 없다고 가정하였다[2,3,4,5]. 게이트 레벨 리타이밍은 FF를 사용한 회로에서 래치를 사용한 회로[4,5]로 확장되었으며 최근에는 와이어의 지연시간을 고려한 리타이밍도 제안 되었다[6,7,8,9]. 또한 SoC 회로의 매크로 레벨 네트리스트를 대상으로 한 리타이밍도 제시되었다[7,8,9]. 하지만 기존의 SoC 리타이밍[7,8,9]은 FF를 사용할 경우만을 대상으로

* 정회원, 삼성전자 반도체
(Semiconductor, Samsung Electronics)
** 정회원, 서강대학교 컴퓨터학과
(Department of Computer Science, Sogang)
※ 본 연구는 idec(반도체설계교육센터)의 지원으로 수행되었습니다.
접수일자: 2006년7월1일, 수정완료일: 2006년8월18일

하고 있어 래치를 사용할 경우 적용될 수 없다.

본 논문에서는 싱글 페이즈 클락에서 래치를 사용한 SoC 회로의 리타이밍 방법을 제시한다. 이를 위해 참고문헌[8,9]의 방법을 래치에 적용되도록 확장 시켰다. 제 II 장에서는 클락 모델 및 래치의 특성과 장점을 알아본다. 제 III장에서는 SoC 회로의 그래프 모델을 소개하고 FF[8,9]와 래치를 사용한 경우의 MILP 식을 제시한다. 제 IV장에서는 리타이밍 해를 구하는 고정점 계산 방법을 소개한다. 제 V장에서는 실험 결과를 기존의 방법과 비교하였고, 그 결과 같은 입력에 대해 FF를 사용한 경우[8,9]보다 평균 10% 감소된 클락주기를 얻을 수 있었다. 마지막으로 제 VI장에서는 본 논문의 결론을 정리한다.

본 논문에서 가정하는 클락 모델은 그림 1과 같다[10]. 클락의 로컬 타임 존은 액티브 구간과 패시브 구간으로 구성된다. 클락 주기를 T 라하고 액티브 구간의 길이를 T_p라 할 때 패시브 구간의 길이는 $T-T_p$가 된다. 액티브 구간에 신호가 도착할 경우 즉시 래치를 통과하지만 패시브 구간에 도착할 경우 신호는 인에이블링 에지 후에 출발할 수 있다.

p,q를 FF, 래치 또는 임의의 지점이라 할 때 $t(p)$는 p에서 가장 늦은 신호의 도착시간, $s(p)$는 p에서 신호의 가장 늦은 출발시간, $d(p,q)$는 p와 q 사이의 지연시간을 나타낸다. 만일 p와 q 사이에 래치가 있을 경우 p와 래치, 래치와 래치, 래치와 q 사이의 지연시간의 합은 $d(p,q)$와 같다고 본다. 이는 버퍼를 최적으로 삽입할 수 있다는 가정을 전제로 한다.

신호의 도착시간 및 출발 시간은 각 로컬 타임 존의 패시브 구간의 시작점을 기준으로 한다. 즉, 그림 2 (a)와 같이 래치 l_i, l_j가 인접해 있을 때 $t(l_i)$, $s(l_i)$는 그림 2 (b)의 시점 1을 기준으로 정의 되고 $t(l_j)$, $s(l_j)$와 두 래치 사이의 지점 p의 도착시간 $t(p)$는 시점 2를 기준으로 정의된다. 따라서 래치 l_i와 인접한 p에 대해 아래의 식 1이 성립한다[10]. 여기서 $t(p)$는 시점 2를 기준으로 정의되었기 때문에 p가 l_i에 가깝다면 (즉 $d(l_i,p)$가 작다면) 음수일 수도 있다.

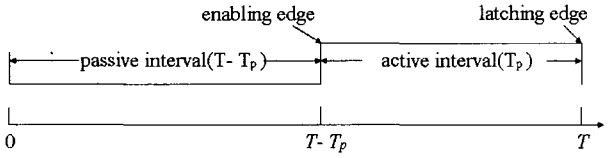

그림 1. 클락 모델
Fig. 1. Clock model.

$$t(p) = s(l_i) + d(l_i,p) - T \tag{1}$$

회로가 올바르게 동작하려면 래치 l의 $t(l)$과 $s(l)$은 아래의 식 2~4를 만족해야 한다[4].

$$s(l_p) + d(l_p,l) - T \le t(l) \tag{2}$$

$$t(l) \le s(l) \tag{3}$$

$$T - T_p \le s(l) \le T \tag{4}$$

식 2에서 l_p는 l의 입력에 영향을 주는 래치로 $t(l)$이 l에 가장 늦게 도착하는 시간이므로 식 1과는 달리 부등식으로 표현 된다. 식 3은 신호의 출발시간은 도착시간보다 크거나 같아야 함을 나타내고 신호가 액티브 구간에 도착할 경우에 등호가 성립한다. 또한 신호는 항상 액티브 구간에서 출발하므로 식 4가 성립된다.

FF 파이프라이닝에 비해 래치 파이프라이닝은 더 짧은 클락 주기를 얻을 수 있다. 그림 3 (a)와 같이 네트가 장애물 위로 배선되어 있을 때 X 지점에 메모리요

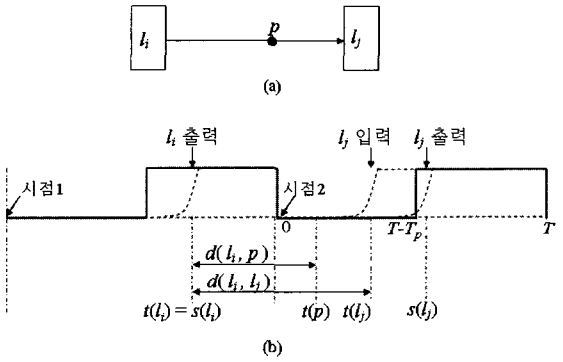

그림 2. 도착시간
Fig. 2. Arrival Time.

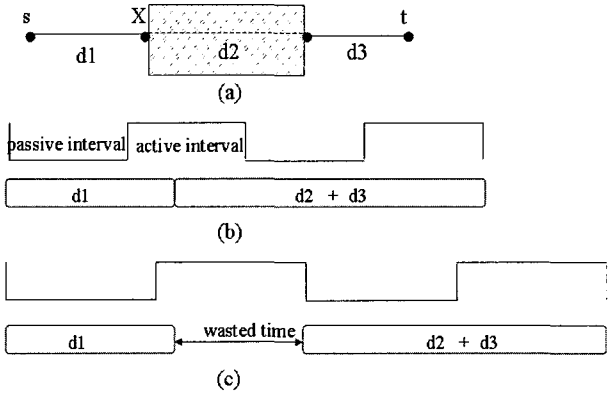

그림 3. 와이어 파이프라이닝
Fig. 3. Wire pipelining.

소를 넣어 파이프라이닝을 하고자 한다. 그림 3 (b)는 래치를 사용할 경우의 타이밍 다이아그램이고 그림 3 (c)는 FF를 사용할 경우이다. 그림에서처럼 신호가 액티브 구간에 도착할 때 래치의 경우 신호를 통과시키지만 FF의 경우 래칭 에지까지 기다려야 하기 때문에 낭비되는 시간(wasted time)이 발생한다. 이와 같은 경우 래치를 사용할 때 더 짧은 클락 주기를 얻을 수 있다.

II. 래치를 사용한 SoC 리타이밍의 MILP식

1. SoC 디자인의 그래프 모델

리타이밍의 입력으로 사용될 그래프 모델을 소개한다. 그림 4 (a)와 같은 SoC 디자인은 그림 4 (b)처럼 그래프 $G=(V,E)$로 나타낼 수 있다[7]. 여기서 V는 정점의 집합이고 E는 간선의 집합이다. V에 속한 각 정점은 칩의 입출력, IP의 입출력(v1/v6), 스타이너 포인트(v8), IP 위로 배선된 와이어와 IP의 교차점(v11,v12), 가상 FF의 입출력(v4,v5) 등을 나타낸다. IP 내의 모든 FF은 하나의 가상 FF으로 표현 가능하고 가상 FF은 간선으로 표현한다. E는 메모리요소를 삽입할 수 있는 간선의 집합 E_a와 할 수 없는 E_f로 나눌 수 있다. 본 논문에서 두 정점 u, v를 잇는 간선은 (u,v)로 표기하고 $d(u,v)$를 (u,v)의 지연시간, $w(u,v)$는 간선 위의 메모리 요소 수를 나타낸다. 그림 4 (b)에서 각 간선 (u,v)에는 $(d(u,v),w(u,v))$의 가중치 값이 있다. 간선 (u,v)가 가상 FF을 나타낼 경우 $d(u,v)$=0 이고 $w(u,v)$=1의 값을 갖는다(그림 4 (b)의 (v4,v5)).

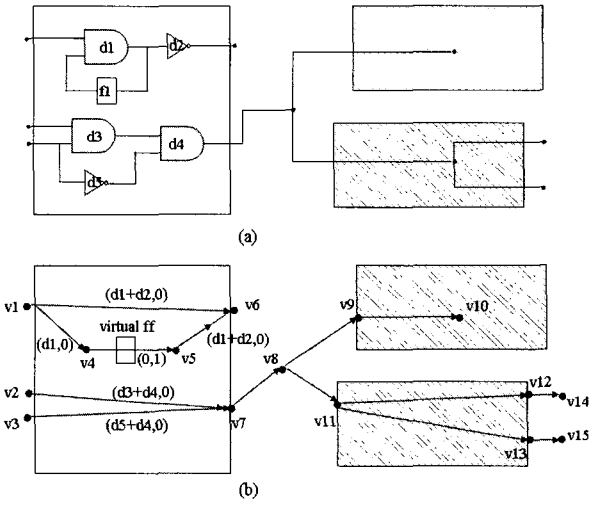

그림 4. SoC 디자인의 그래프 모델
Fig. 4. graph model of the SoC design $G=(V,E)$.

2. 기존식 및 기호 정리

이번 절에서는 리타이밍에 사용될 기호를 정리하고 기존의 FF를 사용한 SoC 리타이밍의 MILP 식[8,9]을 소개한다. PI 와 PO은 각각 칩의 입력과 출력을 나타내는 정점의 집합이라 한다. $r(v)$는 리타이밍 이후 정점 v의 진출 간선으로부터 v의 진입 간선으로 이동하는 메모리 요소의 수를 나타낸다[2]. $w_r(u,v)$를 리타이밍 이후 간선 (u,v)의 메모리 요소 수라 하면 $w_r(u,v)$는 아래의 식 5를 통해 구할 수 있다.

$$w_r(u,v)=w(u,v)+r(v)-r(u) \tag{5}$$

$fd(v)$와 $bd(v)$는 식 6, 7과 같이 정의한다. v의 진입 경로와 진출 경로에 최적으로 메모리 요소가 배치되어 있을 때, $fd(v)$는 정점 v까지 신호가 오기위해 반드시 필요한 도착시간이고 $bd(v)$는 v에서 출발한 신호가 메모리 요소를 만날 때까지 반드시 소모해야 하는 최대 지연시간이다. 따라서 $fd(v)$, $T-bd(v)$는 각각 $t(v)$의 최소값과 최대값이 된다. 여기서 변수 L은 참고문헌 [8,9]의 정의를 래치에도 적용할 수 있게 도입된 변수이다(FF: $L=0$, 래치:$L=1$).

$$fd(v)=\begin{cases} -T_p\cdot L & if\ \forall(u,v)\in E_a \\ 0 & if\ \forall(u,v)\in E_f, w(u,v)=1 \\ \max\limits_{\forall(u,v)\in E_f}\{d(u,v)+fd(u)\} & if\ \forall(u,v)\in E_f, w(u,v)=0 \end{cases} \tag{6}$$

$$bd(v)=\begin{cases} \max\limits_{\forall(v,u)\in E_f}\{d(v,u)\} & if\ \forall(v,u)\in E_f \\ 0 & otherwise \end{cases} \tag{7}$$

FF 기반의 SoC 리타이밍의 MILP식[5]은 다음과 같다.

$$r(u)=r(v), \qquad \forall\ (u,v)\in E_f \tag{8}$$

$$t(v)\geq t(u)+d(u,v) -(w(u,v)+r(v)-r(u))T,\ \forall\ (u,v)\in E \tag{9}$$

$$fd(v)\leq t(v)\leq T-bd(v),\ \forall v\in V \tag{10}$$

$$t(v)=0, \qquad \forall\ v\in PI \tag{11}$$

$$r(v)=0, \qquad \forall\ v\in PI\ \ or\ \ v\in PO \tag{12}$$

식 8은 리타이밍 후 IP 내부의 메모리 요소가 변경되지 않도록 해준다. 식 9, 10은 각 정점 v의 도착시간을 긴 경로 제약 조건에 맞도록 해준다. 회로의 입력에

신호의 도착시간이 0임을 가정했기 때문에 식 11이 필요하고, 식 12는 칩 외부에서 메모리 요소가 추가되지 않도록 하기 위해 필요하다. 주기 T가 고정될 때 식 8~12를 통해 리타이밍 해 $r(v)$,$t(v)$를 구할 수 있다. 따라서 주기의 상한값[7]과 하한값[8,3]을 구한 후 이진 탐색으로 최소의 주기를 갖는 리타이밍 해를 구할 수 있다.

3. 래치 SoC 리타이밍의 MILP 식

본 논문에서는 와이어파이프라이닝에 positive level-sensitive latch를, IP 내부는 falling edge-triggered FF을 사용한다고 가정한다. 또한 클락 스큐는 없고 각 래치 및 FF은 이상적으로 동작한다고 가정한다. E_a에 속하는 간선의 지연시간은 추후에 버퍼가 최적으로 삽입 가능하다는 전제하에 길이에 비례하는 것으로 본다. 또한 각 래치 및 FF에 대해 긴 경로제약 조건만 고려하여 리타이밍을 하고 짧은 경로 제약은 지연 회로의 삽입으로 해결 가능하다고 가정한다. 이 때 래치 SoC 리타이밍을 해결하기 위한 MILP 식은 다음과 같다.

$$r(u) = r(v), \qquad \forall (u,v) \in E_f \tag{13}$$

$$w(u,v) + r(v) - r(u) \geq 0, \forall (u,v) \in E_a \tag{14}$$

$$t(v) \geq t(u) + d(u,v) - (w(u,v) + r(v) - r(u))T, \forall (u,v) \in E \tag{15}$$

$$fd(v) \leq t(v) \leq T - bd(v), \forall v \in V \tag{16}$$

$$t(v) = 0, \qquad \forall v \in PI \tag{17}$$

$$r(v) = 0, \qquad \forall v \in PI \ or \ v \in PO \tag{18}$$

래치를 사용할 경우 식 16의 $fd(v)$의 범위가 식 6에 의해 FF의 경우와 다르다. FF의 경우 $fd(v) \geq 0$이지만 래치의 경우 $fd(v) \geq -T_p$이다.

식 14는 리타이밍 후 간선 위의 메모리 요소 수가 음수가 되는 것을 막기 위해 필요하다. 하지만 FF의 경우 식 14가 불필요하다[8,9]. FF를 사용할 경우 식 9(래치의 식 15)를 $w(u,v)+r(v)-r(u)$에 관해 정리하면, $w(u,v)+r(v)-r(u) \geq \frac{t(u)+d(u,v)-t(v)}{T}$ 이고 $d(u,v) > 0, t(u) \geq fd(u) \geq 0, t(v) \leq T$ 이므로 $w(u,v)+r(v)-r(u) > \frac{fd(u)-T}{T} \geq -1$이다. 따라서 FF의 경우 식 14가 항상 만족된다. 반면에 래치의 경우 $fd(u) \geq -T_p$이므로 식 14가 필요하다.

식 15, 16은 FF의 경우와 같은 모양이다. 하지만 래치를 사용할 때에도 식 15와 16으로 구한 도착시간에 대해 모든 간선 $(u,v) \in E$ 위의 $w_r(u,v)$ 개의 래치를 긴 경로 제약조건을 만족하게 배치시킬 수 있다. 이를 확인하기 위해 식 15, 16을 만족하는 $t(u)$를 기준으로 간선 위의 각 래치를 식 2 ~ 4의 조건에 맞게 배치하여 v의 도착시간을 구해 본다. 구한 v의 도착시간이 식 15, 16을 만족하면 모든 지점에서 긴 경로제약조건이 만족된다고 할 수 있다.

$(u,v) \in E_f$ 이거나 $(u,v) \in E_a, w_r(u,v) = 0$인 경우는 직관적으로 확인할 수 있으므로 $(u,v) \in E_a$, $w_r(u,v) > 0$인 경우만 살펴보자. 단순하게 $w_r(u,v)$를 w_r로 표기하고 w_r개의 래치를 각각 l_1, l_2,$\cdots$, l_{w_r}라 하자. 식 1(또는 2)에 의해 각 래치 및 정점 v의 도착시간을 적으면,

$$t(l_1) = t(u) + d(u,l_1)$$

$$t(l_2) = s(l_1) + d(l_1,l_2) - T$$

$$\vdots$$

$$t(l_{w_r}) = s(l_{w_r-1}) + d(l_{w_r-1},l_{w_r}) - T$$

$$t(v) = s(l_{w_r}) + d(l_{w_r},v) - T$$

이 된다. 이 때 각 $t(l_k)$, $s(l_k)$가 식 3, 4를 만족하도록 $d(u,l_1)$, $\cdots$, $d(l_{w_r-1},l_{w_r})$이 결정 되었다고 하자. 이를 모두 더하여 $t(v)$에 관해 정리하면

$d(u,l_1) + d(l_1,l_2) + \cdots + d(l_{w_r},v) = d(u,v)$이므로

$$t(v) = t(u) + d(u,v) - w_r T + \sum_{k=1}^{k=w_r} (s(l_k) - t(l_k))$$

가 된다. 식 3에 의해 위 식은 아래와 같이 쓸 수 있다.

$$t(v) \geq t(u) + d(u,v) - w_r T \tag{*}$$

식 *에서 등호가 성립할 때는 각 래치의 도착 시간 $t(l_k)$과 출발 시간 $s(l_k)$이 같을 때로 신호가 액티브 구간에 도착하는 경우이다.

한편, $t(v) = s(l_{w_r}) + d(l_{w_r},v) - T$ 에서 식 4에 의해 $s(l_k) \geq T - T_p$이고 $d(l_{w_r},v) \geq 0$이므로 $t(v)$는 $-T_p$ 보다 크거나 같아야 한다.

만약 각 래치를 $s(l_k)=t(l_k)$이 되도록 배치할 수 있을 만큼 $d(u,v)$가 긴 경우 식 *의 우변의 값이 $-T_p$ 보다 크거나 같은 값이 된다. 이 때 각 래치는 식 2 ~ 4를 만족하므로 긴 경로 제약 조건을 만족하게 배치시킬 수 있다. 이 때 $t(v)$는 식 15의 결과와 같다.

만약 식 *의 우변의 값이 $-T_p$ 보다 작다면 $s(l_k)=t(l_k)$가 만족되도록 래치를 배치시킬 수 없으므로 $s(l_k)>t(l_k)$가 되도록 래치를 배치하거나 지연시간 삽입을 통해 $t(v)\geq -T_p$로 만들어 줄 수 있다. 이 때 $t(v)$는 식 16의 결과와 같다.

따라서 어떤 경우든 식 13~16을 만족하는 해를 구한 후 각 간선 위의 래치를 긴 경로 제약조건에 맞게 배치할 수 있다.

리타이밍 문제를 MILP로 해결할 수 있으나 이는 시간 면에서 비효율적인 방법이다. 따라서 본 논문에서는 참고문헌 [8, 9]의 고정점 계산 방법을 래치에 적용되도록 확장시켜 문제를 해결하도록 한다.

III. 고정점 계산을 이용한 SoC 리타이밍

1. 플립플롭 SoC 리타이밍의 고정점 계산

이 절에서는 FF을 사용한 SoC 리타이밍을 해결하기 위한 고정점 계산 방법[8,9]을 설명한다.

어떤 함수 $F,F: P\rightarrow P$에 대해 $X=F(X)$를 만족하는 X를 F의 고정점이라 하고 이중 최소값을 최소 고정점이라 한다. 만일 집합 P가 Partially Ordered set이고 상한값과 하한값을 가지고 있을 경우 P를 Complete Partially Ordered set(CPO)이라 한다. P가 CPO이고 F가 단조함수인 경우 반복적인 방법으로 F의 최소 고정점을 구할 수 있다. P의 하한값 X_L에 대해 $F(X_L)=X_1$을 구하고 $F(X_1)=X_2$, $F(X_2)=X_3$등의 계산을 반복하여 $F(X_i)=X_i$인 X_i를 구할 수 있고 이는 F의 최소 고정점이 된다[11,8,9].

$r(v),t(v),\forall v\in V$를 식 8~10을 만족하는 리타이밍 문제의 해라하고 $r_1(v)$, $t_1(v)$를 아래의 식 19, 20과 같이 정의 하자.

$$r_1(v)=\max\{\max_{\forall(u,v),(v,u)\in E_f, r(v)<r(u)} r(u), \max_{\forall(u,v)\in E_a}\left\lceil\frac{t(u)+d(u,v)}{T}-w(u,v)+r(u)\right\rceil-1\} \quad (19)$$

$$t_1(v)=\max\{fd(v),\ \max_{\forall(u,v)\in E} t(u)+d(u,v)-(w(u,v)+r_1(v)-r(u))T\} \quad (20)$$

$r(u),t(u)$가 주어졌을 때 $r_1(v)$, $t_1(v)$로 u에 인접한 정점 v의 리타이밍 해를 구할 수 있다.

먼저 (u,v), (v,u)가 E_f에 속하는 간선이면 식 8을 만족해야 하므로 항상 $r(u)=r(v)$가 되어야 한다. 식 19에 의해 $r_1(v)=r(u)$가 된다.

다음으로 (u,v)가 E_a에 속하는 경우를 알아보자. 만일 $r_1(v)$ 개의 FF가 v의 진입간선에 추가된다면 v에서의 도착시간은 $t_1(v)$이다. v로의 진입 간선에 너무 적은 FF가 삽입될 경우 정점 v로의 도착시간이 T보다 커져 회로가 제대로 동작되지 않게 된다. 반대로, 과다한 FF를 삽입하여도 v로의 도착시간은 0보다 작아 질 수 없으므로 FF가 낭비 된다. $r_1(v)$는 최소한의 FF를 삽입하여 $t_1(v)$ 범위를 $0<t_1(v)\leq T$로 하기 위한 값이다[8,9].

$t_1(v)$는 $r(v)=r_1(v)$일 때 식 9와 식 10의 좌변을 만족하는 $t(v)$ 중 최소값이다. 4 장에서는 정점의 도착시간을 구할 때 항상 식 9, 식 10을 만족하는 값 중 최소값으로 구한다. 그 이유는 각 정점의 도착시간이 최소값일 때도 해가 존재하지 않으면 주어진 주기에서 리타이밍 해가 존재하지 않음을 명확하게 알 수 있기 때문이다. 아래의 식 21에서 $t_1(v)$의 값에 따라 $r(v)$, $t(v)$ 값을 구할 수 있다.

$$(r(v),t(v))=\begin{cases}(r_1(v),t_1(v)) & if\ t_1(v)\leq T-bd(v)\\ (r_1(v)+1,fd(v)) & otherwise\end{cases} \quad (21)$$

$t_1(v)\leq T-bd(v)$인 경우 $r_1(v)$, $t_1(v)$는 식 9, 10을 모두 만족하므로 리타이밍의 해가 된다.

$t_1(v)>T-bd(v)$인 경우를 살펴보자. 간선 $(v,w)\in E_f$가 존재할 때 식 7에 의해 $bd(v)$는 $d(v,w)$가 된다. (v,w)에는 추가적으로 FF를 삽입할 수 없으므로 $t_1(v)>T-bd(v)$인 경우 $t(w)=t_1(v)+d(v,w)$는 T 보다 큰 값이 되므로 긴 경로 제약조건에 위배되어 회로가 정상 동작하지 않는다. 따라서 이 경우 v에 FF를 추가하여 v에서의 도착시간을 줄여야 한다. $r(v)=r_1(v)+1$일 때 식 9와 식 10의 좌변을 만족하는 $t(v)$ 중 최소값을 구하기 위해 먼저 식 9에 $r(v)=r_1(v)+1$을 대입하면 $t(v)\geq t(u)+d(u,v)-(w(u,v)+r_1(v)+1-r(u))T$가 된다. 위 식은 $t(v)\geq t_1(v)-T$

로 다시 쓸 수 있다. $t_1(v) \le T$ 이므로 위 식의 우변은 항상 0 보다 작거나 같게 된다. 이와 같은 경우 식 6에 의해 $fd(v) = 0$ 이고 v에서의 도착시간은 식 10에 의해 결정되므로 그 중 최소값은 $fd(v)$가 된다. 따라서 $t(v) = fd(v)$가 되어야 한다.

정리 1[8,9]. *식 8~10의 해가 존재하는 것은 $\forall v \in V$에 대해 식 21의 해가 존재하는 것의 필요충분조건이다.* □

정리 1의 증명은 참고 문헌 [8, 9]를 참조한다.

P를 X의 해공간이라 할 때 식 21은 $X = F(X)$, $F: P \rightarrow P$의 형태이다. $x_v = (r(v), t(v))$를 정점 v의 리타이밍 해, $X = (x_1, x_2, ..., x_n), n = |V|$를 집합 V의 리타이밍 해를 나타내는 벡터라 하자. 이때 리타이밍 해 x_v와 x'_v에 대해 부분순서($\le$)를 식 22와 같이 정의한다.

$$x_v \le x'_v \leftrightarrow r(v) < r'(v) \ or$$
$$r(v) = r'(v), t(v) \le t'(v) \quad (22)$$

P에 속하는 두 벡터 $X = (x_1, x_2, ..., x_n)$, $Y = (y_1, y_2, ..., y_n)$에 대한 부분순서($\le$)는 아래와 같다.

$$X \le Y \leftrightarrow x_i \le y_i, \ \forall 1 \le i \le n \quad (23)$$

P를 CPO로 만들기 위해 P의 하한값X_L, 상한값X_U을 다음과 같이 정의 한다[8,9].

$$X_L: \ t(v) = r(v) = 0, \ \forall v \in PI$$
$$t(v) = r(v) = -\infty, \ \forall v \in V - PI$$
$$X_U: \ t(v) = r(v) = \infty, \ \forall v \in V$$

정리 2[9]. *식 19를 나타내는 함수 F는 단조함수이다.* □

X의 해공간이 CPO이고 F는 단조함수이므로 반복적 방법으로 F의 최소 고정점을 찾을 수 있다[11,8,9]. 이때 최소 고정점은 식 8~10, 식 19의 해가 된다. 식 10, 11은 하한값 X_L을 초기값으로 해줄 때 만족된다.

정리 3[8,9]. *주어진 주기 T에서 F가 유한하게 수렴하는 것은 같은 주기에서 리타이밍 해가 존재 하는 것의 필요충분조건이다.* □

$X = F(X)$을 만족하는 해를 구하기 위한 반복과정에서 전체 V 대신 변화가 생기는 정점만 갱신 해줘도 최소고정점을 구할 수 있다[8,9]. 또 각 정점을 갱신할 때 그 순서는 고려하지 않아도 된다[8,9].

2. 래치 SoC 리타이밍의 고정점 계산

이 절에서는 래치를 사용한 SoC 리타이밍의 해를 고정점 계산을 사용해 구하기 위해 MILP 식을 $X = F(X)$의 형태로 변환한다. $r(v), t(v), \forall v \in V$를 식 13 ~ 16을 만족하는 리타이밍 문제의 해라하고 $r_1(v)$, $t_1(v)$를 아래의 식 24, 25, 26과 같이 정의하자. FF의 경우와 마찬가지로 $r(u), t(u)$가 주어졌을 때 u에 인접한 정점 v의 리타이밍 해를 구할 수 있다.

$$r_1(v) = \max\{r_2(v) \overset{def}{=} \max_{\forall (u,v),(v,u) \in E_f, r(v) < r(u)} r(u),$$
$$r_3(v) \overset{def}{=} \max_{\forall (u,v) \in E_a} \left\lceil \frac{t(u) + d(u,v)}{T} - w(u,v) + r(u) \right\rceil - 1 \quad (24)$$
$$r_4(v) \overset{def}{=} \max_{\forall (u,v) \in E_a} (r(u) - w(u,v)) \ \}$$

$$t_1(v) = \max\{fd(v), \ \max_{\forall (u,v) \in E} t(u) + d(u,v) - (w(u,v) + r_1(v) - r(u))T\} \quad (25)$$

$$t_2(v) = \max\{fd(v), \ \max_{\forall (u,v) \in E} t(u) + d(u,v) - (w(u,v) + r_1(v) + 1 - r(u))T\} \quad (26)$$

(u,v) 또는 (v,u)가 E_f에 속하는 경우는 FF의 경우와 동일하므로 (u,v)가 E_a에 속하는 경우만 보도록 한다. 식 24의 $r_3(v)$와 식 25를 통해 $r_1(v)$ 와 $t_1(v)$를 구하면 $t_1(v)$의 범위는 $0 < t_1(v) \le T$가 된다.

$t_1(v)$가 $T - bd(v)$보다 작거나 같으면 $r_1(v)$, $t_1(v)$는 해가 되고, 그렇지 않으면 래치를 1개 더 추가시켜야만 한다. v에 래치 $r_1(v)$ + 1개를 추가 시켰을 때 v에서의 도착시간을 식 26의 $t_2(v)$라 하자. $t_2(v)$는 $r(v) = r_1(v) + 1$ 일 때 식 15와 식 16의 좌변을 만족하는 $t(v)$ 중 최소값이다. 식 26의 $t_2(v)$를 $t_1(v)$를 사용하여 다시 적으면 $t_2(v) = max\{fd(v), t_1(v) - T\}$가 된다. 이때 식 6에 의해 $fd(v) = -T_p$ 이고 $t_1(v)$는 0보다 크고 T 보다 작거나 같으므로 $t_2(v)$는 $fd(v) \le t_2(v) \le 0$의 범위를 갖는다.

$t_2(v)$가 $T - bd(v)$ 보다 큰 경우 v에 래치를 다시 1개 추가시켜 도착시간을 줄일 수 있다. $r(v) = r_1(v) + 2$ 를 식 15에 넣어 정리하면 $t(v) \ge t_1(v) - 2T$ 가 된다. 이 때 $t_1(v)$는 T보다 작거나 같으므로 우변은 항상 $-T$ 보다 작은 값을 갖는다. 이와 같은 경우 식 6에 의해 $fd(v) = -T_p$이므로 v의 도착시간은 식 16에 의해 결정되고 그 중 최소값은 $fd(v) = -T_p$이다.

마지막으로 래치 리타이밍에서는 FF의 경우에는 불

필요한 조건인 식 14를 만족시키기 위해 식 24에 $r_4(v)$가 필요하다. 위의 설명을 종합하면 정리 4의 식 27이 된다. 정리 4의 자세한 증명은 참고 문헌[8, 9]의 방법과 비슷하므로 생략한다.

정리 4. *식 13~16을 만족하는 해가 존재하는 것은* $\forall v \in V$ *에 대해 아래 식의 해가 존재하는 것의 필요충분조건이다.*

$$(r(v),t(v))=\begin{cases}(r_1(v),t_1(v)) & \text{if } t_1(v)\le T-bd(v)\\ (r_1(v)+1,t_2(v)) & \text{if } t_1(v)>T-bd(v), t_2(v)\le T-bd(v)\\ (r_1(v)+2, fd(v)) & \text{otherwise}\end{cases} \quad (27)$$□

X를 리타이밍 해 벡터, P를 X의 해 공간이라 할 때 식 24는 $X=F(X)$, $F: P\rightarrow P$ 형태이다. X의 부분순서, 하한값, 상한값을 4장 1절과 같이 정의하면 P는 CPO이다. P가 CPO이고 아래 정리 5에 의해 F는 단조함수이므로 반복적 방법으로 최소고정점을 구할 수 있다.

정리 5. *식 (27)을 나타내는 함수 F는 단조함수이다.*□

정리 5의 자세한 증명은 참고 문헌[8,9]의 방법과 비슷하므로 생략한다.

3. 래치 SoC 리타이밍 알고리즘

이진탐색으로 최소의 클락주기와 리타이밍 해를 구할 수 있다. 그림 5는 이진탐색 도중 호출되어 고정된 주기에서 해를 구하는 알고리즘이다. 1번 줄은 초기화 과정으로, 1로 초기화 되는 변수 flag는 주어진 주기에서 리타이밍 해가 존재하면 1, 그렇지 않을 경우 0의 값을 갖는다. Q는 갱신이 된 정점을 모아놓는 FIFO형태의 자료구조로 초기화시 *PI*에 속하는 정점을 넣는다. 2번 줄에서 Q가 비는 경우는 최소 고정점을 찾은 것이다. 3번 줄에서 10번 줄은 Q에서 나온 u와 인접한 정점 v의 x_v를 갱신시킨다. 4번 줄의 C0와 6번 줄의 C1~C3는 그림 6에 나와 있다. C0~C3은 식 24~27과 같은 역할을 하나 효율성을 위해 C0를 따로 떼어 놓았다. 9번 줄은 주어진 주기에서 리타이밍 해가 존재하지 않을 경우 무한반복을 피하기 위한 조건이다[9]. 여기서 maxFF는 $e \in E_a$에 속하는 모든 간선의 $w(e)$의 합이다.

```
input: G=(V,E) , T
output: flag, X
 1 flag=1, add PI into Q, X = X_L
 2 while ( Q is not empty and flag==1) do{
 3   u = extract(Q)
 4   update t(u) by C0
 5   foreach(v of (v,u)∈ E_f or (u,v)∈E ) do{
 6     update x_v by C1~C3
 7     if ( x_v is changed )
 8       add v into Q
     }
 9   if ( r(v)>0, ∀v∈PO or r(v)>maxFF, ∀v∈V )
10     flag = 0
   }
11 return flag
```

그림 5. 고정된 주기에서의 리타이밍 알고리즘
Fig. 5. Fixed period retiming algorithm.

```
C0: foreach (v of (v,u)∈E) {
    if(t(u)<t(v)+d(u,v)-(w(u,v)+r(u)-r(v))T)
        t(u)=t(v)+d(u,v)-(w(u,v)+r(u)-r(v))T }
C1: if ((v,u)∈E_f , w(u,v)==1){
    if ((r(v)<r(u)) (r(v),t(v)) = (r(u),fd(v)) }
C2: if( (u,v)∈E_f , w(u,v)==0 or (u,v)∈E_a){
  r_1(v)=max{ ⌈ (t(u)+d(u,v))/T-w(u,v)+r(u) ⌉ -1,r(u)- w(u,v) }
    t_1(v)=max{fd(v),t(u)+d(u,v)-(w(u,v)+r_1(v)-r(v))T}
    t_2(v)=max{fd(v),t(u)+d(u,v)-(w(u,v)+r_1(v)+1-r(v))T}
    if (t_1(v)≤T-bd(v)) (r_tmp(v),t_tmp(v))=(r_1(v),t_1(v))
    else if (t_2(v)≤T-bd(v)) (r_tmp(v),t_tmp(v))=(r_1(v)+1,t_2(v))
    else (r_tmp(v),t_tmp(v))=(r_1(v)+2,fd(v))
if ((r_tmp(v),t_tmp(v))≥(r(v),(v))){ (r(v),t(v)) = (r_tmp(v),t_tmp(v)) }
    }
C3: if ((v,u)∈E_f){
        if ((r(v)<r(u)) (r(v),t(v)) = (r(u),fd(v)) }
```

그림 6. x_v의 갱신 알고리즘
Fig. 6. Update x_v algorithm.

Ⅳ. 실 험

본 논문의 실험은 3.0Ghz 인텔 CPU와 1GB 메모리를 갖는 컴퓨터에서 실행되었고 모든 실험의 입력은 참고문헌 [12]의 방법과 유사하게 ISCAS-89 벤치마크를 수정하여 만들었다. 실험은 크게 두 종류로 나눌 수 있다. 첫째, 모든 IP 블락이 Complete Bipartite(CB)인 경우로 배치 과정 후 게이트를 블락으로 취급하였다. 칩의 크기를 4.3cm^{2}로 가정하고 와이어의 길이를 구한다. 각 와이어의 지연시간은 70nm 공정에서 2cm 와이어의 지연시간은 0.67ns라 가정하여[12] 구했다. 참고문헌 [8,9]에서는 ISCAS-89를 그대로 사용했지만 본 논문에서는 와이어에 파이프라이닝이 있는 경우를 위해 아래의 식으로 글로벌 와이어 $e \in E_a$의 $w(e)$값을 결정한다.

표 1. CB 블락
Table 1. CB block.

circuit			ff result		latch result		
circuit	maxFF	MCRP (0.01ns)	T (0.01ns)	runtime (s)	T (0.01ns)	runtime (s)	impr. (%)
p-s27	59	48.56	57.22	0.01	48.65	0.01	-14.98
p-s208	294	160.64	160.69	0.01	160.69	0.01	0.00
p-s386	797	80.38	86.13	0.03	80.44	0.02	-6.61
p-s400	601	61.80	70.62	0.03	62.83	0.01	-11.03
p-s420	537	99.57	101.9	0.02	99.65	0.01	-2.21
p-s444	620	85.02	95.69	0.02	85.09	0.01	-11.08
p-s820	1984	92.98	98.05	0.08	93.05	0.03	-5.10
p-s838	921	123.34	123.76	0.31	123.43	0.01	-0.27
p-s953	1327	82.08	98.72	0.11	92.14	0.05	-16.79
p-s1196	2012	72.27	89.56	0.16	73.87	0.09	-17.52
p-s1238	2059	77.50	90.47	0.11	77.56	0.08	-14.27
p-s1423	1600	144.41	144.5	0.17	144.50	0.06	0.00
p-s1494	3172	86.56	96.62	0.45	86.62	0.11	-10.35
p-s5378	4939	106.15	110.62	2.19	106.22	0.271	-3.98
avg							-8.16

표 2. NCB 블락
Table 2. NCB block.

circuit				ff result		latch result		
circuit	nPart	maxFF	MCRP	T	runtime (s)	T	runtime (s)	impr. (%)
p-s27	7	110	46.50	151.06	0.01	143.06	0.01	-5.30
p-s208	60	981	314.00	314.05	0.01	314.05	0.01	0.00
p-s386	90	2293	64.00	198.00	0.03	132.04	0.02	-33.31
p-s400	102	2032	251.00	251.09	0.02	251.09	0.02	0.00
p-s420	127	2263	297.00	297.09	0.03	297.09	0.03	0.00
p-s444	111	2309	81.00	245.03	0.02	202.01	0.01	-17.56
p-s820	213	5867	53.25	229.02	0.06	152.72	0.05	-33.32
p-s838	259	4468	251.00	251.08	0.08	251.08	0.06	0.00
p-s953	236	4597	276.00	276.06	0.11	276.06	0.08	0.00
p-s1196	314	6649	54.38	281.01	0.20	187.35	0.09	-33.33
p-s1238	316	6942	96.20	267.03	0.13	178.02	0.08	-33.33
p-s1423	382	7040	328.00	328.05	0.36	328.05	0.22	0.00
p-s1494	410	9961	86.56	284.07	0.27	210.05	0.13	-26.06
p-s5378	1411	23526	84.00	394.01	2.61	305.03	1.76	-22.58
avg								-14.63

$$w(e)=\left\lceil \frac{d(e)}{T_{init}}-1 \right\rceil+\alpha \tag{28}$$

여기서 T_{init}은 리타이밍 이전의 주기이지만 실험을 쉽게 하기 위해 0.33ns의 값으로 하였다. α는 지연시간과는 별도로 파이프라이닝시 회로의 기능을 유지하기 위해 삽입되는 메모리요소의 수를 나타내며 무작위로 0, 1, 2의 값을 갖는다. $e \in E_f$의 $d(e)$는 무작위로 0.25 ~ 0.50ns의 값을 할당하였다. 다음으로 일부의 블락이 Non-Complete Bipartite(NCB)인 경우를 실험하였다. 이를 위해 hMetis[13]로 ISCAS-89를 분할하고 같은 그룹에 속한 간선을 IP내부의 간선으로, 그렇지 않은 간선을 글로벌 와이어로 취급한다. 블락의 와이어는 단위 없이 25~50의 값을 갖고 글로벌 와이어는 20~500의 값을 갖는다. $e \in E_a$의 $w(e)$값은 T_{init}=33으로 하여 식 28을 사용해 결정하였다. 두 실험에서 액티브 구간의 길이 T_p는 $T/2$로 가정하였다.

위에서 소개한 두 가지 입력에 대해 FF를 사용한 경우[8,9]와 래치를 사용한 본 논문의 방법을 각각 적용하여 결과를 비교했다(표 1과 표 2). 각 표에서 'MCRP'는 회로내의 사이클의 지연시간과 메모리요소 수의 비로 클락 주기의 하한값이 된다[3,8]. 표 2의 'nPart'는 분할된 그룹의 수이다. 'impr.'는 참고문헌 [8,9]의 결과(ff result의 T)에 대해 본 논문의 결과(latch result의 T)가 향상된 정도이다. 표 1에서는 평균적으로 8%, 표 2에서는 평균 14% 주기가 줄었고 반면에 수행시간은 별 차이가 없다. 이는 본 논문의 방법이 참고문헌 [8,9]의 방법을 크게 수정하지 않고 적용할 수 있기 때문이다.

V. 결 론

본 논문에서는 FF을 사용한 SoC 리타이밍[8,9]을 래치를 사용한 SoC 리타이밍으로 확장시키기 위해 MILP식을 제시하고 이를 고정점 계산 방법으로 해결하였다. 실험 결과 같은 입력에 대해 참고문헌 [8,9]의 방법보다 평균적으로 약 10% 클락 주기를 줄일 수 있었고 수행시간에는 별 차이가 없었다.

참 고 문 헌

[1] V. Seth, M. Zhao, J. Hu, "Exploiting Level Sensitive Latches in Wire Pipelining," in Proc.ICCAD, pp. 283-290, 2004.

[2] C. E. Leiserson, F. M. Rose, and J.B.Saxe, "Optimizing synchronous circuitry by retiming," in Adv.Res. VLSI: Proc. 3rd Caltech Con., pp. 86-116, 1983.

[3] B. Lockyear and C. Ebeling, "Optimal retiming of level-clocked circuitry using symmetric clock schedules," IEEE Trans. on CAD , vol.13, no.9 pp.1097-1109, September 1994.

[4] N .Maheshwari and S. S. Sapatnekar, "Optimizing large multi-phase level clocked circuits," IEEE Trans. on CAD, vol.18, no.9, pp.1249-1264, September 1999.

[5] P.Saxena, P. Pan, C. L. Liu, "The retiming of single-phase clocked circuits containing level-sensitive latches," in Proc. VLSI Design , pp. 402-407, 1999.

[6] C. Chu, E. F. Y. Young, D. K. Y. Tong, and S. Dechu, "Retiming with interconnect and gate delay," in Proc.ICCAD, pp.221-226, 2003.
[7] C. Lin and H. Zhou, "Retiming for wire pipelining in system-on-chip," in Proc. ICCAD, pp.215-220, 2003.
[8] C. Lin and H. Zhou, "Wire retimng for system-on-chip by fixpoint computation," in Proc. DATE, , pp.1092-1097, 2004.
[9] C. Lin and H. Zhou, "Wire retimng as fixpoint computation," IEEE Trans. on VLSI, vol. 13. no.12, pp1340-1348, December 2005.
[10] K. A. Sakallah, T. N. Mudge, O. A. Olukotun, "Analysis and design of latch controlled synchronous digital circuits," IEEE Trans. on CAD, vol.11, no.3, pp.322-333, Mar. 1992.
[11] B. A. Davey and H. A.Priestley, "Introduction to Lattices and Order," Cambridge, 1990.
[12] V. Nookala, S. S. Sapatnekar, "A method for correcting the functionality of a wire-pipelined circuit," in Proc. DAC, pp.570-575, 2004.
[13] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar, "Multilevel hypergraph partitioning: Applications in VLSI domain," in Proc. DAC, pp. 526-529, 1997.

저 자 소 개

김 문 수(정회원)
2004년 서강대학교 컴퓨터학과 학사 졸업.
2006년 서강대학교 컴퓨터학과 석사 졸업.
2006년 현재 삼성전자 반도체 CAE 팀
<주관심분야 : CAD, VLSI 설계>

임 종 석(정회원)
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~현재 서강대학교 컴퓨터학과 교수.
<주관심분야 : 알고리즘, CAD, VLSI 설계>