지연시간과 회로 구조 변화를 고려한 증가적 타이밍 분석

오 장 욱† · 한 창 호††

요 약

본 논문에서는 허위 경로 문제를 해결하고 지연 시간 정보를 추출해내는 지연 시간 부울법을 이용하여 조합 회로에서 증가적 지연 시간 검사를 수행할 수 있는 방법을 제시한다. 내부 출력단에서 대치되는 내부 입력단의 히스토리를 작성하고 외부 출력단의 활성화 경로를 검사하여 최대 지연 시간을 구한다. 이때 외부 출력단의 히스토리를 참조하여 변형된 지연 시간을 적용시켜 다시 외부 출력단의 최대 지연 시간을 구할 수 있다. 이 방법으로 일단 외부 출력단의 근지연항의 합을 구하면 내부 회로의 지연 시간이 변하더라도 이미 구해 놓은 외부 출력단의 근지연항의 합으로써 빠르고 효율적으로 최대 지연 시간과 입력값을 추출해 낼 수 있다. 회로의 구조가 변경되었을 때 전체 회로를 다시 계산해야 할 필요는 없다. 전체 회로를 검사하여 변경된 구조의 영향을 받아서 다시 계산해야 할 필요가 있는 게이트를 선택하고 이 선택된 게이트만을 계산하여 부분적인 지연 시간 분석을 행할 수 있다. 이러한 증가적 지연 시간 분석은 회로의 지연 시간의 변화 뿐만 아니라 회로 구조의 변화를 고려하였고, 기존의 지연 시간 분석에 비해 회로 설계시 성능 시험 단계에서 생기는 시행 착오의 비용을 줄일 수 있다

Incremental Techniques for Timing Analysis Considering Timing and Circuit Structure Changes

Jang-Wook Oh† · Chang-Ho Han††

ABSTRACT

In this paper, we present techniques which perform incremental timing analysis using Timed Boolean Algebra that solves the false path problem and extracts the timing information in combinational circuits. Our algorithm sets histories of internal inputs that are substituted for internal output and extracts maximal delays through checking sensitizability of primary outputs. Once finding the sum of primitive delay terms, then it applies modified delay with referencing histories of primary output and it can extract maximal delays of primary outputs fast and efficiently. When the structure of circuit is changed, there is no need to compute the whole circuit again. We can process partial timing analysis of computing on the gates that are need to compute again. These incremental timing analysis methods are considered both delay changes and structure of circuit, and can reduce the costs of a trial and error in the circuit design.

※ 본 연구는 한국과학재단 특정기초연구비(96 0102-16-01-3) 지원으로 수행되었으며 지원에 감사드립니다.
† 준 회 원 : 인하대학교 대학원 전자계산공학과
†† 종신회원 : 인하대학교 전자계산공학과 교수
논문접수 : 1998년 11월 17일, 심사완료 : 1999년 7월 14일

1. 서 론

최근 몇 년 동안 반도체 기술의 진보는 집적 회로와 전자 시스템의 복잡성을 급격히 증가시켰다. 오늘날 집적회로는 몇 백 만개의 트랜지스터가 포함되고, 매초 수억 개의 클럭 주기에서 동작하도록 설계된다. 이러한 복잡성의 증가는 설계된 회로의 정확성을 검증하는 것을 더욱 더 어렵게 만든다. 급속도로 빠르게 증가하는 설계의 복잡성과 설계 시간을 줄이기 위해 설계 과정에서 중요한 설계 결정을 일찍 내릴 수 있게 도움을 주는 CAD 도구의 필요성이 생겼다. 여러 가지 회로 검증 방법으로는 기능적 검증(functional verification)과 물리적 검증(physical verification), 지연 시간 검증(timing verification or timing analysis)이 있다. 기능적 검증은 설계된 회로가 정확하게 동작하느냐를 검증하는 것이며, 물리적 검증은 레이아웃(layout) 단계에서 정확하게 구현될 수 있느냐의 검증이다. 이 중 지연 시간 검증(timing verification)은 설계된 회로가 원하는 속도로 동작하는가를 보장하는 방법으로 회로가 커지고 복잡해 짐에 따라 그 중요성이 점점 더 커지고 있다.

지연 시간 분석(timing analysis)은 부울 대수(Boolean algebra)를 이용하는 방법과 ATPG(Automatic Test Pattern Generator)를 이용하는 방법이 있다. 이 중 ATPG방법은 주어진 설계 회로에 대하여 고장 모델(fault model)을 가정하여 입력값을 자동으로 생성시켜 주는 방법이고 부울 대수를 이용하는 방법은 부울 함수의 연산을 통하여 디지털 논리 설계(digital logic design), 테스팅(testing)에서 많은 문제를 해결해 낼 수 있다. 부울 대수를 이용하는 방법 중 대표적인 것으로 이진 결정 다이어그램(Binary Decision Diagrams :BDDs)[1]이 있다. BDDs를 이용한 수많은 연구가 현재까지 계속 되고 있고 여러 장점을 가지고 있지만 BDDs는 ISCAS 85 벤치마크 회로[11]의 하나인 C6288과 같은 복잡한 회로를 표현하는 것이 불가능하다. 따라서 BDDs를 이용한 지연 시간 분석의 하나인 대수적 결정 다이어그램(Algebraic Decision Diagrams:ADDs)[2]도 또한 C6288의 검증이 불가능하다. 그 이유는 메모리가 지수적으로 증가하기 때문이다. 이외의 다른 증가적 지연 시간 분석을 가능하게 하는 방법[3]도 BDDs를 이용하기 때문에 이러한 단점이 있고, 회로의 지연 시간 변화만을 고려하여 회로의 구조적 변화가 오면 BDDs를 기반으로 한 논리 표현방식을 재구성하여야 한다. 이러한 재구성의 필요성은 증가적 지연 시간 분석법을 불가능 하게 한다.

본 논문에서는 부울 대수를 이용하는 방법 중 BDDs와 비교하여 C6288의 최대 지연 시간 추출이 가능하고 허위 경로 문제를 해결한 지연 시간 부울 대수(Timed Boolean Algebra)[4]에 기초하여 내부 회로의 지연 시간 변화뿐만 아니라 회로의 구조 변화가 있을 때 반복적인 지연 시간 계산과 활성화 경로 계산 없이 최대 지연 시간을 추출해 내는 증가적 지연 시간 부울법을 제안한다. 이는 테스트 대상 회로를 지연 시간 부울 대수에 따라 외부 출력단을 외부 입력단으로 표현하는 과정에서 히스토리(history)를 작성한다. 히스토리를 작성함으로써 외부 입력으로 표현된 외부 출력의 지연 시간을 변경하고자 하는 정보가 있기 때문에 증가적 지연 시간 분석(incremental timing analysis)이 가능하다. 그리고, 회로의 구조 변화가 있을 경우에는 다시 계산하여야 할 필요가 있는 구조를 선택하여 부분적으로 지연 시간 계산과 활성화 계산을 행한다.

본 논문의 2장에서는 관련 연구인 지연 시간 부울 대수의 간략한 설명 및 용어를 기술하고, 3장에서는 제안한 증가적 지연 시간 부울법을 기술하였다. 4장은 구현한 시스템의 실험 결과, 그리고 5장에서는 결론 및 향후 연구 과제를 기술한다.

2. 관련 연구

본 장에서는 본 논문에서 지연 시간 부울 대수(Timed Boolean Algebra)가 회로를 부울 함수로써 모델링 하는 방법과, 관련된 여러 가지 정의들에 대하여 알아 본다. 또한 부울 대수의 방법으로 어떻게 활성화 경로를 알아 낼 수 있는지 설명한다.

2.1 지연 시간 부울 대수

지연 시간 분석기의 주요 목적은 논리 네트워크의 임계 경로를 추출해 내는 것이다. 논리 네트워크의 경로 중 어떠한 입력 벡터(input vector)로도 절대 활성화 되지 않는 경로를 허위 경로(false paths)라 부른다. 반면에 다른 것들은 활성화 경로(sensitizable paths)라 한다.

본 논문의 근간인 지연 시간 부울 대수[4]는 지연 시간 분석에서 허위 경로 문제를 해결하는 방법이다. 지연 시간 부울 대수는 회로의 지연 시간을 모델링하

기 위한 지연 시간 연산자를 추가한 부울 함수이다. 지연 시간 부울 대수를 이용하여 각 논리 소자와 논리 네트워크의 동작을 지연 시간 연산자의 부울 함수로써 모델링 할 수 있다. 따라서 부울 함수의 대수적 조작과 지연 시간에 따른 모델의 계산에 의해서 유용한 지연 시간 정보가 추출 된다. 부울 함수를 이용하기 때문에 부울 대수의 교환(commutative), 조합(associate), 분배(distributive)법칙이 적용 가능하다. 이 방법은 허위 경로 문제를 해결하고 지연 시간 분석의 문제를 대수적 방법으로 다루며 상승, 하강 지연 시간을 각각의 내부 출력단에서 추출할 수 있다. 논리 소자의 상승, 하강 지연 시간은 대개 틀리기 때문에 논리 네트워크에서 최대 지연 시간을 추출하기 위해서 지연 시간 분석기는 활성화 경로뿐만 아니라 경로에 속하는 모든 노드(nodes)의 논리 상태도 알아야 한다.

2.1.1 정 의

지연 시간 부울 대수와 관련된 중요한 정의는 다음과 같다[5].

정의 1 : 지연 시간 부울 대수의 부울식 또는 부울 변수는 입력 데이타와 시간의 함수이다. I를 입력 데이타라 하고 시간 t에서의 식 B는

$$B : [I, t] \rightarrow \{0, 1\}$$

으로 표현한다.

정의 2 : 주어진 입력 데이터 I와 시간에서 식 B의 부울 값은 B[I,t]로 표현하고 지연 시간 연산자 ' , ' 를 삽입한 식 B는

$$([B, t])[I, t] = B[I, t - t_1]$$

으로 표현한다.

t_1은 지연 시간이고 I는 입력 벡터(vector)이다. 앞으로 사용할 지연항(delay term) $[B, t_1]$은 위의 식의 우변을 간결히 나타낸 것이다. 즉 B는 부울 변수이고 t_1는 그 부울 변수의 지연 시간을 의미한다. 입력 벡터에 따라 B의 상승, 하강 지연 시간을 t_1로 표현 할 수 있다[5].

정의 3 : 기본 연산자

a. '+'과 '×' : A와 B는 식이고 A+B는 A와 B에 OR 연산을 적용한 결과이고, A×B는 AND 연산을 적용한 결과이다. A+B와 A×B의 정의는 다음과 같다.

$$(A + B)[I, t] = A[I, t] + B[I, t]$$
$$(A \times B)[I, t] = A[I, t] \times B[I, t]$$

b. NOT 연산자 '!' : A는 식이고 !A는 A에 NOT 연산을 적용한 결과라 할 때 !A의 정의는 다음과 같다.

$$(!A)[I, t] = !(A[I, t])$$

지연 시간 연산자를 가진 모든 변수, 식, 함수는 각각 지연 시간 변수, 지연 시간 식, 지연 시간 함수라 하고 논리 네트워크의 외부 입력(primary input)으로 표현된 모든 부울 변수를 근원 지연 시간 부울 변수(primitive timed Boolean variable)라 한다. 지연 시간 부울 대수는 내부 출력과 외부 출력(primary output)을 대치 정리(substitution theorem), 최대 지연 시간 정리(maximal delay theorem), 부울 전파 정리(Boolean propagation theorem), 활성화 정리(sensitization theorem)을 이용하여 근지연항(primitive delay term)과 OR 연산자로 조합된 근지연항의 합으로 나타내어 활성화 계산을 하고 최대 지연 시간을 추출한다[4].

3. 증가적 지연 시간 부울법

회로 설계는 디버깅과 최적화 단계에서 대개 수십, 또는 수백번의 서로 다른 지연 시간에 대하여 분석된다. 따라서 반복적인 지연 시간 분석은 가능한 빠르게 하는 것이 매우 중요하다[3]. 증가적 지연 시간 분석은 설계가 조금 바뀔 때 전체를 재조사 할 필요가 없다는 사실을 이용한 것이다. 회로 설계자가 회로를 설계하고 성능 시험 단계에서 속도 향상을 원할 때 내부 출력단의 속도 변화를 가하고 지연 시간 분석을 다시 한다면, 변화된 출력이 외부 출력단에 영향을 미치는 것을 알 수가 없기 때문에 여러 번의 시행착오를 겪게 될 것이다. 따라서 증가적 지연 시간 분석은 이러한 과정을 일반적인 지연 시간 분석보다 좀 더 빠르게 회로 설계자가 성능 향상을 결정할 수 있도록 한다. 본 논문에서는 주어진 회로의 게이트의 지연 시간 값과 함께 구조의 변화를 고려하여 부분적 계산에 의한 지연 시간 추출법을 기술한다.

3.1 히스토리

본 논문에서 제안하는 히스토리는 근지연항이 상승 지연 시간의 영향을 받은 것인지, 아니면 하강 지연

시간의 영향을 받은 것인지 구별하기 위하여 출력단의 정보와 함께 그것의 정보도 저장한다. 히스토리는 지연 시간 부울 대수의 이론으로 부울 식이 전개 되면서 근지연항의 히스토리가 계속 추가된다. 따라서 히스토리는 근지연항의 경로에 대한 정보이다.

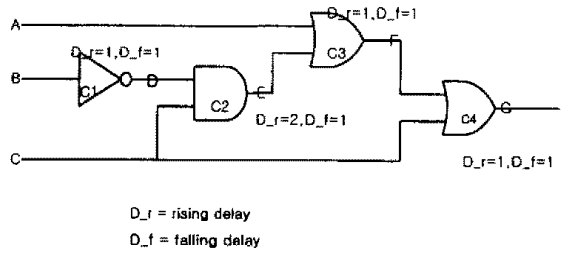
히스토리는 지연 시간 부울 대수의 이론이 적용되기 전, 모든 근지연항에 추가된다. 각 출력단이 근지연항의 합으로 나타날 때 해당 출력은 각각의 근지연항에 나타남으로써 근지연항이 어떠한 경로를 통하여 최대 지연 시간이 추출되었나를 히스토리를 참조하여 알 수 있다. 히스토리가 추가된 근지연항은 다음과 같다.

$$[B,\ d:H]$$

B는 근원 지연 부울 변수이거나 식이고 d는 최대 지연 시간 이론으로 계산된 근지연항의 지연 시간이다. H는 히스토리를 나타내고 ':'은 히스토리와 지연 시간을 구분한다. 히스토리는 출력단의 정보와 상승 또는 하강에 대한 정보를 가지고 있는 집합이므로 후에 설명할 히스토리 연산이 적용되지 않은 지연 시간 부울 식은 히스토리가 공집합이기 때문에 ':'과 히스토리를 생략한다. 히스토리를 추가하는 방법은 각 근지연항에 출력단을 가리키는 히스토리를 각 근지연항의 히스토리에 추가하는 것으로 각 근지연항이 그 출력단으로 전달되었다는 정보를 알 수가 있다. 식(3-1)은 지연 시간 부울 대수의 최대 지연 시간 이론을 승가적 지연 시간 부울법에 적용할 때 히스토리는 각 근지연항의 히스토리에 대한 합집합임을 표현한 식이다.

$$[B_1,\ t_1:H_1]\times[B_2,\ t_2:H_2]=[B_1\times B_2,\ MAX(t_1,\ t_2):H_1\cup H_2] \qquad \text{식(3-1)}$$

근지연항의 최대 지연 시간은 경로에 대한 시간 정보이므로 최대 지연 시간을 가지는 근지연항의 히스토리를 적용하는 것이다.



(그림 1) 예제 회로

(그림 1)의 예제 회로[4]에 따라 논리 네트워크(logic network)를 모델링(modeling)하면 다음과 같다.

$$D=[\overline{B},\ 1] \qquad \overline{D}=[B,\ 1]$$
$$E=[D,\ 2]\times[C,\ 2] \qquad \overline{E}=[\overline{D},\ 1]+[\overline{C},\ 1]$$
$$F=[A,\ 1]+[E,\ 1] \qquad \overline{F}=[\overline{A},\ 1]\times[\overline{E},\ 1]$$
$$G=[F,\ 1]+[C,\ 1] \qquad \overline{G}=[\overline{F},\ 1]\times[\overline{C},\ 1]$$

```
VOID Make_History(Node, Edge)
{
        hist = set_histset(Node) ; // 히스토리가 게이트를 가
               리키게 한다.
        Bool = Edge >true ; // true 근지연항을 가리킨다.
        while (Bool >next != NULL) { // 근지연항이 존재하
               는 동안
                if (Bool->histhead == NULL) { //근지연항
                                에 히스토리가 없으면
                        b->histhead = set_history(hist) ;
                                // 히스토리 삽입
                }
                else {
                        hist_union(hist) ; // 존재하는
                                히스토리에 더한다.
                }
                Bool = Bool->next ;
        }
        hist_union(hist) ; // 존재하는 히스토리에 더한다.
} // false에 대하여 같은 함수 적용.
```

(그림 2) 근지연항에 히스토리 정보를 삽입하는 알고리즘

히스토리는 각 부울 함수의 좌변을 그대로 참조한다. 논리 네트워크의 모델링이 끝나면 이를 히스토리와 함께 지연 시간 부울 대수의 이론을 적용하여 식을 전개하면 다음과 같다.

$D=[\overline{B},\ 1:D]$ $\qquad \overline{D}=[B,\ 1:\overline{D}]$

$E=[D,2:E]\times[C,2:E]$ $\qquad \overline{E}=[\overline{D},1:\overline{E}]+[\overline{C},1:\overline{E}]$

$=[\overline{B},3:DE]\times[C,2:E]$ $\qquad =[B,2:\overline{DE}]+[\overline{C},1:\overline{E}]$

$=[\overline{B}\times C,3:DE]$

$F=[A,1:F]+[E,1:F]$

$=[A,1:F]+[\overline{B}\times C,4:DEF]$

$\overline{F}=[\overline{A},1:\overline{F}]+[\overline{E},1:\overline{F}]$

$=[\overline{A},1:\overline{F}]+([B,3:\overline{DEF}]+[\overline{C},2:\overline{EF}])$

$=[\overline{A},1:\overline{F}]+[B,3:\overline{DEF}]+[\overline{A},1:\overline{EF}]\times[\overline{C},2:\overline{EF}]$

$=[\overline{A}\times B,3:\overline{DEF}]+[\overline{A}\times\overline{C},2:\overline{EF}]$

$G=[F,1:G]+[C,1:G]$

$=[C,1:G]+[A,2:FG]+[\overline{B}\times C,5:DEFG]$

$$\begin{aligned}\overline{G} &= [\overline{F},1:\overline{G}]\times[\overline{C},1:\overline{G}]\\ &= [\overline{C},1:\overline{G}]\times([\overline{A}\times\overline{C},3:\overline{EFG}]+[\overline{A}\times B,4:\overline{DEFG}])\\ &= [\overline{C},1:\overline{G}]\times[\overline{A}\times\overline{C},3:\overline{EFG}]+[\overline{C},\overline{G}]\\ &\qquad\times[\overline{A}\times B,4:\overline{DEFG}]\\ &= [\overline{A}\times B\times\overline{C},4:\overline{DEFG}]+[\overline{A}\times\overline{C},3:\overline{EFG}]\end{aligned}$$

히스토리는 활성화(sensitization) 계산에 아무런 영향을 끼치지 않기 때문에 지연 시간 부울 대수의 이론들을 그대로 적용할 수 있으며, 게이트 레벨(gate level)이 증가할 때마다 하나의 히스토리가 추가되므로 메모리 사용도 과하지 않고 추가적인 연산을 수행하지 않기 때문에 지연 시간 부울 대수보다 속도면에서 크게 떨어지지 않는다.

지연 시간 부울 대수는 절삭 임계값(cutting threshold)보다 많은 개수의 근지연항을 하나의 근지연항으로 대치한다. 그 이유는 근지연항의 합으로 나타나므로 근지연항 개수의 지수적 증가를 보이기 때문이다. 이 방법은 정확성이 떨어지는 단점이 있지만 각 입력단의 최대 지연 시간을 다음 출력단으로 정확히 전달하는 것을 보장한다[4].

예를 들어 AND게이트의 입력단 개수가 10개이고 각 입력단의 근지연항 개수가 2개라면 AND게이트 출력단의 근지연항 개수는 2^{10}개가 될 것이다. 절삭 임계값이 1000이라면 AND게이트 출력단의 근지연항 개수가 절삭 임계값보다 크므로 이것을 하나의 근지연항으로 치환한다. 수많은 근지연항을 하나의 근지연항으로 치환할 때, 치환되는 근지연항은 입력단의 최대 지연 시간이 전달되므로 치환되는 근지연항의 히스토리 또한 최대 지연 시간을 가지고 있는 근지연항의 히스토리를 가져야 한다.

$$P = \sum_{i=1}^{n}[P_i,\ d_i : H_i] \qquad \text{식(3-2)}$$

식(3-2)는 근지연항의 개수가 절삭 임계값보다 큰 근지연항의 합을 나타낸다. 이를 식(3-3)으로 대치한다.

식(3-3)에서 히스토리는 근지연항의 합의 모든 히스토리중에서 최대 지연 시간을 갖는 근지연항 히스토리들의 합집합이다.

$$P = \left[P,\ MAX_{i=1}^{n} d_i\ :\ \bigcup_{\forall H_i, d_{\max}} H_i\right] \qquad \text{식(3-3)}$$

최대 지연 시간을 추출하는 방법은 식(3-4)와 같다[4]. 히스토리는 활성화 계산에는 영향을 미치지 않으므로 생략하였다.

$$SENS([P_i,\ d_i]) = P_i \times \left(\sum_{\forall P_j, d_j < d_i} \overline{P_j}\right) = P_i \times \left(\prod_{\forall P_j, d_j < d_i} \overline{B_j}\right) \qquad \text{식(3-4)}$$

식(3-4)를 적용하여 (그림 1)의 예제 회로에서 외부 출력단 G의 활성화 계산은 다음과 같다.

$$G = [C,1:G] + [A,2:FG] + [\overline{B}\times C,5:DEFG]$$

위와 같이 G의 최대 상승 지연 시간을 알 수 있는 전개식에서 식(3-4)를 적용하면,

$$\begin{aligned}&SENS([C,1:G]) = C\\ &SENS([A,2:FG]) = A\times(\overline{C})\\ &SENS([\overline{B}\times C,5]) = \overline{B}\times C\times(\overline{A+C}) = 0\end{aligned}$$

따라서 G의 최대 상승 지연 시간은 2이고 입력 벡터는 반드시 $A\times(\overline{C})$를 참으로 하는 값이어야 한다[4]. 최대 하강 지연 시간도 이와 같이 행한다.

3.2 지연 시간 변화에 따른 증가적 활성화 계산

일단 근지연항의 합과 히스토리로 출력단이 구성되면 히스토리를 참조하여 근지연항의 지연 시간을 바꿀 수 있다. 지연 시간 부울 대수에서는 출력값은 근지연항의 합인데, 그 결과값으로는 내부 게이트의 정보가 저장이 안 되어 있으므로 어느 근지연항이 내부 게이트를 통하여 전달 되었는지 알 수가 없다. 따라서 내부 게이트의 지연 시간을 달리 하여 최대 지연 시간을 추출해 내고자 할 때 새로운 지연 시간에 대하여 처음부터 다시 실행 시켜야 한다.

증가적 지연 시간 부울법에서는 논리 네트워크를 모델링할 때 내부 게이트를 히스토리의 형식으로 각 근지연항에 저장하므로 내부 게이트의 변화를 근지연항에 직접 적용하는 것이 가능하다. 따라서 근지연항의 히스토리를 참조하여 변화가 생긴 게이트의 출력단을 포함하고 있는 근지연항의 지연 시간 값을 바꾸어 줄 수가 있다. 또한 히스토리는 경로의 내부 게이트의 정보와 함께 그것의 상승, 하강에 대한 정보를 가지고 있으므로 각각에 대하여 지연 시간을 바꾸어 볼 수 있다.

(그림 1)의 예제 회로에서 전개된 식 중 외부 출력단, G는 다음과 같다.

$$\begin{aligned}G &= [C,1:G] + [A,2:FG] + [\overline{B}\times C,5:DEFG]\\ \overline{G} &= [\overline{A}\times B\times\overline{C},4:\overline{DEFG}] + [\overline{A}\times\overline{C},3:\overline{EFG}]\end{aligned}$$

여기서 D게이트의 하강 지연 시간이 1만큼 줄였다면,

$$G=[C,1:G]+[A,2:FG]+[\overline{B}\times C,5:DEFG]$$
$$\overline{G}=[\overline{A}\times B\times \overline{C},3:\overline{DEFG}]+[\overline{A}\times \overline{C},3:\overline{EFG}]$$

으로 히스토리를 참조하여 D게이트의 하강 지연 시간 정보를 가지고 있는 근시연항을 찾아 지연 시간을 1만큼 줄인다.

바뀐 지연 시간 값을 적용하여 기존의 방법인 지연 시간 부울 대수[4]의 활성화 계산을 하면 새로운 최대 지연 시간을 추출할 수 있다. 지연 시간 부울 대수와 마찬가지로 최대 상승 지연 시간, 최대 하강 지연 시간의 추출이 가능하다. 변경된 근지연항의 활성화 계산은 지연 시간 부울 대수를 그대로 적용한다. 지연 시간 부울 대수에서 활성화 계산을 수행한 내부 출력은 정해진 지연 시간에 의하여 수행되고 허위항(false term)은 제거가 된다. 하지만 증가적 지연 시간 부울법에서는 허위항이 될지 참항(true term)이 될지 모르는 것이므로 최대 지연 시간을 추출할 출력단은 전처리 과정에서 허위항을 제거 하지 말아야 한다.

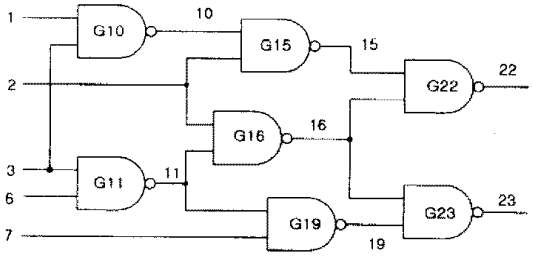
3.3 구조 변화를 고려한 증가적 지연 시간 분석

본 논문에서는 회로의 내부 지연 시간의 변화 뿐만 아니라 회로의 내부 구조의 변화를 고려한 증가적 지연 시간 분석을 고려하였다. 구조 변화는 내부 회로에 게이트를 추가하는 것으로 기존의 지연 시간 부울 대수로는 단 하나의 게이트가 추가되면 처음부터 다시 계산을 하여야 한다. 그 이유는 메모리의 지수적 증가 때문에 각 게이트의 근지연항의 합을 계속 가지고 있지 않기 때문이다. 따라서 입력 단자의 근지연항이 없기 때문에 추가된 게이트의 근지연항을 구할 수가 없다.

제안한 회로 구조 변화를 고려한 증가적 지연 시간 부울법은 추가된 출력단의 최대 지연 시간을 얻기 위하여 필요한 입력단들과 추가된 게이트의 출력단으로 연결된 외부 출력의 최대 지연 시간을 얻기 위하여 선택적으로 필요한 게이트만을 탐색한다. 탐색 된 결과로 회로의 각 게이트 중 필요한 게이트를 선택하여 선택된 게이트의 출력단의 근시연항을 구하여 최대 지연 시간을 추출한다.

본 논문에서 제안한 회로의 구조 변화를 고려한 증가적 지연 시간 부울법의 기본 내용은 지연 시간 부울 대수에 기반을 두고 있기 때문에 이번 절에서는 게이트의 선택에 대하여 예를 들어 설명한다.

ISCAS 벤치마크 회로 중 C17은 5개의 외부 입력과 2개의 외부 출력, 내부 게이트의 출력은 4개로 구성되어 있다. 이러한 C17회로에 (그림 3)과 같이 게이트 G15를 추가하였다고 하자.



(그림 3) 게이트 G15를 삽입한 변형된 C17회로

추가된 게이트 G15의 영향으로 G22의 출력의 근지연항의 합은 본래의 C17과 다르다. 따라서 다시 계산되어야 한다. 이를 일반화 하면, 추가된 게이트의 출력을 입력으로 하는 모든 게이트를 선택한다. 그리고 선택된 게이트의 출력을 입력으로 하는 그 다음 레벨의 게이트를 모두 선택한다. 이를 전방 탐색이라 하겠다.

```
VOID Check_Forward(Node) {
        e= n->out_array ; // Node의 출력단
        c = e->outlist ; // 출력단의 출력 게이트
        for ( ; c!= NULL ; c = c->next) { // 모
                든 출력 게이트에 대해서
            m = c->node ;
            if (m->count == 0 ) Insert_Touch(m)
                    ; // m이 탐색리스트에 삽입
                    되는 경우
            if (Is_M_Added) m->flag = 1 ; // m
                    이 추가된 것이면 선택
            if (n->flag == 1) m->flag = 1;//n이
                    선택된 것이면 m도 선택
        }
}
```

(그림 4) 전방 탐색 알고리즘

(그림 3)에서 G15에 영향을 받은 더 높은 레벨의 게이트인 G22를 선택한다. 전방 탐색은 더 이상 선택할 외부 출력이 없을 때 후방 탐색을 진행한다. 후방 탐색은 외부 출력에서 시작하여 외부 입력으로 끝나는 진행 방식인데 레벨이 가장 큰 게이트에서 시작하여 모든 게이트를 탐색하고 각 게이트가 선택된 게이트라면 그것의 모든 입력 게이트를 선택하는 작업이다. G22가

전방 탐색 과정에서 선택된 게이트이므로 G16게이트를 선택하고 같은 레벨인 G15게이트와 함께 선택한다.

이를 일반화 하면 현재 탐색하고 있는 게이트가 선택된 게이트라면 그것의 모든 입력 게이트를 선택한 후 순차적으로 선택된 모든 게이트의 입력 게이트를 다시 탐색한다.

(그림 3)의 예제회로에서 후방 탐색이 완료되면 G10, G11, G15, G16, G22가 선택된다.

G15게이트의 최대 지연 시간을 구하려면 10과 2가 필요하다. 15는 G22의 입력이므로 에지 22는 근지연항을 다시 생성해야 한다. G22는 16을 입력으로 하기 때문에 G16도 근지연항을 다시 생성해야 하고 G15도 마찬가지이다. 따라서 그것들의 입력은 모두 근지연항을 다시 생성해야 한다. 여기에서 G19와 G23은 선택되지 않음을 알 수 있다. 만약에 22와 23을 입력으로 하는 게이트가 존재한다면 G19와 G23은 선택되어 질 것이다. 이로써 선택된 게이트만을 부분적으로 지연 시간 부울 대수를 적용하여 새로운 최대 지연 시간을 추출할 수 있다.

```
VOID Check_Back(Node) { // Node : 현재 탐색 중
                          인 게이트
    numin = n->inpins - 1 ; // 게이트의 입력단의
                              개수
    for ( ; numin >= 0 ; numin-- ) {
        ee = Node->in_array ;
        e = ee[numin] ; // 게이트의 입력단
        c = e->inlist ; // 입력단의 입력 게이트
        for ( ; c != NULL ; c = c->next) { // 모
                  든 입력 게이트에 대하여
            m = c->node ;
            if (Node-> flag == 1) {
            // 현재 탐색 중인 게이트가 선택되어 진
               것이라면
                if (m-> flag != 1 ) {
                    m-> flag = 1 ;
            // 입력 게이트를 선택한다.
                    if (m->inserted != 1) Insert
                            _Touch(m) ;
            // 탐색 리스트에 더하기 전에 중복을 피한다.
                    m->inserted = 1 ;
                }
            }
        }
    }
}
```

(그림 5) 후방 탐색 알고리즘

4. 실험 결과

제안한 증가적 지연 시간 부울법은 C언어로 구현하였다. 실험은 주메모리 128MB의 삼보 Sparc1 Work station에서 ISCAS 85 벤치마크 회로에 대해 실험하였다. 지연 시간을 고려한 증가적 지연 시간 부울법을 실험하기 위해 임의의 내부 게이트를 변경하였고 절삭 임계값은 2000으로 설정하였다. 회로내의 모든 논리 소자의 지연 시간은 단위 지연(unit delay)으로 가정하였다.

4.1 지연 시간 변화의 실험 결과

기존의 지연 시간 부울 대수를 구현하고 이를 NIT (Non-Incremental Technique)라 한 실험 결과를 <표 1>에, 본 논문에서 제안한 방법의 실험 결과를 <표 2>에 나타내었다.

<표 1> 구현한 지연 시간 부울 대수의 실험 결과

CKT	Maximal Delay		NIT (seconds)	Memory (K bytes)
	Rising	Falling		
C432	17	17	3.49s	729
C499	11	11	1.97s	330
C880	24	24	202.16s	2361
C1355	24	24	27.07s	467
C1908	40	40	372.67s	4084
C2670	30	31	248.81s	3099
C3540	47	47	385.86s	9527
C5315	48	48	306.86s	9435
C6288	124	124	3844.28s	10968
C7552	42	42	608.76s	21487

전처리 과정은 단위 시간으로 계산하였다. <표 2>에서 나타나듯이 전처리 과정으로 외부 출력단을 근지연항의 합과 히스토리로 나타낼 수 있게 되면 100초 이내로 변화한 최대 지연 시간을 구할 수 있다.

메모리의 크기도 최대 50메가 바이트 이하이다. 전처리 과정에서 내부 출력단의 히스토리를 작성하는데 레벨에 따라 하나씩 증가하기 때문에 시간에 영향을 끼치지 않는다. 또한 외부 출력단은 지연 시간의 변화에 따라 어느 항이 참항인지 허위항인지 알 수가 없는 이유에서 활성화 계산을 하지 못하므로 <표 1>의 지연 시간 부울법(NIT)보다 전처리 과정이 느린 경우도 있다.

〈표 2〉 지연 시간의 변화를 고려

CKT	전처리 과정 (seconds)	증가기법적용 (seconds)	Memory (K bytes)
C 432	5.71s	0.03s	1100
C 499	2.81s	1.00s	1104
C 880	210.60s	16.20s	3679
C1355	23.62s	5.79s	6108
C1908	290.60s	51.51s	13076
C2670	141.45s	95.81s	6219
C3540	384.31s	17.77s	14592
C5315	221.04s	61.73s	45608
C6288	4050.85s	24.51s	26695
C7552	489.78s	64.03s	50696

4.2 회로 구조 변화의 실험 결과

실험 환경은 위와 동일하며 설삭 임계값은 500으로 설정하였고 ISCAS 85 벤치마크 회로에 임의의 게이트를 추가하였다. <표 3>은 구현한 지연 시간 부울 대수(NIT)와 BDD를 기반으로 하는 증가적 시간 분석[3]의 비교표이다.

〈표 3〉 증가적 지연 시간 분석의 비교

	BDD기반의 증가적 지연시간 분석	증가적 지연 시간 부울법
회로 구조 변화	불가능	가능
C6288의 증가적 지연 시간 분석	불가능	가능

5. 결 론

본 논문에서는 지연 시간 부울 대수를 기반으로 히스토리를 작성하고 그 히스토리를 참조하여 증가적 지연 시간 분석을 행할 수 있는 방법과 전방 탐색, 후방 탐색으로 추가된 게이트의 영향을 분석하여 부분적 활성화 계산을 수행할 수 있는 방법을 제안하였다. 제안한 방법은 회로 내부에서 게이트의 지연 시간 값이 변할 때 지연 시간 분석을 처음부터 다시 하는 것이 아니라, 빠른 시간 안에 원하는 내부 출력의 최대 지연 시간을 추출해 낼 수 있다는 장점이 있다. 또한 회로의 구조가 변경되어도 변경된 부분과 그것의 영향을 받는 게이트만을 선택하여 전체 회로를 재조사 하는 것이 아니라 부분적으로 재조사할 수 있게 하여 비교적 많은 시간 비용이 드는 지연 시간 분석을 좀 더 빠르게 할 수 있도록 하였다.

향후에는 지연 시간 부울 대수의 정확성 부족을 극복하고 추가된 게이트가 전체 회로에 영향을 미치는 경우를 대비한 좀 더 효율적인 증가적 지연 시간 분석법의 연구가 필요하다.

참 고 문 헌

[1] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation," IEEE Transactions on Computers, Vol.C-35, No.8, pp.677-691, Aug. 1986.

[2] R. I. Bahar, E. A. Frohm, C. M. Gaona, G. D. Hachtel, E. Macii, A. Pardo, and F. Somenzi, "Algebraic decision diagrams and their applications," the 30th DAC, pp.188-191, 1993.

[3] Y. C. Ju, and R. A. Saleh, "Incremental Techniques for the Identification of Statically Sensitizable Critical Paths," Proc., of the 28th DAC, 1991.

[4] S. T. Huang, T. M. Parng, and J. M. Shyu, "A new approach to solving false path problem in timing analysis," IEEE Int. Conf. Computer-Aided Design, pp.216-219, 1991.

[5] S. T. Huang, T. M. Parng, and J. M. Shyu, "A Polynomial-Time Heuristic Approach to Approximate a Solution to the False Path Problem," Proc. of the 30th DAC, pp.118-122, 1993.

[6] David H.C.Du, Steve H.C.Yen, and S. Ghanta, "On the general false path problem in timing analysis," Proc. 26th DAC, pp.555-560, 1989.

[7] David H.C.Du, Steve H.C.Yen, and S. Ghanta, "Efficient Algorithms for Extracting the K Most Critical Paths in Timing Analysis," in Proc. 26th DAC, pp.649-654, 1989.

[8] H.C. Yen, S. Ghanta, H.C.Du, "A path Selection Algorithm for Timing Analysis," Proc 25th DAC, pp.720-723, 1988.

[9] Patrick C. McGeer, Robert K. Brayton, "Timing Analysis in Precharge/Unate Networks," Proc. of 27th DAC, 1990.

[10] N. Jouppi, "Timing Analysis and Performance Improvement of MOS VLSI Designs," IEEE Transactions on Computer-Aided Design, CAD-6(no.4), pp.650-665, July. 1987.

[11] ISCAS-85 Benchmarks, Special Session: Recent Algorithms for Gate Level ATPG with Fault Simulation and Their Performance Assessment, IEEE International Symposium on Circuits and Systems, June 1985.

오 장 욱

e-mail : ohnova@nownuri.net

1997년 인하대학교 전자계산공학과 졸업(공학사)

1999년 인하대학교 대학원 전자계산공학과 졸업

관심분야 : 설계 자동화, 컴퓨터 그래픽스 등

한 창 호

e-mail : chhan@dragon.inha.ac.kr

1980년 성균관대학교 이공대학 전자공학과 졸업(공학사)

1982년 서울대학교 대학원 전자공학과 졸업(공학석사)

1991년 The Univ. of Texas at Austin Electrical Engineering Ph.D(공학박사)

1983년~1986년 한국전자통신연구소연구원

1991년~1992년 Cadence Design Systems,Inc., Senior Member of Technical Staff

1992년~현재 인하대학교 공과대학 전자계산공학과 부교수

관심분야 : 설계 자동화, 컴퓨터 그래픽스 등