論文94-31A-5-19

샘플링에 의한 시뮬레이션 결과의 압축

(Compression of Simulation Results by Sampling)

安 泰 均*, 崔 起 榮* *

(Taek yoon Ahn and Ki young Choi)

要 約

시뮬레이션은 현재 사용되고 있는 가장 효율적이고 널리 쓰이는 설계의 검증 방법이지만 모든 가능한 조합의 테스트 벡터에 대해서 시험해 보지 않으면 그 설계를 완전히 검증할 수 없다는 제약이 있다. 일반적으로 설계자는 일부의 테스트 벡터만으로 설계를 검증하는데, 회로가 커질수록 쓰이는 테스트 벡터는 많아지고 시뮬레이션의 결과에 대한 정보가 많아져서 그 정보를 효율적으로 처리하기가 어려워진다.

본 논문에서는 회로의 네트를 샘플링함으로써 시뮬레이션 결과를 압축시킬 수 있는 알고리즘을 제시한다. 시뮬레이션이 끝난 후 사용자가 저장되지 않은 네트 값을 알고자 할 경우 시뮬레이터가 incremental simulation technique을 응용하여 즉시 그 값을 재생성하게 된다. 몇 개의 회로로 15.7 MIPS 워크스테이션 상에서 실험해 본 결과, 충분히 빠른 시간 내에 재생성할 수 있으면서 10 정도의 압축률이 쉽게 얻어졌다. 이 방법을 이용하면 설계의 debug cycle time을 효과적으로 줄일 수 있다.

Abstract

It is very common in today's design practice to simulate a big design with a large set of test vectors thereby generating a huge set of data(simulation results) to be analyzed. As the design grows, the simulation results grow and become harder to be handled.

In this paper, we present algorithms for the compression and regeneration of simulation results. The compression is performed by sampling nets in a circuit. If the user wants to examine the lost part of the data, it is quickly regenerated by applying incremental simulation technique. Experimental results obtained for several practical circuits show that the compression ratio of 10 is easily obtained while maintaining a reasonably fast regeneration of data on a 15.7 MIPS workstation. Using the proposed method we can effectively reduce debug cycle time.

* 正會員, 서울大學校 電子工學科
(Dept. of Elec. Eng., Seoul Nat'l Univ.)
* * 正會員, 서울大學校 半導體 共同研究所
(Inter-University Semiconductor Research Center, Seoul National University)
※이 논문은 1993년도 교육부 학술 연구 조성비에 의하여 연구되었음.
接受日字 : 1993年 10月 9日

Ⅰ. 서론

설계된 회로의 정확한 동작 여부를 실제로 구현하기 전 단계에서 검증하는 방법으로서 시뮬레이션은 많은 각광을 받아 왔으며 또한 가장 많이 쓰이고 있다. 실제로 시뮬레이션이 많은 문제를 안고 있음에도 불구하고 널리 쓰이는 이유 중 가장 큰 것은 컴퓨터 상에서 회로를 모델링함으로써 그 회로가 구현되지 않았어도 그와 비슷한 효과를 낼 수 있고 구현된 칩보다도 다루기가 훨씬 용이하다는 것이다. 그러나 컴퓨터 상에 모델링되기 때문에 많은 양의 주기억장치 및 보조 기억장치를 차지하게 되고 시뮬레이션 도중에 또는 후에 그 결과를 저장하기 위해서 또한 많은 양의 보조 기억장치가 필요하게 된다. 더욱이 집적 기술이 발달함에 따라 회로는 더욱 커지고 또 그 회로를 검증하기 위한 테스트 벡터도 많아져야 한다. 만약 기억 장소가 충분하지 않다면 시뮬레이션 결과는 더 이상 저장되지 못하고 오랫동안 수행된 시뮬레이션은 허사가 되고 만다. 따라서 효율적인 시뮬레이션 결과의 저장 방법이 필요하게 되었다.

현재의 사용되는 시뮬레이터가 결과를 저장하는 방법은 두 가지로 요약될 수 있다. 그 중 한 가지는 시뮬레이션이 시작할 때부터 끝날 때까지 모든 네트 값을 저장하는 것이다. 이 방법은 많은 양의 기억 장소를 필요로 하는 대신 시뮬레이션이 끝난 후에 사용자는 어떤 네트이든지 그 결과를 볼 수 있다는 장점이 있다. 다른 한 가지는 시뮬레이션하기 전에 사용자가 특정한 네트를 선택하게 하고 시뮬레이션하는 동안 선택된 네트의 값만을 저장하는 방법이다. 이 방법은 기억 장소를 절약하는 대신 설계자가 선택하지 않은 네트의 값을 알고자 할 경우에는 처음부터 다시 시뮬레이션해야만 하는 단점이 있다.

본 논문에서는 공간 영역에서 데이터를 샘플링함[1]으로써 시뮬레이션 결과를 효율적으로 압축시킬 수 있는 방법을 제시한다. 샘플링은 회로 전체를 여러 개의 부분 회로로 나눈 후 각 부분 회로의 경계에 해당하는 네트의 값만을 저장함으로써 이루어진다. 시뮬레이션이 끝난 후 샘플링되지 않아서 저장이 안된 데이터는 빠른 시간 내에 재생성되어서 사용자가 불편하지 않도록 해야 한다. 본 논문에서는 빠른 재생성을 위해 incremental simulation [1,2] 에 사용되는 방법을 이용한다. 본 논문에서 제안하는 방법은 incremental change가 없는 회로에 대하여 원하는 네트 값을 빠른 시간내에 재생성하고자 하는 것으로 근본적으로는 incremental simulation과 다르나 재생성 방법은 비슷하므로 동일한 알고리즘을 사용할 수 있다.

2장에서는 재생성 알고리즘에 대해 설명하고 3장에서는 샘플링의 개념과 여러 가지 샘플링 방법에 대해 설명하고 그 장단점을 비교한다. 그리고 4장에서는 각 샘플링 방법의 성능 비교 등의 실험 결과를 보인다.

Ⅱ. 재생성 알고리즘

시뮬레이션이 끝난 후 샘플링되어 저장된 네트의 값으로부터 설계자가 원하는 임의의 네트의 파형을 만들어 내는 재생성 알고리즘은 비교적 간단하다. 그러므로 이를 먼저 설명하고 다음 장에서 구체적인 샘플링 알고리즘을 기술하겠다.

어떤 네트를 재생성하기 위해 걸리는 시간에 대한 성능을 측정하기 위해 다음과 같이 평균 가속비를 정의한다.

$$\text{평균 가속비} = \text{시뮬레이션 시간} \;/\; \text{평균 재생성 시간} \quad (1)$$

저장되지 않은 네트 값의 재생성은 시뮬레이션이 수행되기 전 샘플링 알고리즘에 의해 선택된 네트의 값에 기초하여 이루어진다. 즉 주어진 네트의 값을 재생성하고자 한다면 그 네트의 팬인 로직 콘(fanin logic cone)에 해당하는 회로만을 재시뮬레이션하여 그 네트의 값을 재구성하게 된다. 이는 Backsim [3] 의 개념과도 유사하다. 그러나 여기에서 팬인 로직 콘은 주입력(primary input)까지가 아니라 샘플링된 네트까지만 포함한다. 물론 주입력까지 거슬러 올

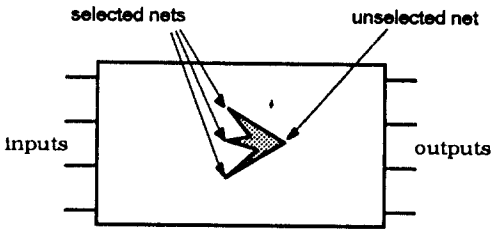

그림 1. 파형을 재생성하기 위해 재시뮬레이션되는 팬인 로직 콘

Fig. 1. Fanin logic cone to be resimulated to regenerate a waveform.

1) 공간영역에서 샘플링한다는 것은 회로상의 네트를 샘플링한다는 뜻으로 일반적으로 사용되는 시간 영역에서의 샘플링에 대비한 말

주입력까지 뻗게 된다. 그림 1은 그 예를 보여주는데 빗금 친 부분의 회로만을 재시뮬레이션하면 원하는 네트의 값을 얻을 수 있다. 본 논문에서 사용된 재생성 알고리즘은 incremental simulation [1] 의 incremental-in-space algorithm을 약간 수정한 것이다. 구체적인 알고리즘은 다음과 같이 역탐색, 초기 스케줄링, 재시뮬레이션으로 이루어진다

1. 역탐색

알고자 하는 네트가 샘플링되지 않았다면 그 네트의 파형을 재생성하기 위해 필요한 회로를 구분해 내어야 한다. 필요한 회로는 파형을 재생성해야 할 네트의 팬인 로직 콘이므로 그 네트로부터 입력 쪽으로 역탐색을 수행하여 구한다. 역탐색은 네트의 경우는 그 네트를 구동하는 엘리먼트로, 엘리먼트의 경우는 그 엘리먼트의 입력과 입출력 핀에 연결된 네트로의 탐색을 반복 수행함으로써 이루어진다.

역탐색을 해 나가면서 방문한 엘리먼트(element)는 INTERNAL로 표시한다. 역탐색 도중에 샘플링된 네트를 만나면 BOUNDARY로 표시하고 현재의 가지에 대한 탐색을 끝낸다. 탐색 도중 주입력을 만나는 경우에도 BOUNDARY로 표시하고 다른 가지에 대한 탐색으로 넘어간다.

2. 초기 스케줄링

역탐색이 끝나면 재시뮬레이션을 하기 전에 BOUNDARY로 표시되어 있는 네트에 저장되어 있는 값으로 스케줄링한다(BOUNDARY 네트는 모두 샘플링된 네트이거나 주입력이므로 값이 저장되어 있음). 이와 같이 스케줄 된 값들은 재시뮬레이션할 팬인 로직 콘의 테스트 벡터로 이용된다.

3. 재시뮬레이션

초기 스케줄링이 끝나면 보통의 이벤트 구동 시뮬레이션(event driven simulation)과 유사하게 시뮬레이션이 수행된다. BOUNDARY 네트로부터 발생되는 이벤트는 INTERNAL로 표시되어 있는 엘리먼트로만 전파되도록 한다. 매 시간 스텝마다 스케줄되어 있는 BOUNDARY 네트의 이벤트를 추출하여 이로부터 INTERNAL 회로를 시뮬레이션한다. 이때 INTERNAL 회로에서 생기는 이벤트도 물론 스케줄 대상에 포함시켜서 INTERNAL 회로 내부적으로는 원래의 시뮬레이션과 똑같은 이벤트들이 일어나고 똑같은 값으로 바뀌게 한다.

샘플링되지 않은 네트의 값을 재생성할 때 걸리는 시간은 전체 회로의 크기와는 무관하다. 위의 재생성 알고리즘에서 보듯이 재생성에 필요한 회로는 샘플링된 네트의 위치와 팬인의 크기, 그리고 재생성을 위한 팬인 로직 콘 내에 있는 각 엘리먼트를 계산하는데 걸리는 시간과 각 네트의 활동도 등에만 관련되어 있다. 따라서 회로가 커져도 재생성 시간은 증가하지 않지만 전체 회로를 시뮬레이션하는 데에는 많은 시간이 걸리게 되므로 평균 가속비는 그 만큼 증가한다.(식 1 참고)

Ⅲ. 샘플링 방법과 알고리즘

회로의 많은 네트 중 어떠한 네트를 샘플링하느냐에 따라 시뮬레이션 결과의 압축비가 높을 수도 있고 그렇지 않을 수도 있다. 또한 샘플링된 네트의 값으로부터 샘플링되지 않은 네트의 값을 재생성할 때 걸리는 시간도 샘플링을 어떤 방법으로 하는가에 달려있다. 따라서 샘플링은 기억 장소의 크기와 재생성 시간을 함께 고려해서 행해져야 한다.

샘플링 알고리즘의 성능은 우선 다음 식과 같은 평균 압축비에 의해 평가될 수 있다.

(평균 압축비)
= (시뮬레이션 결과의 총 크기) / (선택된 네트의 결과의 크기)
≒(회로의 총 네트 수) / (선택된 네트의 수) (2)

그러나 샘플링되어서 저장된 각각의 네트 값은 그 효용성이 다를 것이다. 어떤 네트는 그 값을 저장하는데 적은 장소가 필요하면서 다른 많은 네트의 재생성에 유용하게 쓰일 수도 있고 또 다른 네트는 활동도(activity)가 커서 많은 기억 장소를 차지하고도 다른 네트의 재생성에는 별로 영향을 못 끼치는 경우도 있다. 따라서 샘플링은 시뮬레이션이 끝나고 난 후 재생성을 얼마나 효율적으로 할 수 있을 것인지도 함께 고려해야 한다.

샘플링의 방법은 크게 두 가지로 나눌 수 있다. 그 중 한가지는 전체 회로 중에서 활동도가 작으면서 재생성 시의 효용성이 크다고 예상되는 네트를 임의로 선택하는 것이다.(그림 2) 그런데 재생성은 샘플링된 네트의 값을 기초로 이루어지므로 이 방법으로는 재생성 시 참조해야 할 네트의 위치가 일정하지 않다. 최악의 경우에는 재생성을 위한 팬인 로직 콘이 주입력으로부터 시작된다. 또 다른 방법으로는 그림 3과 같이 전체 회로를 두개 이상의 부분 회로로 나누어 각 부분 회로의 경계에 해당하는 네트를 선택하는 방법이다. 이렇게 하면 재생성할 네트로부터의 역탐색이 이 경계에서 완전히 차단된다. 따라서 재생성을

위한 팬인 로직 콘의 크기는 항상 두 인접한 경계 사이의 간격에 의해 제한되고 모든 네트의 재생성도 일정 시간 이내에 이루어진다고 보장할 수 있다. 이러한 이유로 본 논문에서는 두 번째 방법에 관한 알고리즘만을 기술하고자 한다.

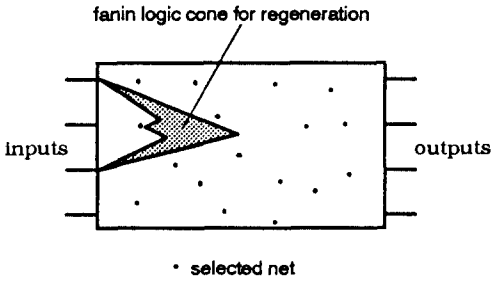

그림 2. 임의 선택
Fig. 2. Arbitrary selection.

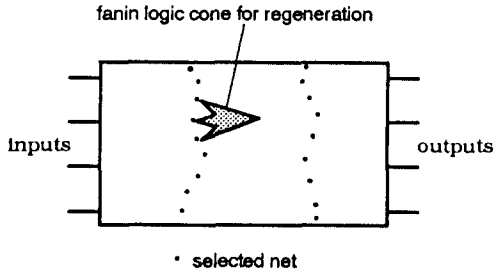

그림 3. 부분 회로에 의한 선택
Fig. 3. Selection by partitioning.

전체 회로를 여러 개의 부분 회로로 나누는 데에도 여러 가지 방법을 생각할 수 있는데 이러한 방법에 따라 평균 가속비와 평균 압축비가 차이 난다. 같은 평균 가속비에서 더 높은 평균 압축비를 보이거나 또는 같은 평균 압축비에서 더 높은 평균 가속비를 보일 때 그 샘플링 알고리즘의 성능이 높다고 말할 수 있다.

회로를 분할하는 방법 중 비교적 간단한 것은 회로를 단계화하여 그 단계에 따라서 부분 회로로 나누는 방법이다. 그리고 평균 압축비 또는 평균 가속비를 고려하여 성능을 향상시킬 수 있도록 회로를 분할하는 방법도 생각할 수 있다. 이제부터 이러한 방법에 관하여 구체적으로 기술하고자 한다.

1. 회로의 단계화

회로의 단계화에 의한 방법은 본 논문에서 기술하는 알고리즘 중 가장 간단한 방법이다. 먼저 각 회로의 엘리먼트는 주입력에서부터 출력을 따라가면서 단계가 정해진다. 주입력에 테스트 벡터 생성기가 있다고 가정하고 이 생성기의 단계를 0으로 정한다. 그리고 어떤 엘리먼트의 모든 입력 엘리먼트의 단계가 정해지면 그 엘리먼트의 단계 l은 각 입력 엘리먼트의 단계를 각각 l_1, l_2, ... l_i 라 할 때 다음과 같다.

$$l = \max(l_1, l_2, ..., l_i) + 1 \tag{3}$$

모든 엘리먼트의 단계가 정해지면 회로는 엘리먼트의 단계에 따라서 여러 개의 부분 회로로 나눌 수 있다. 여기에서 각 부분 회로의 단계 깊이를 L(단계 깊이 상수)이라 하면 각 부분 회로는 nL에서 $(n+1)L-1$ (n=0, 1, ...)까지의 단계를 갖는 엘리먼트들로 이루어진다.(그림 4) 그리고 각각의 부분 회로의 경계선에 위치해 있는 네트 즉, 서로 다른 부분 회로에 있는 엘리먼트를 연결하고 있는 네트가 샘플링되어 시뮬레이션 과정에서 저장된다.

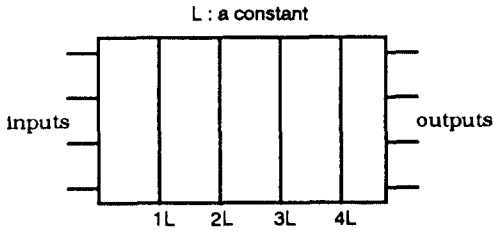

그림 4. 회로의 단계화에 의한 네트 선택
Fig. 4. Net selection by circuit levelization.

회로의 각 엘리먼트를 단계화하는 알고리즘의 복잡도는 $O(n)$이다. 따라서 빠른 시간 내에 샘플링을 수행할 수 있다.

2. Min-cut에 의한 방법

Min-cut에 의한 방법은 비용을 최소화하는 분할을 하여 여러 개의 부분 회로로 나누는 방법이다. 어떤 비용을 최소화할 것인가에 따라 여러 다른 방법으로 나뉘어질 수 있다. 첫 번째 방법은 압축 지향 방법으로 저장 장소의 크기를 비용으로 하여 목표를 평균 압축비의 향상에 둔다. 둘째 방법은 가속 지향 방법으로 재생성 시간을 비용으로 하여 평균 가속비의 향상에 목표를 둔다. 그리고 마지막 방법은 저장 장소의 크기와 재생성 시간을 모두 비용으로 하여 샘플링을 수행하는 방법으로 이 두 가지 목표 사이의 trade-off가 중요한 문제가 된다.

이 방법들은 모두 maximal flow-minimal cut theorem [4]에 의한 그래프 분할 알고리즘에 기초하고 있다. 이 알고리즘을 이용하면 어떤 방향성 그래프가 주어졌을 때 그 그래프를 소스(source)를 포함하는 한 부분과 싱크(sink)를 포함하는 다른 한 부분으로 나눌 수 있다. 그리고 잘려진 에지(edge)의 비용의 합은 최소가 된다. 가령 그림 5와 같은 그래프가 있을 때 이 그래프는 점선을 따라서 나뉘어진다. 이러한 알고리즘으로 Ford and Fulkerson algorithm [5], specialized primal simplex algorithms [6], Gomory and Hu algorithm [7], Dinic algorithm [8] 등이 있는데 본 논문에서는 비교적 구현이 쉽고 회로를 분할하는데 적합한 Ford and Fulkerson algorithm을 사용하였다. 이 알고리즘의 복잡도는 그래프의 버텍스(vertex)의 수를 V라 하고 에지의 수를 E라 했을 때 $O(|V|^2 |E|)$가 된다. [9]

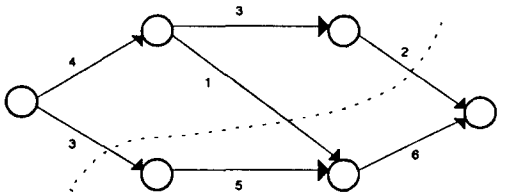

그림 5. maximal flow-minimal cut theorem에 의한 그래프의 분할

Fig 5. Graph partitioning by maximal flow-minimal cut theorem.

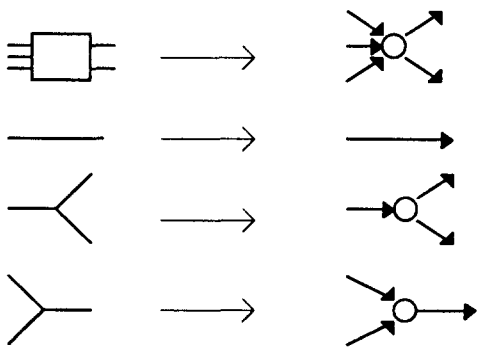

그림 6. 엘리먼트와 네트를 그래프로 변환하는 방법

Fig. 6. Transforming elements and nets to a graph.

이 알고리즘의 적용을 쉽게 하기 위해 회로를 그래프로 변환한다. 모든 엘리먼트는 버텍스가 되고 모든 네트는 버텍스와 에지의 집합이 된다.(그림 6) 그리고 각 에지마다 적당한 비용을 할당하면 그래프 분할 알고리즘에 의해 잘려진 에지의 비용의 합이 최소가 되도록 회로를 분할할 수 있다. 여기에서 에지의 비용을 무엇으로 정해 주는가에 따라 샘플링의 성능이 차이 나게 된다 샘플링을 수행하는 edge_sel( )이라는 함수는 그림 7과 같이 수행된다. compute_ each_ cost는 에지의 비용을 계산하는 함수의 포인터로서 샘플링 방법에 따라 가리키는 함수가 달라진다. 이상의 설명은 회로를 두 개의 부분 회로로 나누는 경우의 최적해를 구하는 알고리즘이다. 회로를 세 개 이상의 부분 회로로 나눌 때에는 해를 쉽게 구하기 위하여 heuristic을 사용한다. 즉 회로의 모든 엘리먼트를 단계화한 후 0에서 $2L$-1까지의 단계를 가진 엘리먼트로 이루어진 회로에 대해서 두 부분 회로로 분할한다. 그리고 두 부분 회로 중 주입력에 가까운 쪽의 부분 회로는 고정시키고 다른 쪽 부분 회로를 포함하여 $3L$-1까지의 단계를 가진 엘리먼트로 이루어진 회로에 대해서 다시 두 부분 회로로 분할한다. 이러한 과정을 주출력에 이를 때까지 계속하면 회로는 엘리먼트의 최대 단계를 L_{max}라 했을 때 $[L_{max}/L]$개의 부분 회로로 나뉘어진다. 그림 7의 source는 아직 샘플링이 수행되지 않은 출력 측 부분 회로에 대한 입력 네트를 저장하고 있는 변수이다. source는 최초의 분할에서는 주입력이 되고 두 번째 분할부터는 먼젓번의 분할에서 선택된 네트들이 된다. subcircuits()라는 함수는 새로운 source로부터 $(n+2)L$-1까지의 단계를 가진 엘리먼트로 이루어진 부분 회로를 돌려준다. min_cut( )라는 함수는 maximal flow-minimal cut theorem에 의한 알고리즘을 수행하여 잘려진 에지에 해당하는 네트를 돌려준다.

```
edge_sel(){
  source = primary_inputs();
  for(n=0; (n+1)L < max_level; n++){
    circuit = subcircuits(source, (n+2)L-1); /* nL에서 (n+2)L-1까지의 회로만 고려 */
    graph = transform_to_graph(circuit);     /* 회로를 그래프로 변환한다. */
    (*compute_each_cost)(graph, circuit);    /* 각 에지의 비용을 계산한다. */
    source = min_cut(graph, circuit);        /* 부분 회로를 분할한다. */
    mark_selected_nets();                    /* 샘플링된 네트를 표시한다. */
  }
}
```

그림 7. 샘플링 알고리즘

Fig. 7. Sampling algorithm.

1) 압축 지향 방법

샘플링 알고리즘의 성능을 평가하는 두 가지 기준 중에서 평균 가속비보다 평균 압축률을 높이는 방법

을 생각해 볼 수 있다. 즉 재생성 시간은 크게 차이가 나지 않는다고 가정하고 저장 장소를 줄여서 성능을 향상시키는 방법이다.

평균 압축률은 평균적으로 선택된 네트의 개수에 반비례한다. 그러나 각각의 네트마다 활동도가 다르기 때문에 선택된 네트의 개수만을 줄여서는 충분한 평균 압축률의 증가를 기대하기 힘들다. 따라서 실제 저장 장소의 크기를 줄이려면 선택된 네트의 활동도의 합을 줄이는 것이 바람직하다. 네트의 활동도는 샘플링을 하기 전에 미리 얼마간의 테스트 벡터를 입력하여 시뮬레이션하는 것으로 대략적으로 예측이 가능하다. 가령 1000개의 테스트 벡터를 입력해야 하는 경우 50개 또는 100개 정도의 테스트 벡터로 시뮬레이션한 후 샘플링을 수행하여 선택된 네트의 결과 값은 계속 유지하고 선택되지 않은 네트의 값은 버린다.

압축 지향 방법에 의한 샘플링 수행 과정을 그림 8에 pseudo code로 나타내었다. activity_of_net라는 함수는 네트의 현재 시뮬레이션 시간까지의 활동도를 돌려주는 함수이다. 따라서 회로를 그래프로 변환했을 때 각 에지의 비용은 그 에지에 해당하는 네트의 활동도가 된다.

```
sampled = FALSE;
compute_each_cost = activity_of_net;
for each test vector{
  simulate the test vector;
  if((# of simulated test vector)/(# of total test vector) > a
    && sampled == FALSE){      /* a는 0에서 1사이의 값 */
    edge_sel();
    sampled = TRUE;
  }
}
```

그림 8. 압축 지향 방법에 의한 샘플링
Fig. 8. Sampling by compression-oriented method.

2) 가속 지향 방법

시뮬레이션이 끝난 후 임의의 네트 값을 재생성할 때 걸리는 시간은 사용자가 큰 불편을 느끼지 않을 정도로 짧아야 한다. 샘플링할 때 평균 압축률보다는 평균 가속비에 중점을 둔다면 재성성 시간에서 큰 이익을 볼 수 있을 것이다.

평균 가속비를 향상시키기 위해서는 샘플링되지 않은 모든 네트의 재생성 시간을 예측하여 이 시간의 합이 최소가 되게끔 회로를 분할하여야 한다. 재생성 시간은 저장 장소 지향 방법에서의 활동도처럼 시뮬레이션하지 않고 회로의 구조로부터 대략적으로 예측한다. 이를 위해 다음과 같이 두 가정을 세운다.

가정 1: 시뮬레이션 시간은 주로 엘리먼트의 계산과 스케줄 작업에 걸리는 시간이 차지하며, 네 트의 계산과 그 스케줄 작업에 걸리는 시간은 엘리먼트의 계산 및 스케줄 시간에 비례 한다.

가정 2: 모든 엘리먼트의 계산 시간은 같고 입력 네트에서 이벤트가 발생할 확률도 같다. 전체 시뮬레이션 시간 중 각각의 엘리먼트가 계산되는데 걸리는 시간을 편의상 1로 정한다.

물론 실제로는 이 두 가정이 만족되지 않으므로 최적해는 구할 수 없지만 어느 정도 효과적인 해는 구할 수 있다. 이 두 가정 하에서 주어진 회로를 두 부분 회로로 분할하는 방법은 다음과 같다.

먼저 모든 네트의 비용을 계산한다. 각 네트의 비용은 그 네트가 선택되어 그 값이 저장되어 있다고 가정했을 때 그 네트의 팬인 로직 콘과 팬아웃 로직 콘 내의 모든 네트의 값을 독립적으로 재생성하는데 걸리는 총 시간 중에서 그 네트에 관련된 일부 시간이 된다. 이 때 팬아웃이나 팬인이 1보다 큰 경우에는 비용이 중복되어 계산되므로 네트의 비용 중에서 이에 관련되는 부분에 대해서는 팬아웃 수나 팬인 수로 나누어준다.

모든 네트의 비용이 계산되면 그래프 분할 알고리즘을 적용하여 회로를 분할한다. 분할된 두 부분 회로 사이의 경계를 이루는 모든 네트의 비용을 합하면 두 부분 회로 내부의 모든 네트 값을 재생성하는데 걸리는 시간이 된다. 따라서 이 비용의 합을 최소로 하는 분할, 즉 "minimal cut"을 해야 재생성 시간을 최소로 할 수 있다.

위에 언급한 바와 같이 각 네트의 비용은 팬인 로직 콘에 포함된 네트들의 값을 재생성하는데 걸리는 시간(팬인 비용)과 팬아웃 로직 콘에 포함된 네트들의 값을 재생성하는데 걸리는 시간(팬아웃 비용)의 합이 된다.

팬인 비용의 계산:

팬인 비용은 회로의 입력에서 시작하여 출력 쪽으로 각 네트를 방문하면서 계산한다. 방문한 네트의 팬인 비용은 그 네트의 팬인 로직 콘에 속한 모든 네트의 값을 독립적으로 재생성하는데 걸리는 시간이다. 주어진 네트의 값을 재생성하는 시간은 그 네트의 팬인 로직 콘에 속한 엘리먼트의 수로 계산된다.

피이드백이 없는 회로에서(피이드백이 있는 경우는 나중에 설명) 어떤 네트 N_i의 팬인 비용 $C_F(N_i)$를 구하는 방법은 다음과 같다. $C_F(N_i)$는 네트 N_i가 선택되었을 때의 비용이므로 네트 N_i를 재생성하기 위한

시간은 포함하지 않는다.(네트 Ni의 값은 저장되어 있으므로)

1) 모든 네트에 대해 팬인 로직 콘에 속하는 엘리먼트의 수를 구한다. 네트 Ni에 대한 이 수를 $C_I(N_i)$이라 하자.
2) 회로의 주입력에 해당하는 네트는 이미 선택이 되어 있다고 가정하여 $C_F(N_i) = 0$으로 한다
3) 그 외의 네트에 대해서는 네트 Ni의 모든 입력 엘리먼트 Ej의 모든 입력 네트 N_k, 즉 네트 N_i의 앞 단계에 있는 모든 네트 N_k에 대한 비용과 N_k의 팬인 로직 콘에 속하는 엘리먼트의 수를 모두 합한다. 그런데 E_j 또는 N_k의 팬아웃이 2 이상이라면 그 팬아웃 수로 나누어서 합한다. 즉 E_j와 N_k의 팬아웃을 각각 $O(E_j)$, $O(N_k)$라 하면

$$C_F(N_i)=\sum_{E_j \in input\ of\ N_i}\frac{\sum_{N_k \in input\ of\ E_j}\frac{C_F(N_k)+G_I(N_k)}{O(N_k)}}{O(E_j)} \qquad (4)$$

와 같이 $C_F(N_i)$을 구한다.

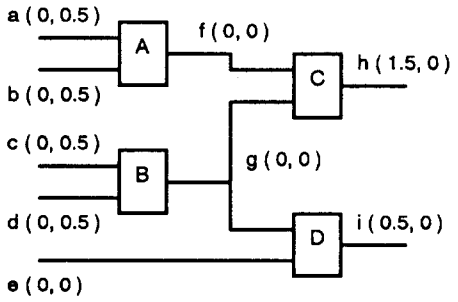

그림 9. C_F와 C_B 계산의 예

Fig. 9. An example for the computation of C_F and C_B.

위와 같은 방법으로 모든 네트에 대해 $C_F(N_i)$를 구하면 $C_F(N_i)$는 Ni의 팬인 로직 콘 내의 모든 네트를 재생성하는데 걸리는 시간을 중복됨이 없이 합한 값이 된다. 그림 9는 한 예를 보여준다. 각 네트의 괄호 안의 첫 번째 수는 네트의 팬인 비용 C_F이다. 주입력 a, b, c, d, e의 값은 모두 저장되어 있다고 가정하여 C_I를 모두 0으로 한다. 나머지 네트에 대해서는 앞 단계에 있는 네트의 비용과 팬인 로직 콘에 속하는 엘리먼트의 수를 합한다. 앞 단계에 있는 네트의 비용이란 앞 단계에 있는 네트의 팬인 로직 콘에 속하는 모든 네트를 재생성하는데 걸리는 시간이 되고, 앞 단계에 있는 네트의 팬인 로직 콘에 속하는 엘리먼트의 수는 그 네트를 재생성하는 시간이 된다. 따라서 두 값을 합하면 현재 네트의 팬인 로직 콘에 속하는 모든 네트를 독립적으로 재생성하는 시간이 된다. f의 비용은 a와 b의 비용을 합하고 여기에 a, b 각각의 팬인 로직 콘에 속하는 엘리먼트의 수를 합하여 구하는 데, 그 값은 0이 된다. 즉 f의 팬인 로직 콘에 속하는 네트 a와 b를 재생성하는데 걸리는 시간의 합은 0이다. g의 비용도 마찬가지로 계산할 수 있다. h의 비용은 f와 g의 비용을 합하고 f의 팬인 로직 콘에 속하는 엘리먼트(A)의 수, 즉 f의 값을 재생성하는데 걸리는 시간 1과 g의 팬인 로직 콘에 속하는 엘리먼트(B)의 수, 즉 g의 값을 재생성하는데 걸리는 시간 1을 더하면 2가 된다. 이 수도 역시 h의 팬인 로직 콘 내에 있는 모든 네트 a, b, c, d, f, g를 재생성하는데 걸리는 시간을 뜻한다. 그런데 네트 g의 비용과 g의 값을 재생성하는데 걸리는 시간은 h와 i의 비용을 계산할 때 각각 더해진다. 즉 엘리먼트나 네트의 팬아웃이 2 이상이면 그 팬아웃 수만큼 중복되어 더해지므로 팬아웃 수로 나누어서 각각의 다음 단계의 네트에 더해 준다. 그러면 최종적으로 h의 비용은 f의 비용 0, f의 값을 재생성하는데 걸리는 시간 1, g의 비용 0, g의 값을 재생성하는데 걸리는 시간 1을 2로 나눈 값 0.5를 모두 더하여 1.5가 된다. 같은 방법으로 계산하면 i의 비용은 0.5가 된다. 재집중 팬아웃(reconvergent fanout)이 있는 경우에는 경로가 분리될 때 나뉘어진 비용이 경로가 통합될 때 다시 합해지므로 전혀 문제가 되지 않는다. 이런 방식으로 계산하면 회로를 임의의 두 부분 회로로 나누었을 때 경계선에 해당하는 모든 네트의 $C_F(N_i)$의 합은 입력 측 부분 회로의 모든 네트의 재생성 시간을 합한 결과가 된다. 이 때 주출력(primary output) 네트의 비용을 모두 합하면 그 회로 내의 주출력 네트를 제외한 모든 네트의 값을 재생성할 때 필요한 시간이 된다. 그림 9에서 h와 i의 비용을 합한 값 2.0은 h와 i를 제외한 회로 내의 모든 네트를 재생성하는데 필요한 시간이다.

팬아웃 비용의 계산:

팬아웃 비용은 회로의 출력에서 시작하여 입력 쪽으로 각 네트를 방문하면서 계산한다. 네트의 비용을 구할 때는 엘리먼트의 팬아웃 로직 콘에 있는 모든 네트의 재생성에 그 엘리먼트가 한 번씩 사용된다는 사실을 고려한다. 피이드백이 없는 회로에서 Ni의 비용 CB(Ni)를 구하는 방법은 다음과 같다.

1) 모든 엘리먼트에 대해 팬아웃 로직 콘에 속하는 네트의 수를 구한다. N_i의 출력 엘리먼트 E_j 에

대한 이 수를 $G_O(E_j)$이라 하자. 단, 주출력 네트는 $G_O(E_j)$에서 제외된다.

2) 회로의 주출력에 해당하는 네트는 이미 선택이 되어 있다고 가정하여 $C_B(N_i) = 0$으로 한다.

3) 그 외의 네트에 대해서는 네트 N_i의 모든 출력 엘리먼트 E_j의 모든 출력 네트 N_k, 즉 N_i의 다음 단계에 있는 모든 네트 N_k에 대한 비용을 N_k의 팬인으로 나눈 수와 E_j의 팬아웃 로직 콘에 속하는 네트 중 주출력 네트를 제외한 네트의 수를 모두 합한다. 그런데 E_j의 팬인이 2 이상일 때에는 그 팬인 수로 나누어서 합한다. Ej와 Nk의 팬인을 각각 $I(E_j)$, $I(N_k)$라 하면

$$C_B(N_i)=\sum_{E_j \in output\ of\ N_i}\frac{\sum_{N_k \in input\ of\ E_j}\frac{C_B(N_k)}{I(N_k)}+Go(E_j)}{I(E_j)} \qquad (5)$$

의 식에 의해 $C_B(N_i)$를 구한다.

그림 9에서 각 네트의 괄호 안의 두 번째 수는 네트의 팬아웃 비용 C_B를 나타낸다. 주출력인 h와 i의 비용은 이미 선택되어 있다고 가정하므로 0이 된다. f의 비용은 f가 선택되었을 경우를 가정하면 h의 비용 0과 f의 값이 영향을 미치는 선택되지 않은 네트(여기서는 없다)의 수 0의 합을 C의 팬인 2로 나눈 수 0이 된다. g와 e의 비용도 마찬가지로 구하되 g의 비용은 양쪽 팬아웃에서의 계산되는 비용을 더한다(역시 0이 된다). 그런데 a의 비용은 f의 비용 0과 a의 값이 영향을 미치는 선택되지 않은 네트(f 밖에 없다)의 수 1의 합을 A의 팬인 2로 나눈 수 0.5가 된다. b, c, d의 비용도 마찬가지 방법으로 구하면 모두 0.5가 된다. 이러한 방법으로 회로를 두 부분 회로로 나누었을 때 샘플링되는 네트의 비용 $C_B(N_i)$을 모두 합하면 출력 측 부분 회로의 모든 네트를 재생성하기 위해 필요한 시간의 합이 된다. 또한 주입력의 $C_B(N_i)$를 합하면 주입력을 제외한 회로 내 모든 네트를 재생성하기 위해 필요한 시간의 합이 된다. 그림 9에서 주입력 a, b, c, d, e의 비용을 모두 합한 값 2.0은 주입력을 제외한 회로 내의 모든 네트를 재생성하는데 필요한 시간이 된다.

회로 내에 피이드백이 있는 경우에는 피이드백 네트를 모두 선택한다. 피이드백 루프 내의 엘리먼트는 전체 회로를 시뮬레이션 할 때에 피이드백 네트의 이벤트에 따라서 한 테스트 벡터에 대해 두 번 이상 계산될 수 있다. 따라서 피이드백 네트가 선택되어 이벤트가 저장되어 있으면 피이드백 루프 내의 엘리먼트가 한 번만 계산되어 팬인 로직 콘이 피이드백 루프를 포함하는 네트의 재생성 시간을 줄일 수 있다. 피이드백 네트가 선택되면 그 값이 저장되므로 피이드백 네트에 의해 생기는 다른 네트의 추가 비용은 없다. 그러므로 피이드백 네트를 회로에서 제거한 후 피이드백 루프가 없는 회로와 마찬가지로 위의 방법을 적용하여 C_F와 C_B를 쉽게 구할 수 있다 모든 네트에 대해 C_F와 C_B의 값이 계산이 되면 네트의 최종 비용 C는 다음과 같이 계산된다.

$$C(N_i)=C_F(N_i)+C_B(N_i) \qquad (6)$$

회로의 단계에 따라 $C_F(N_i)$와 $C_B(N_i)$가 변하는 경향을 그림 10와 같이 나타낼 수 있다. 회로를 그래프로 변환했을 때의 에지의 비용을 $C(N_i)$로 하여 그래프 분할 알고리즘을 적용하면 샘플링된 네트의 C_F의 합은 입력 측 부분 회로의 모든 네트 값을 재생성할 때 걸리는 시간이 되고 C_B의 합은 출력 측 부분 회로의 모든 네트 값을 재생성할 때 걸리는 시간이 된다. 따라서 이 두 비용의 합을 최소화하여 회로를 분할하면 전체 네트 값을 재생성할 때 걸리는 시간을 최소화할 수 있다. 그림 9의 예에서는 회로는 f, g, e를 경계로 분할된다.

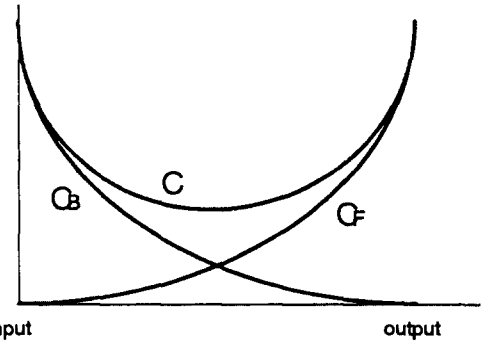

그림 10. 엘리먼트의 단계에 따른 $C_F(N_i)$, $C_B(N_i)$, $C(N_i)$

Fig. 10. $C_F(N_i)$, $C_B(N_i)$ and $C(N_i)$ to the element level.

3. 압축과 가속을 같이 고려한 방법

저장 장소 또는 재생성 시간을 따로 고려하여 회로를 분할하게 되면 그 성능이 크게 높아지지 않는다. 샘플링 알고리즘은 평균 압축비와 평균 가속비를 함께 높여야만 성능 향상을 기대할 수 있다. 저장 장소만을 줄이려 하면 각 부분 회로의 크기가 균일하지 않기 때문에 재생성 시간이 많이 걸리게 되고, 재생성 시간만을 줄이려 하면 샘플링되는 네트의 수가 많아지던가 활동도가 높은 네트들이 샘플링되어 평균

압축비가 떨어진다. 따라서 저장 장소와 재생성 시간을 같이 고려하여 회로를 분할하는 것이 바람직하다. 저장 장소와 재생성 시간을 같이 고려한 네트의 비용 $T(N_i)$는 네트 N_i의 활동도를 $A(N_i)$라 하고 총 네트의 개수를 n이라 했을 때 다음과 같다.

$$T(N_i)=\frac{C(N_i)}{\frac{1}{n}\sum_{i=1}^{n}C(N_i)}+\alpha\frac{A(N_i)}{\frac{1}{n}\sum_{i=1}^{n}A(N_i)} \tag{7}$$

위 식에서 α의 최적 값은 회로에 따라 달라지므로 여기에서는 대부분의 회로에 대하여 공통적으로 적절하다고 생각되는 값을 실험적으로 구하여 사용한다.

Ⅳ. 결과와 성능 평가

앞장에서 설명한 샘플링과 재생성 알고리즘은 UNIX 환경 하의 SUN SPARCStation에 구현하였다. 그리고 시뮬레이터 코어(simulator core)는 THOR [10, 11] 라는 혼합 레벨 시뮬레이터를 썼다. 이 장에서는 여섯 개의 회로에 대한 시뮬레이션 결과를 저장하는데 필요한 기억 장소의 크기, 재생성하는데 걸리는 시간, 그리고 샘플링 방법에 따른 성능의 비교에 대한 실험 결과를 기술한다. 필요한 주기억 장소의 크기는 기존의 시뮬레이터와 차이가 없다. 실험에 사용된 회로는 네 개의 ISCAS'85 benchmark 회로, pipelined 16X16 곱셈기, special purpose 시그날 프로세서(SETI)인데, ISCAS '85 benchmark 회로와 곱셈기는 모두 combinational circuit이며 게이트 단계로 기술되어 있고 SETI는 행위 단계로 기술되어 있다. 각 회로의 크기에 대한 정보가 표 1에 나타나 있다.

표 1. 실험에 사용된 회로
Table 1. Circuits used in the experiments.

회로	네트 수	엘리먼트 수	주입력 수	주출력 수
c5315	2485	2308	178	123
c7552	3719	3513	207	108
c2670	1426	1194	233	140
c880	443	384	60	26
곱셈기	1387	1356	32	33
SETI	1405	548	82	22

구현된 시뮬레이션 시스템에서는 사용자가 시뮬레이션을 수행하기 전에 여러 샘플링 방법 중 한가지를 선택하고 나뉘어질 부분 회로의 단계 깊이 상수 L을 줄 수 있다. 여기에서는 샘플링 알고리즘으로 회로의 단계화에 의한 방법(방법 1)과 압축과 가속을 같이 고려한 방법(방법 2)을 써서 두 결과를 비교하였다. 실험에서 쓰인 테스트 벡터는 모두 랜덤하며 그 수는 2000개이다. 활동도 예측을 위한 α값은 0.1, 즉 200개의 테스트 벡터에 대해서 활동도를 예측하였다. 그리고 방법 2에서의 재생성 시간 비용에 대한 저장 장소 비용의 비는 1로 하였다.

평균 압축비는 같은 L 값에 대해서라도 회로의 형태, 주어진 테스트 벡터 등에 따라 달라진다. 표 2에 L=10일 때의 각 샘플링 방법에 대한 평균 압축비를 나타내었다. 평균 압축비는 대체로 주어진 L 값과 비슷하게 나올 것이라고 예상되지만 여러 부분 회로에 걸쳐 있는 네트 때문에 방법 1에서는 L보다 낮은 평균 압축비를 나타내고 방법 2에서는 활동도를 이용하여 저장 장소를 고려하기 때문에 L보다 높은 평균 압축비를 나타낸다.

표 3에서는 L=10으로 샘플링하여 결과를 저장한 후 네트 값을 재생성할 때 걸리는 평균 시간을 각 샘플링 방법에 대해 나타내었다. 모든 회로에 대해 평균 재생성 시간은 수 초 미만이다. 그러나 이 값은 주어진 테스트 벡터의 수에 따라 달라지게 된다. 방법 2에서는 재생성 시간을 고려하였지만 저장 장소를 같이 고려한 때문에 대체로 방법 1보다 많은 시간을 필요로 한다 표 4에서는 L=10일 때 샘플링 알고리즘이 수행되는 시간을 기록하였다. 이 알고리즘은 처음 시뮬레이션할 때에만 필요하며 재생성 시에는 필요 없으므로 시간에 대한 오버헤드는 없다. 비교적 간단한 방법 1에서는 모두 0.2 초 이내의 짧은 시간 내에 샘플링이 수행되지만 방법 2를 수행하는 데는 수 초 또는 수 십 초 이상의 많은 시간이 걸린다. 따라서 네트의 값을 재생성할 필요가 별로 없는 경우에는 빠른 시간 내에 샘플링이 가능한 방법 1을 쓰고 많은 네트의 값을 재생성해야 할 필요가 예상되는 경우에는 높은 성능을 필요로 하므로 방법 2를 써서 처음 시뮬레이션하는 시간보다는 재생성 시간에서 이득을 보는 것이 바람직하다. 방법 2의 샘플링 알고리즘이 수행되는 시간은 전체 회로의 시뮬레이션 시간(표 3)에 비하여 5~20% 정도가 된다. 그러나 이 실험에서 쓰인 2000개의 테스트 벡터는 실제 시뮬레이션에서 쓰기에는 크게 부족한 수이고 또한 테스트 벡터의 수는 시뮬레이션되는 회로의 주입력 수에 따라 지수 함수 적으로 증가하는데 비해 방법 2의 샘플링 알고리

즘을 수행할 때 가장 많은 시간이 걸리는 Ford and Fulkerson Algorithm의 복잡도는 앞에서 밝혔듯이 $O(|V|^2 |E|)$이므로 실제의 시뮬레이션에서는 시뮬레이션 시간에 대비한 방법 2의 샘플링 알고리즘 수행 시간이 무시할 수 있을 정도가 될 것이다.

그림 11에 각 회로에 대한 샘플링 방법의 성능을 수평축을 평균 압축비로 하고 수직 축을 평균 가속비로 하여 비교하였다. 그림에서 보는 바와 같이 방법 2가 방법 1보다 같은 압축비에 대해서 높은 가속비를 나타낸다. SETI는 행위 단계로 기술되어 있기 때문에 재생성 시간 지향 방법에서의 모든 엘리먼트의 계산 시간이 같다는 가정과 많은 차이가 있다. 따라서 이 경우 방법 2의 성능이 다른 회로에 비해 비교적 떨어지지만 여전히 방법 1보다는 좋은 성능을 보이고 있다 평균 압축비와 평균 가속비에 의한 성능 평가에서는 회로가 커질 수록 평균 가속비가 증가하여 더 좋은 성능을 나타낸다. 그림 11에서와 같이 비교적 큰 회로인 c5315와 c7552에서는 10정도의 압축비에서 100내지 200정도의 평균 가속비를 보이는데 반하여 비교적 작은 회로인 c880에서는 20내외의 평균 가속비를 보인다. 이는 같은 구조의 16bit, 32bit, 64bit 곱셈기에 대한 실험 결과를 비교한 그림 12에서도 잘 나타나 있다. 16bit(32bit) 곱셈기에 비하여 32bit(64bit) 곱셈기는 회로 크기가 대략 4배 정도가 되며 시뮬레이션 시간도 4배 정도로 증가하지만 평균 재생성 시간은 별로 차이가 없기 때문에 같은 평균 압축비에 대해서 평균 가속비는 4배 정도로 증가한다. 이는 회로가 커질수록 평균 가속비가 증가함을 보여 준다.

표 2. 샘플링 방법에 따른 평균 압축비
Table 2. Average compression ratios for the two sampling methods.

회로 / 방법	c5315	c7552	c2670	c880	곱셈기	SETI
1	5.22	4.98	6.89	7.36	5.13	3.63
2	11.24	16.26	17.52	10.96	10.57	10.79

네트의 비용에 압축을 더 많이 고려하느냐 또는 가속을 더 많이 고려하느냐에 따라 샘플링 알고리즘의 성능이 다르다. 그림 13에서는 곱셈기에 대해서 α의 값을 달리했을 때 그 성능이 변하는 것을 보여준다. α가 크면 압축이 더 많이 고려되어 높은 압축비를 얻을 수 있으나 가속비는 떨어져서 그래프는 오른쪽 아래에 위치하고 α가 작으면 높은 가속비를 얻는 반면 낮은 압축비를 나타내어 그래프는 왼쪽 위에 위치하게 된다. 그림 13에서 보듯이 곱셈기에서는 α=1일 때의 성능이 대체로 높게 나타나는 것을 알 수 있다. 다른 회로에 대해서는 최적의 α값이 달라지지만 대체로 1 정도의 값이 가장 높은 성능을 보였다.

표 3. 샘플링 방법에 따른 평균 가속비
Table 3. Average accelaration ratios for the two sampling methods.

회로 / 구분-방법	c5315	c7552	c2670	c880	곱셈기	SETI
평균 재생성 시간-1(a1)	3.1s	3.2s	3.2s	1.3s	2.9s	2.9s
평균 재생성 시간-2(a2)	3.6s	4.0s	3.0s	1.5s	3.6s	4.9s
시뮬레이션 수행시간(b)	340.9s	578.7s	141.9s	28.2s	327.3s	115.3s
평균 가속비-1(b/a1)	110.0	180.8	44.3	21.7	112.9	39.8
평균 가속비-2(b/a2)	94.7	144.7	47.3	18.8	90.9	23.5

표 4. 샘플링 알고리즘 수행 시간
Table 4. Time taken by sampling algorithm.

회로 / 방법	c5315	c7552	c2670	c880	곱셈기	SETI
1	0.14s	0.18s	0.07s	0.02s	0.14s	0.13s
2	48.91s	96.10s	14.26s	1.63s	21.78s	14.38s

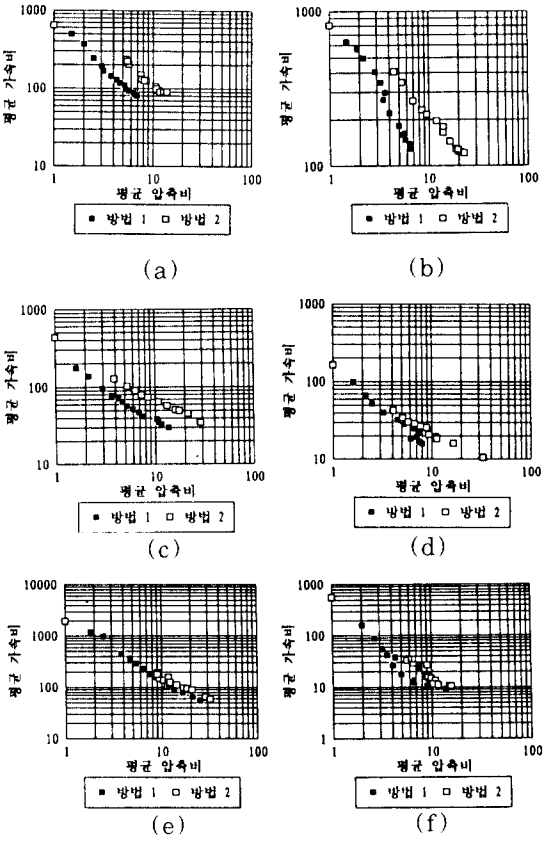

그림 11. 샘플링 방법의 성능 비교 (a)c5315 (b) c7552 (c)c2670 (d)c880 (e)곱셈기 (f)SETI

Fig. 11. Comparison of performances of the sampling methods (a)c5315 (b)c7552 (c) c2670 (d)c880 (e)multiplier (f)SETI.

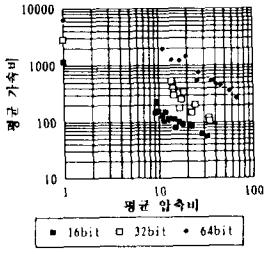

그림 12. 회로의 크기에 따른 성능 비교(곱셈기)

Fig. 12. Performance comparison for different circuit size(multiplier).

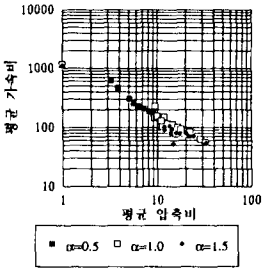

그림 13. α값의 변화에 따른 성능 비교(곱셈기)

Fig. 13. Performance comparison for different values of (weighting factor).

V. 결론 및 추후 과제

시뮬레이션 결과를 샘플링하여 저장하고 빠른 시간 내에 저장되지 않은 결과를 재생성하는 기능은 기존의 결과 저장 방법보다 대략 1/10정도의 기억 장소만으로도 같은 효과를 낼 수 있다. 앞으로 점점 시뮬레이션해야 할 회로가 커짐에 따라 더욱 많은 결과를 저장하기 위해 더 많은 기억 장소가 필요해질 것이다. 더구나 회로를 보다 정확히 검증하기 위해서는 많은 테스트 벡터에 대해서 결과를 볼 수 있어야 한다. 샘플링에 의한 저장 방법은 그러한 기억 장소 문제를 해결해 주고 빠른 시간에 시뮬레이션 결과를 보여줄 수 있으므로 시뮬레이터에 그 기능이 부가되었을 때 매우 효과적으로 시뮬레이터의 기능을 향상시킬 수 있다.

본 논문에서는 주어진 회로를 여러 개의 부분 회로로 나누어 그 경계에 해당하는 네트를 샘플링하여 샘플링된 네트의 결과만을 저장하는 방법을 말하였다. 그리고 저장된 결과로부터 빠른 시간 내에 샘플링되지 않은 네트의 파형을 재생성할 수 있는 방법을 제시하고 그 결과가 만족할 만함을 보였다. 기억 장소의 크기는 회로의 크기, 이벤트의 발생 빈도, 그리고 테스트 벡터의 수에 비례한다. 재생성 시간은 회로의 크기와는 무관하고 또한 이벤트의 발생 빈도와 팬인 수가 비슷하다면 주어진 단계 깊이 상수에 따라 일정하다는 것을 알아보았다.

아직은 min-cut에 의한 방법을 수행하는 데는 많은 시간이 걸리지만 실제 시뮬레이션에서 쉽게 이 방법을 이용하게 하기 위해 샘플링 알고리즘 수행 시간을 줄이는 작업이 진행 중이다 압축 및 재생성 알고리즘은 통상의 시뮬레이션 시스템에 적절히 통합될 수 있다.앞으로 이러한 기능을 좀더 사용자에게 편리하게 확장하여 적절히 활용할 수 있도록 할 것이다.

參 考 文 獻

[1] Kiyoung Choi, *Incremental Approach to Digital Simulation*, Ph.D dissertation, Stanford University, Stanford, CA, June 1989.

[2] S. Y. Hwang, T. Blank, K. Choi, "Fast Functional Simulation: An Incremental Approach," *IEEE Trans. Computer-Aided Design*, vol. 7, pp. 765-774, July 1988.

[3] S. P. Smith, M. R. Mercer, and B. Brock, "Demand driven simulation: Backsim," *Proc. 24th ACM/IEEE Design Automation Conference*, pp. 181-187, June 1987.

[4] Dantzig, G. B., and D. R. Fulkerson, "On the Max-Flow Min-Cut Theorem of Networks," in H. W. Kuhn and A. W. Tucker (eds.), *Linear Inequalities and Related Systems*, Annals of Mathematics Study No. 38, Princeton University Press, Princeton, N.J., pp. 215-221, 1956.

[5] Ford, L. R., and D. R. Fulkerson, "Maximal Flow Through a Network," *Canadian Journal of Mathematics*, vol. 8, pp. 399-404, 1956.

[6] Glover, F., D. Klingman, J. Mote, and D. Whitman. "A Primal Simplex Variant for the Maximum-Flow Problem," *Naval Research Logistics Quarterly*, vol. 31, pp. 41-61, 1984.

[7] Gomory, R. E., and T. C. Hu. "Multi-Terminal Network Flows," *SIAM*, vol. 10, 1961.

[8] Dinic, E. A., "Algorithm for Solution of a Problem of Maximum Flow in a Network with Power Estimation," Soviet Math. Dokl., vol. 11, pp. 1277-1280, 1970.

[9] Edmonds, J., and Karp, R. M., "Theoretical Improvements in Algorithm Efficiency for Network Flow Problems," J. ACM, vol. 19, pp248-264, 1972.

[10] R. Alverson, T. Blank, K. Choi, S. Y. Hwang, A. Salz, L. Soule, and T. Rokicki, *THOR User's Manual: Tutorial and Commands*, Tech. Rep. CSL-TR-88-348, Stanford University, Stanford, CA, 1988.

[11] R. Alverson, T. Blank, K. Choi, S. Y. Hwang, A. Salz, L. Soule, and T. Rokicki, *THOR User's Manual: Library Functions*, Tech. Rep. CSL-TR-88-349, Stanford University, Stanford, CA, 1988.

著者紹介

安 泰 均(正會員)

1969年 5月 17日生. 1991年 한국과학기술원 전기 및 전자공학과(공학사). 1993年 서울대학교 전자공학과(공학석사). 1993年 ~ 현재 서울대학교 전자공학과 박사과정 재학중. 주관심 분야는 CAD, computer architecture 등임.

崔 起 榮(正會員)

1955年 8月 30日生. 1978年 서울대학교 전자공학과 졸업. 1980年 한국과학원 전기 및 전자공학과 석사. 1989年 미국 Stanford 대학 전기공학과 박사. 1978年 ~ 1983年 (주)금성사 중앙연구소 근무. 1989年 ~ 1991年 미국 Cadence Design System, Inc 근무. 1991年 ~ 현재 서울대학교 반도체공동연구소 및 전자공학과 조교수. 주관심 분야는 CAD, VLSI 설계 등임.