분할기법에 의한 효율적인 3논리값 시뮬레이션

오 상호*, 조 동균*, 강 성호*
* 연세대학교 공과대학 전기공학과

Efficient 3-Value Logic Simulation Using Partitioning

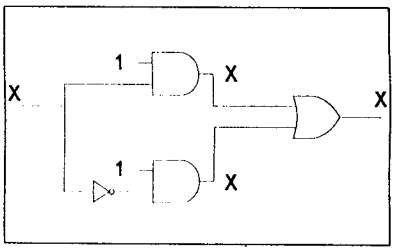
Sangho Oh, Dongkyoon Cho, Sungho Kang
Dept. of Electrical Eng. Yonsei Univ.

영문요약

Logic simulation is playing a very important role for design verification as circuits are larger and more complicated. However unknown values in 3 value simulators may generate the X-propagation problem which makes inaccurate output values. In this paper, a new partitioning method and a new simulation algorithm are developed to deal with the X-propagation problem efficiently. The new algorithm can optimize simulation time and accuracy by controlling partition depth. Benchmark results prove the effectiveness of the new simulation algorithm.

1. 서론

논리 시뮬레이션(logic simulation)[1]은 회로를 설계하고 검증하는데 매우 중요한 역할을 하고 있으며 특히 회로의 크기가 방대해지고 복잡해짐에 따라 설계과정에서의 시뮬레이션의 정확성과 속도는 더 강조되고 있다. 좋은 회로 시뮬레이터는 실제회로에서 출력되는 정확한 값을 예상할 수 있어야 하지만 회로에서 발생하는 미지값(unknown value)은 그림1과 같은 X값 전파(unknown propagation 또는 X propagation) 문제를 발생시켜 정확한 결과를 얻어내는데 큰 장애로 작용하게 된다[2].

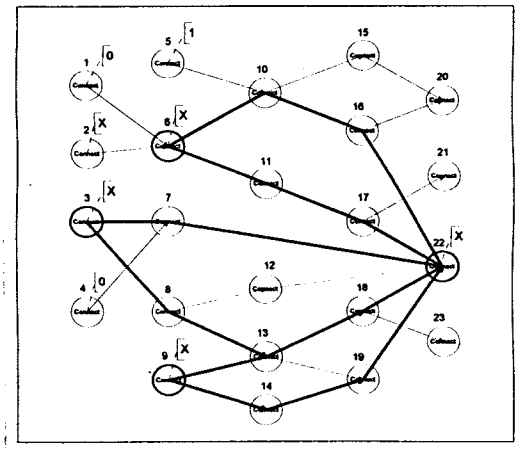


<그림 1> X값 전파 문제

미지값을 처리하는 3논리값 시뮬레이션 방법에는 미지값과 그 보수(complement)를 구분하는 방법[3]과 회로에 대한 카르노 맵(Karnaugh map)을 작성한 다음 여분주함축소 (redundant prime implicant)를 찾아내어 추가함으로써 정확하게 미지값을 처리하는 방법 등이 있지만, 시뮬레이션 정확도에 명확한 한계가 있거나 근본적으로 NP-complete 문제[4]를 갖고있어 실제 회로에 적용하는데 어려움이 많다는 것이 밝혀져있다. 본 논문은 X값 전파 문제를 효과적으로 처리하기 위해 분할기법[5]을 사용하였으며, 분할기법을 통하여 회로에서 발생하는 미지값을 정확하고 효율적으로 처리하였다.

2. 분할기법

게이트가 많은 회로의 시뮬레이션에서 미지값을 정확하게 처리하기 위해서는 회로를 부분 부분으로 잘라서 분석하는 분할(partition) 방법이 유용하게 작용하는데, 이렇게 회로를 부분 부분으로 자르고 각 부분에 대한 정확한 시뮬레이션을 통해 전체 회로에서의 정확한 출력값을 구하는 시뮬레이션 방법이 분할기법이다. 다음 그림 2는 어떤 회로에서의 분할을 보여주고 있다. 분할은 어떤 게이트의 출력의 개수가 둘 이상일 때 이루어지며, 갈라져 나간 출력이 다시 어느 한 게이트에서 모이게 될 때 분할은 끝나게 된다. 회로에서 이러한 분할을 이룰 수 있는 부분은 많지만, 실제 시뮬레이션에서 의미가 있는 분할은 정확하게 처리할 수 있는 미지값이 발생하는 경우인 분할의 시작 게이트와 끝 게이트의 값이 미지값인 경우로 한정된다.



<그림 2> 3개의 미지 입력을 갖는 분할

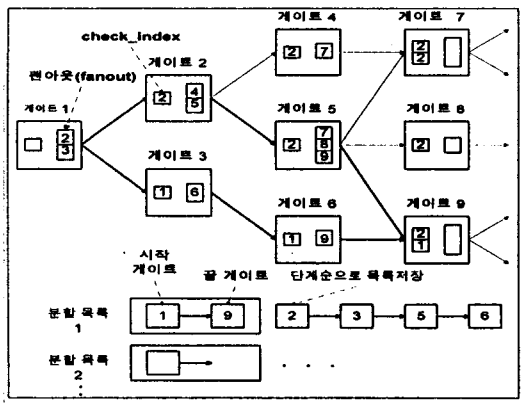
분할이 결정되고 나면 분할목록(partition_list)을 만들어 저장하고 일반모드에서 3논리값 시뮬레이션을 실시한 후 그 결과가 미지값으로 나온 경우에 한해 부분분석 시뮬레이션을 실행하는데 부분분석 시뮬레이션은 어떤 분할부분에 대해 가능한 모든 경우의 미지값을 대입해보고, 분할부분의 출력이 한가지 값으로 고정되는지 살펴본 후 그 출력값을 0이나 1의 특정한 값으로 고정시키는 것이다. 모든 분할에 대해 부분분석 시뮬레이션을 실행한 후 전체를 다시 일반모드로 시뮬레이션하면 정확한 출력값을 얻을 수 있다.

3. 알고리즘

본 분할모드 시뮬레이션 프로그램의 전체 알고리즘을 그림 3에 그리고 분할 찾기 알고리즘과 분할목록의 예를 그림 4와 그림 5에 나타내었다. 분할 찾기 알고리즘을 살펴보면 먼저 게이트에 대하여 깊이고정 트리탐색(depth fixed tree search)을 실시하고 check_index를 팬아웃의 개수로 초기화시킨 후 첫 번째 팬아웃(fanout)에 대한 트리탐색에서는 check_index를 저장하면서 일정한 깊이까지 탐색하고, 두 번째 팬아웃에 대한 탐색부터는 check_index를 1씩 감소시켜 탐색을 수행하면서 만일 탐색 도중 게이트에 저장된 값이 check_index보다 크면 분할이 찾아진 것이다. 분할이 발견되면 분할 목록에 분할이 이루어진 게이트의 목록(list)을 단계순으로 추가시킨다.

```
main( )                          // partition-mode simulation
{
  parse();                       // parse the input circuit
  levelize();
  find_partition();
  optimize_partition();
                                 // merge partitions
  while (get_input_value())
  {  simulate();                 // normal-mode simulation
     simulate_partition(int &suspected, int detected);
                                 // partition-method simulation
     simulate();                 // accurate results
  }
}
```

<그림 3> 분할모드 시뮬레이션의 전체 흐름



<그림 4> 분할깊이 3인 분할 찾기와 분할목록의 예

```
ALGORITHM : find_partition()
(1) temp = head->next
(2) for (num_fanout of gat[temp->index] > 2) do this
  (2.1) next = gat[temp->index].next
  (2.2) num_fanout = gat[temp->index].num_fanout
  (2.3) test_partition(next->gate, num_fanout, depth_limit)
  (2.4) next = next->next
  (2.5) num_fanout--
  (2.6) if (next != NULL) goto (2.3)
(3) tmep = temp->next
(4) if (temp != NULL) goto (2)
(5) return

ALGORITHM :test_partition(index, check_index, depth_limit)
(1) if (gat[index].value < check_index)
    gat[index].value = check_index      // initialize
(2) if (gat[index].value = check_index)
    return                              // ignore
(3) if (gat[index].value > check_index)
    add_partition_List(index)
    return                          // add to partition-list
(4) for (depth_limit > 0) do the following
  (4.1) next = gat[index].next;
  (4.2) gat[next->gate].sp[next->index] = index
  (4.3) test_partitoin(n->gate, check_index,--depth_limit)
                                    // recursive routine
  (4.4) next = next->next
  (4.5) if (next !=NULL) goto (4.2)
(5) return
```

<그림 5> find_partition() 알고리즘

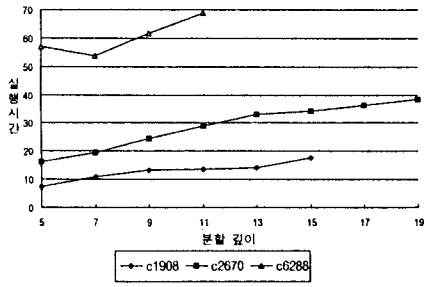
4. 분할 방법의 적용 결과

본 시뮬레이션은 ISCAS 85 벤치마크 회로[6]을 대상으로 100Hz Pentium칩에서 시뮬레이션했으며 메모리는 600KB로 제한하여 분할 깊이와 메모리의 양의 상관 관계를 밝혔다. 다음 표 1은 단순한 3논리값 시뮬레이션인 일반모드의 시뮬레이션과 분할모드 시뮬레이션의 실행시간 차이와 정확하게 처리되는 미지값의 개수를 비교한 것이다. 입력값은 3분의 1이 미지값을 포함하도록 임의로 선정된 1000개의 입력패턴이 입력되었고 분할 깊이는 10으로 고정하였다.

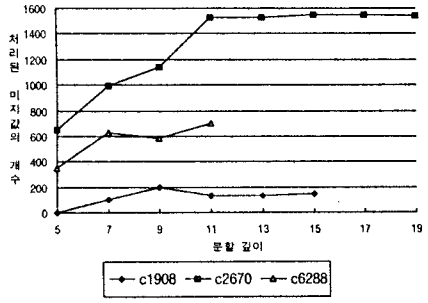
<표 1> 일반모드와 분할모드의 실행시간 비교

회로	일반모드 [초]	분할모드 [초]	분할/일반	처리된 미지값
c432	0.71	5.65	7.96	116
c499	0.98	18.07	18.44	242
c880	1.64	7.41	4.52	279
c1355	2.25	15.65	6.96	140
c1908	3.13	13.02	4.16	184
c2670	5.82	27.91	4.80	1504
c3540	6.09	41.81	6.87	2507
c5315	10.38	58.24	5.61	3544
c6288	10.32	68.35	6.62	699
c7552	15.44	57.58	3.73	1912

그림 6과 7은 분할깊이에 따른 실행시간의 지연과 처리되는 미지값의 개수를 비교한 것으로 분할 깊이가 깊어짐에 따라 실행시간이 선형적으로 증가함을 알 수 있다. 하지만 분할의 깊이가 깊어짐에 따라 처리되는 미지값의 개수가 증가하기는 하지만 선형적으로 증가한다고 할 수 없는데 이는 c2670의 예에서 볼 수 있듯이 분할 깊이 11에서 이미 정확하게 처리 할 수 있는 미지값의 거의 대부분이 정확하게 처리되었기 때문에 더 이상 정확하게 처리할 수 있는 미지값이 존재하지 않기 때문이다.



<그림 6> 분할깊이와 실행시간의 관계



<그림 7> 분할 깊이와 처리된 미지값 개수

마지막으로 표 2는 입력되는 미지값의 개수와 실행시간의 상관관계를 밝힌 것으로 입력되는 미지값의 수가 적어짐에 따라 일반모드의 시뮬레이션 시간은 거의 변함이 없지만 분할모드 시뮬레이션의 시간은 상당히 줄어듦을 알 수 있다. 이는 적은 미지값을 갖는 시뮬레이션에서 분할모드 실행시간이 줄어드는 본 시뮬레이터의 특성을 말해주는 것으로 미지값이 출력된 경우에만 분할 시뮬레이션을 하기 때문에 적은 미지값을 갖는 시뮬레이션에서 상대적으로 빠르게 동작하는 특징이 있다.

<표 2> 미지값의 비율과 실행시간의 관계

회로 / 미지값 비율	c2670		c5315		c6288	
	일반모드	분할모드	일반모드	분할모드	일반모드	분할모드
1/3	5.82	27.91	10.38	58.24	10.32	68.35
1/7	5.27	19.01	10.16	40.05	10.16	54.89
1/11	5.21	15.76	10.05	33.02	10.16	48.62
1/15	5.21	13.46	10.05	29.06	10.10	41.64
1/19	5.21	12.19	10.01	27.08	10.05	40.43

5. 결론

어떤 시뮬레이터이든지 그 정확성과 속도 그리고 메모리에는 반비례의 상관 관계가 존재하며 어떠한 방법으로 시간과 메모리 그리고 정확도를 적절하게 설정하느냐 하는 것은 좋은 시뮬레이터를 결정하는데 중요한 요소로 작용한다. 본 시뮬레이터는 우선 메모리를 600KB로 제한하여 효율적인 메모리 사용과 분할 깊이의 적절한 절충을 고려했으며, 사용 시스템의 성능과 사용자에 따라 분할 깊이를 적절히 선택함으로서 정확도와 실행시간을 최적의 상태로 조절할 수 있도록 하였다. 그리고 본 시뮬레이터의 분할모드 시뮬레이션 또 하나의 특징은 미지값이 출력된 경우에만 분할 시뮬레이션을 실행하므로 적은 미지값을 갖는 시뮬레이션에서 상대적으로 빠르게 동작하며 따라서 절약되는 시간 동안 분할 깊이를 깊게 함으로써 그 정확도를 더욱 향상시킬 수 있다는 점을 들 수 있다.

참고문헌

[1] Stephen A. Szygenda and Edward W. Thompson "Modeling and Digital Simulation for Design Verification and Diagnosis," IEEE Trans. Computers, vol.c-25, no.12, pp.1242-1253, Dec, 1976.

[2] Miron Abramovici, Melvin A. Breuer and Arthur D. Friedman "Digital System Testing & Testable Design," IEEE Press, pp. 43-46, 1990.

[3] Susheel J. Chandra and Janak H. Patel "Accurate Logic Simulation In The Presence Of Unknowns," IEEE Proc. of ICCAD, pp.34-37, 1989.

[4] H. P. Chang and J. A. Abraham, "The Complexity of Accurate Logic Simulation," Proc. International Conference on Computer-Aided Design, 1987.

[5] M. A. Breuer, "A Note on Three-Valued Logic Simulation," IEEE Transaction on Computers, pp.399-402, April 1972.

[6] F. Brglez and H. Fujiwara "A Neutral Netlist of 10 Combinational Benchmark Circuit and a Target Translation in FORTRAN," Proc. of ISCAS, pp.695-698.