# MDD를 이용한 효율적인 테스트 케이스 생성 방법론 연구*

안영정[0], 방기석, 최진영
고려대학교 컴퓨터학과
{yjahn, kbang, choi}@formal.korea.ac.kr

# A Method for the Effective Test-Case Generation using MDD

Young-Jung Ahn[0], Ki-Seok Bang, Jin-Young Choi
Dept. of Computer Science and Engineering, Korea University

요 약

복잡한 하드웨어 및 소프트웨어를 설계함에 있어 그 안정성에 대한 보장이 매우 중요하다. 이를 위해 정형 검증 이나 테스팅과 같은 많은 기법을 활용하고 있다. 그러나 안정성 검증을 위해 시스템을 모델링하고 테스트 케이스를 만드는 과정에서 상태 폭발에 따른 메모리의 한계에 부딪히게 된다. 본 논문에서는 이러한 문제를 해결하고, 메모리를 효율적으로 이용할 수 있는 탐색방법을 이용한 테스트 케이스 생성 알고리즘을 제안한다.

## 1. 서론

전자회로나 시스템 설계의 설계 과정에서 보면 모델링, 검증, 테스팅 기간이 전 설계 기간의 70%를 차지하고 선진국의 경우 검증 전문가의 수가 설계자 보다 많다는 통계 자료에서 알 수 있듯이 검증은 설계에 가장 큰 영향을 미치는 단계가 되었다. 이를 반영하듯, SOC(System On a Chip) 검증 분야에 동적 시뮬레이션과 정형 검증[1] 기술이 적절히 결합된 기법이 개발되어 설계에 점차로 적용되기 시작하고 있는데 이 새로운 기법을 assertion-driven formal verification이라 하여 중요한 신제품 분야로 부각되고 있다. Assertion-driven formal verification은 하드웨어 검증에서 정형 검증 기술 중 BDD(Binary Decision Diagram)[2]를 이용한 symbolic model checking을 이용하여 테스트 케이스를 생성하고 수식을 통해서 시스템을 검증 하여 시스템 개발 주기를 단축하는데 많은 기여를 하고 있다. 하지만 BDD를 이용한 테스트 케이스 생성 과정에서 시스템을 표현하는데 상태 폭발이 발생하여 시스템이 점차로 복잡해지는 현제 추세로 볼 때 모델 체킹[3]을 검증에 적용하는 방식이 맞지 않다고 지적되어 왔다. 또한 수식을 통한 검증 방식에서 수식에 위배될 때 반례를 하나 발생하게 되는데 이는 디버깅 측면에서 그 효율을 인정받지 못하고 있다.

본 논문에서는 이 두가지 문제점을 해결하기 위해 MDD(Multi Decision Diagram)[4]를 통한 테스트 케이스 생성 방법을 제안함으로써 상태 폭발을 다소 해결하고 모든 경우의 테스트 케이스를 생성함으로써 디버깅 측면에서의 효율을 높이고자 한다.

2장에서는 관련연구를 조사하고, 3장에서 Verilog-HDL[5]로 설계된 시스템을 MDD로 표현하는 규칙에 관하여 설명하며, 4장에서는 MDD로 변환된 시스템에서 모든 경우의 테스트 케이스를 생성하는 알고리즘에 대해 논한 다음 결론 및 향후 연구 방향을 제시함으로써 본 논문을 마무리한다.

* 본 연구는 한국과학재단 목적기초연구(R01-2000-00287)지원으로 수행되었음.

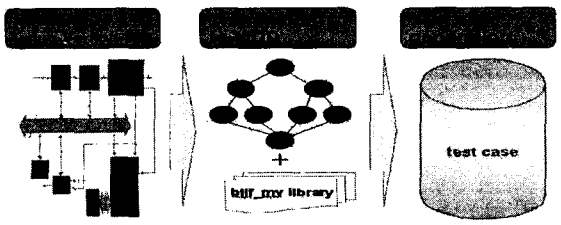
## 2. 관련 연구

이러한 BDD나 FSM[3]을 이용한 테스트 케이스 생성에 관한 연구는 다양한 측면으로 연구되고 있다. 예를 들면, 프랑스 Grenoble 연구소의 Lesar란 도구가 있다. Lesar는 BDD기반의 모델에서 PLTL을 적용하여 반례(count example)을 발생 수만큼 출력한다. 그리고 Vertex사의 Blacktie, Real Intent사의 Verix, Averant사의 solidify 등이, 지정된 시간 내에 정형 검증이 완료되지 않으면 자동으로 시뮬레이션용 테스트벤치를 생성하여 시뮬레이션 단계로 넘어가게 하는 " semi-formal verification" 의 범주에 있다. @HDL사의 @Verifier, Synopsys사의 Chechum 등이, simulation을 수행하면서 정형적 분석 방법을 위한 초기 상태(initial state)를 찾고 이를 기준으로 특정 clock cycle 동안 formal analysis를 수행 함으로서 효율화를 꾀한 " dynamic-formal verification" 으로 0-In사의 0-In check 등이 있다.

## 3. Vreilog-HDL에서 MDD로의 변환 규칙

본 논문에서 아래 [그림 1]과 같이 모델 명세 언어로는 Verilog-HDL을, MDD를 표현하는 중간 언어로는 BLIF-MV(Berkeley Logic Interchange Format - Multi-valued Variable)[6]를 제안한다.

Verilog-HDL는 " Gateway Design Automation" 에서 1983년에 개발되었다. 이 언어는 1990년에 Cadence Design




[그림 1] 테스트 생성 과정

System에 의해 공개됨으로써 표준이 되었고 전 세계적으로 논리회로 설계에서 널리 사용되는 언어이다. 특징을 살펴보면 동적인 행위 뿐만 아니라 정적인 하드웨어의 혼합-수준 명세도 허용한다. 동적인 행위의 표현을 용이하게 하기 위해 Verilog-HDL은 conditional, loop-control, process fork/join같은 고급 구조를 갖는다. Verilog-HDL은 또한 시간 표현을 용이하게 하는 특수 기능을 갖추고 있다. 이 기능는 스테이트먼트, 게이트, 모듈과 관련된 지연(delay)를 명세하기 쉽게 한다. BLIF-MV는 1996년 University of California, Berkeley에서 개발 되었다. BLIF-MV는 비결정적 계층적 순차 시스템을 설계하기 위한 언어이다. BLIF-MV은 입력된 언어로부터 cartesian product로 시스템의 행위를 표현하고, 이는 MDD로 명세되며 계층적 트리 형태로 저장된다. MDD는 BDD와 같이 상태의 나열을 피하고 도달 가능한 상태(reachable state)에 관해서만 표현하며 symbolic computation을 수행할 수 있는 특성이 있고 반면에 여러 값에 의해 레이블(label)화되고 내부 정점은 여러 값의 outgoing 간선(edge)를 가지는 variable $X_i$로 레이블화된다. 즉, BDD보다 상태의 수가 적다는 장점이 있어서 테스트 케이스의 생성시 시스템의 자원의 한계를 넘어서야 하는 제약점을 극복한다. BLIF-MV는 시스템의 행위를 아래 a)와 같이 표현하며 그 의미는 아래 b)와 같이 v3가 1이고 u78가 0일 때, v6가 1이고 u78가 1일 때, v3가 0이고 j가 1, 그리고 u78가 1일 때 v13.15가 1이 된다는 의미를 갖는다.

```
a) .name v3 v6 j u78 v13.15
1 - - 0 1
- 1 - 1 1
0 - 1 1 1

b) v13.15 = ( v3 u78' ) + ( v6 u78 ) + ( v3' j u78 )
```

다음은 Verilog-HDL(굵은체)에 따른 BLIF-MV(이탤릭체)의 변환 규칙을 나타낸다. 변수이름_ps, 변수이름_ns는 변수가 latch로 선언되었을 때 현재 상태와 다음 상태를 나타낸다.

**Net variables**

```
assign a = x;
assign a = y;
.names x y a
- - =x
- - =y
```

**Register variables**

```
always @(posedge clk) if (c1) a = e1;
always @(posedge clk) if (c2) a = e2;
.names c1 e1 c2 e2 a_ps a_ns
1 - - - - =e1
- 1 - - - =e2
0 0 - - - =a_ps
```

**Array Declaration**

```
assign b = a[1];
.names I<0> I<1> a[0] a[1] a[2] b
0 0 - - - =a[0]
0 1 - - - =a[1]
1 0 - - - =a[2]
1 1 - - - -
```

**Concatenation**

```
wire b;
wire [0:1] c, d;
wire [0:4] a
assign a = {2' b10, b, c & d};
```

```
.names t<0>
0
.names t<1>
0
.names b t<2>
- =b
.names c<0> d<0> t<3>
.def 0
1 1 1
.names t<0> a<0>
- =t<0>
.names t<1> a<1>
- =t<1>
.names t<2> a<2>
- =t<2>
.names t<3> a<3>
- =t<3>
.names t<4> a<4>
- =t<4>
```

**If statement**

```
if (c1) a = e1;
else a = e2;
.names c1 e1 e2 a_ps a_ns
1 - - - =e1
0 - - - =e2
```

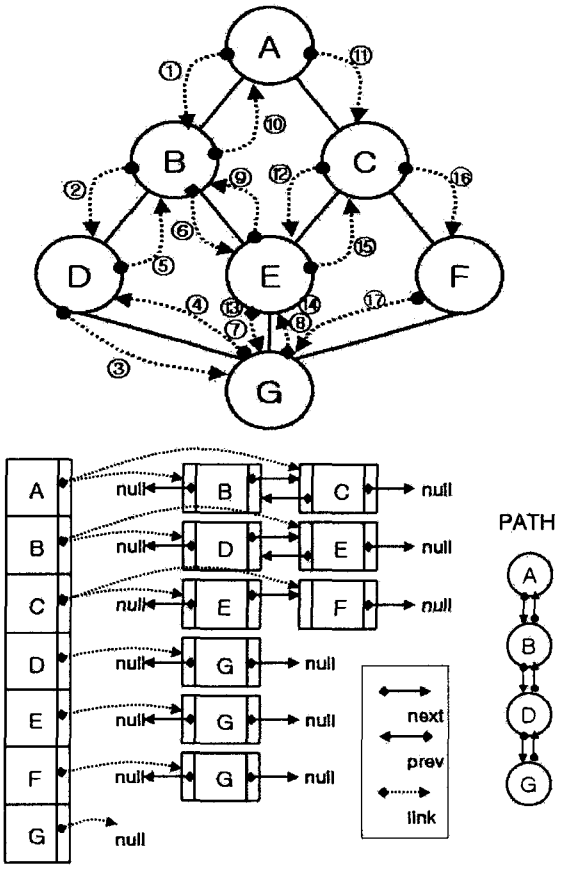
**Case statement**

```
case (STATE)
S0:
        a = e1;
S1:
        a = e2;
S3:
        a = e3;
endcase
.names STATE t<0>
.def 0
S0 1
.names STATE t<1>
.def 0
S1 1
.names STATE t<2>
.def 0
S2 1
.names t<0> t<1> t<2> e1 e2 e3 a_ps a_ns
1 - - - - - - =e1
0 1 - - - - - =e2
0 0 1 - - - - =e3
```

### 4. MDD로부터 테스트 케이스 생성 알고리즘

모델링한 시스템을 MDD로 전환하고 테스트 케이스를 생성하는 알고리즘은 깊이 우선 탐색(Depth-first search)[7]을 기본으로 한다. 깊이 우선 탐색과 테스트 케이스 생성 알고리즘의 차이점은 두가지로 볼 수 있는데, 깊이 우선 탐색은 그래프를 생성하면서 경로 탐색을 수행하고 본 논문에서 제안하는 테스트 케이스를 생성하는 알고리즘은 BLIF-MV로부터 그래프를 생성한 다음 경로 탐색을 수행한다. 두번째로 깊이 우선 탐색은 그래프의 정점에 대한 다음 정점을 큐에 저장할 때 inoutcoming 간선에 대한 다음 큐를 저장하지만 테스트 케이스를 생성하는 알고리즘은 outgoing에 대한 간선의 정점만을 정점의 다음 정점으로 큐에 저장하므로써 메모리 소비를 줄인다. 테스트 케이스 생성 알고리즘을 살펴보면 다음과 같다.

<그래프로부터 인접 리스트 생성>
단계-1. 그래프에 나타난 정점에 대해 정적으로 메모리 할당한다. 정점 메모리는 다음 정점을 가리키는 포인터 링크(link)를 갖는다.




[그림 2] MDD로부터 테스트 케이스 생성 알고리즘과 인접 리스트

단계-2. 정점에 따른 다음 정점을 큐 방식으로 그림과 같이 저장한다. 링크는 처음의 다음 정점을 가리키고 마지막 정점의 링크는 널(null)을 가리킨다.
<인접 리스트로부터 경로 탐색>
단계-3. 시작 정점의 다음 정점을 탐색한다. 패스 리스트에 스타트 정점을 삽입한다.
단계-4. 시작 정점의 링크가 가리키는 다음 정점의 정점 메모리로 간다. 패스 리스트에 다음 정점 삽입한다.
단계-5. 다음 정점이 널일때까지 단계-3과 단계-4를 반복한다.
단계-6. 다음 정점이 널을 가리키면 패스 리스트를 출력한다.
단계-7. 다음 정점이 널을 가리키면 패스 리스트에 그 이전 정점으로 가서 링크가 큐의 다음(next)를 가리키게 하고 이 정점의 다음 정적 메모리 링크들은 처음 정점을 가리키고 패스 리스트는 그 정점 이후의 정점들은 삭제하고 그 정점의 정보를 바꾼다.
단계-8. 만약 큐의 다음이 널이면 그 이전 정점의 이전 정점으로 가서 링크가 큐의 다음을 가리키게 한다. 패스 리스트
정보를 바꾼다. (큐의 다음이 널이 아닐때까지 역추적한다.)
단계-9. 단계-6을 수행한다.
단계-10. 시작 정점 메모리의 링크가 널을 가리킬때까지 단계-7, 단계-8, 단계-9를 수행한다.

## 5. 결론 및 향후 연구

시스템의 안정성 보장을 위해 정형검증 및 테스팅을 적용하는 연구가 활발히 진행되고 있다. 그러나 두 방법을 사용하는 데 있어서 모델링 및 테스트 케이스의 생성시 시스템 자원의 한계를 넘어서야 하는 큰 제약점이 있다. 이런 제약점 때문에 현재는 많은 제약사항을 가정한 상태로 검증 및 테스팅을 수행 하고 있다. 본 연구에서는 이러한 제약사항을 극복하고 보다 효율적인 시스템의 모델링 및 테스트 케이스 생성을 위해 MDD를 이용하는 방법론을 제안하고 있다. MDD를 이용한 테스트 케이스 생성 알고리즘은 다음과 같은 특징을 지닌다.
첫째, outgoing 간선에 관해서 이진값이 아닌 다중값을 가지는 MDD로 모델을 표현함으로써 상태폭발을 줄일 수 있다. 즉, 모델의 상태를 나타내는 정점의 레이블 수를 줄임으로써 메모리면에서의 효율을 높일 뿐 아니라 수행 속도면에서의 효율을 높일 수 있다. 본 연구에서 사용하는 MDD는 pure parallelism[3]을 따름으로써 모델을 명세시 좀 더 명확하게 병렬 시스템을 표현한다.
둘째, MDD를 표현하는 중간 언어로 BLIF-MV를 선택함으로써 이 언어를 받아들이는 CTL Model Checker VIS(Verification Interaction with Synthesis)[4]를 이용하여 요구 명세와 설계 명세에 대한 검증에도 적용할 수 있다.
셋째, 모델의 모든 경로를 자동으로 생성함으로써 테스트의 용이성과 정확성을 높여 준다. 즉, 보다 안정되고 신뢰성있는 내장형 시스템을 제작할 수 있다. 또한, 테스트 비용 절감에 기여한다
향후 연구 과제로는 본 논문에서 제시한 알고리즘을 구현하여 다른 툴과의 실험적 결과를 비교하고 입력 언어를 하드웨어/소프트웨어 통합 설계 언어로 확대하여 시스템 설계에 대한 적용분야를 넓히고자 한다.

## 참고 문헌

[1] E. M. Clarke and J. M. Wing, " Formal Methods: State of Art and Future Directions" , ACM Computing Surveys, 28(4), pp.626-643, December 1996
[2] R. E. Bryant. Graph-based Algorithms for Boolean Funtion Manipulation. IEEE Transations on Computers, Vol. 35, No. 6, pp. 677-691, August 1986
[3] Kenneth L. McMillan, Symbolic Model Cheking, Kluwer Acadecmic Publisher, 1993
[4] Tiziano Villa,Gitanjali Swamy,Thomas Shiple, " VIS User' s Manual" , University of California, Berkeley,1996
[5] Bob ZeidMan, Verilog Designer' s Library, prentice-Hall,Inc.,Upper Saddle River, New jersey, 1999
[6] Yuji Kukimoto, " BLIF-MV" ,The VIS Group,University of Califonia, Berkely, May 31, 1996
[7] Horowitz, Sahni, Anderson - Freed, Fundamental Data Structures in C, computer science press, 1993