Verilog HDL 로 기술된 조합 논리회로의 Cadence SMV 기반 정형 검증 방법

조성득*, 김영규*, 문병인*, 최윤자**
*경북대학교 전자공학부
**경북대학교 IT 대학 컴퓨터학부
e-mail : yuchoi76@knu.ac.kr

A Cadence SMV Based Formal Verification Method for Combinational Logics Written in Verilog HDL

Seong-Deuk Jo*, Young-Kyu Kim*, Byungin Moon*, Yunja Choi**
*School of Electronics Engineering, Kyungpook National University
**School of Computer Science and Engineering, Kyungpook National University

요 약

하드웨어 디자인 설계에서 초기 단계의 설계 오류 발견은 개발 비용 감소 및 설계 시간 단축 측면에서 그 효과가 매우 크다. 이러한 초기 설계 오류 발견을 위한 대표적인 방법으로는 정형 검증(formal verification)이 있으며, Cadence SMV(Symbolic Model Verifier)는 정형 검증을 위해 Verilog HDL(Hardware Description Language) 을 SMV 로 자동 변환 해주는 장점이 있지만, 사건 기반 구조(event based structures)의 sensitivity list 에 대한 지원을 하지 않는 한계가 있다. 이에 본 논문에서는 Cadence SMV 에서 디지털회로(digital circuit) 중 하나인 조합 논리회로(combinational logic circuit)를 sensitivity list 가 고려된 검증이 가능하도록 하는 방법을 제안한다. 신뢰성 있는 실험을 위해 본 논문에서는 제안하는 방법의 일반적인 규칙을 도출하였고, 도출된 규칙이 적용된 SMV 파일을 생성하는 자동화 프로그램을 구현하여 실험하였다. 실험결과 제안한 방법을 적용한 경우 기존 Cadence SMV 가 발견하지 못한 설계상의 오류를 발견할 수 있었다.

1. 서론

최근 VLSI(Very Large Scale Integration) 시스템의 복잡도가 증가함에 따라 설계 오류 가능성은 한층 커지고 있으며, 이러한 오류들은 설계 결과물에 치명적인 결함을 초래하여 개발 시간 및 비용 측면에서 큰 손실을 야기한다. 따라서 VLSI 시스템 설계에서 초기 설계 단계 오류를 검증하는 것은 설계 사이클 단축 및 개발 비용 절감 효과 측면에서 매우 중요하게 인식되고 있으며, 논리식이나 수학식 등 정형화된 식을 이용하여 시스템의 동작을 검증하는 정형 검증(formal verification) 기술들을 VLSI 시스템 설계 및 하드웨어 디자인 산업 분야에 적용하려는 연구들이 활발히 진행되고 있다[1, 2].

Cadence SMV(Symbolic Model Verifier)는 정형 검증 기법 중 모델 체킹 기법을 사용하는 검증 도구로서, Verilog HDL(Hardware Description Language) 파일을 SMV 파일로 자동 변환하여 정형 검증을 진행한다[3, 4]. 그러나 SMV 파일로 자동 변환하는 과정에서 사건 기반 타이밍 제어(event based timing control)에 대해서는 지원을 하지 않기 때문에 sensitivity list 에 따라 타이밍 시뮬레이션(timing simulation) 결과와 Cadence SMV 를 이용한 검증 결과 간의 차이가 발생하여 설계 오류를 검증하지 못하는 문제가 발생한다[4]. 이에 본 논문에서는 Cadence SMV 를 이용해 디지털회로(digital circuit) 중 하나인 조합 논리회로(combinational logic circuit)의 sensitivity list 를 고려한 검증이 가능하도록 하는 방법을 제안한다. 제안하는 방법은 SMV 파일을 자동으로 생성하는 프로그램으로 구현하여 Cadence SMV 에서 실험 하였으며, 본 논문에서 제안하는 방법의 검증 결과와 타이밍 시뮬레이션 결과를 비교하여 검증 결과를 확인하였다.

2. 연구배경

2.1. Verilog HDL

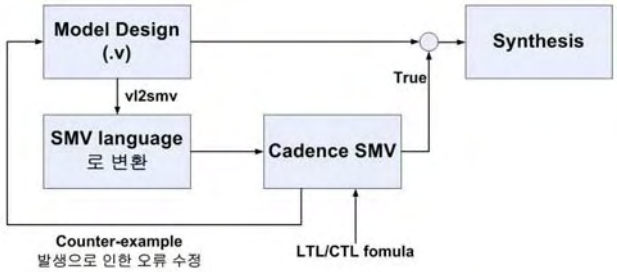
Verilog HDL 은 Gateway Design Automation 사에 의해 개발된 디지털 회로의 동작을 기술하기 위한 언어이다. 집적 기술의 발달로 인해 VLSI 시스템을 손이나 컴퓨터 지원 설계 도구(computer-aided design)로 직접 그린다는 것은 불가능해졌다. 그래서 현재 대부분의 디지털 회로 설계 기술자들은 디지털 회로의 동작을

** 교신저자

Verilog HDL 로 기술하고, 논리 합성 도구(logic synthesis tool)를 이용하여 디지털 회로를 설계한다. Verilog HDL 은 C 언어와 문법(syntax)이 비슷하여 쉽게 배우고, 사용할 수 있다[5].

2.2. Cadence SMV

Cadence SMV 는 Berkeley 대학의 K. L. McMillan 에 의해 개발된 정형 검증 도구이다[4]. SMV 는 ROBDD(Reduced Ordered Binary Decision Diagram)를 기반으로 하는 심볼릭 모델 체킹 알고리즘을 사용하여, 유한 상태 시스템(finite state machine)이 LTL(Linear Time temporal Logic) / CTL(Computational Tree Logic)로 표현된 요구 명세를 만족하는 지를 검증한다[3]. Cadence SMV 는 GUI (Graphical User Interface) 환경을 제공하여 사용자가 편리하게 사용할 수 있으며, 모델 체킹 기법으로 기술이 간단하고 명확하며, 검증이 용이하다는 장점이 있다. 그리고 Cadence SMV는 시스템 모델링 언어로 유한 상태 시스템을 쉽게 기술할 수 있는 SMV 언어와 하드웨어 기술 언어인 Verilog HDL 을 지원한다. 시스템 모델링 언어로 Verilog HDL 을 사용하여 검증할 경우, 그림 1 처럼 Cadence SMV 에서 제공하는 vl2smv 유틸리티에 의해 SMV 로 자동으로 변환되어서 사용된다[1].



(그림 1) Cadence SMV 의 검증 절차

3. 조합 논리회로 검증

3.1. 조합 논리회로

조합 논리회로는 래치(latch)나 레지스터(register)같은 내부 저장회로가 없고 회로의 동작이 현재의 입력 신호에 의해서만 결정되는 디지털 회로이다[6]. Verilog HDL 을 이용해 조합 논리회로를 기술할 때는 연속적인 할당(continuous assignment)방법이나 always 문을 이용할 수 있다. 그리고 always 구문을 이용해 조합 논리회로를 기술할 때 posedge, negedge 와 같은 edge 사건(edge event)은 sensitivity list 에 없어야 한다[7].

3.2. vl2smv 의 문제점

그림 2 는 조합 논리회로인 2-to-1 MUX 에 대한 Verilog HDL 코드(code)와 Cadence SMV 에 의해 자동 변환된 SMV 코드이다. 그리고 그림 3 은 2-to-1 MUX 에서 sensitivity list 중 입력신호 b 를 제외한 Verilog HDL 코드와 Cadence SMV 에 의해 자동 변환된 SMV 코드이다. 두 Verilog HDL 코드는 sensitivity list 의 차이 때문에 동작이 서로 상이하다. 그림 4 는 각각의 코드에 대한 타이밍 시뮬레이션 결과이다. 입력신호 s 가 0 일 때, 출력신호 c 의 값은 입력신호 b 의 sensitivity list 포함 여부에 따라 달라진다. 하지만 vl2smv 유틸리티에 의해 자동 변환된 두 SMV 코드는 그림 2, 그림 3 과 같이 sensitivity list 와 상관없이 동일하게 변환된다. 그래서 Cadence SMV 를 이용해 그림 3 의 코드를 가지고 입력신호 s 가 0 일때 출력신호 c 와 입력신호 b 가 같은지를 LTL 로 기술한 식(1)의 검증결과는 True 이다.

$$G(s = 0 \rightarrow c = b) \quad (1)$$

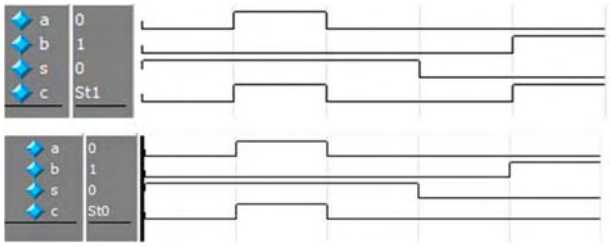
이는 sensitivity list 를 고려한 결과가 아니므로, 본 논문에서는 sensitivity list 를 고려한 검증이 가능한 SMV 코드로 보정하는 방법을 제안한다.

```
always @(a or s or b)          do {
begin                              if((s =1))
    if(s == 1)                         c  := a ;
        c = a;                     else
    else                               c  := b ;
        c = b;                 }
end
```

(그림 2) 2-to-1 MUX 의 Verilog HDL 코드(좌), SMV 코드(우)

```
always @(a or s)               do {
begin                              if((s =1))
    if(s == 1)                         c  := a ;
        c = a;                     else
    else                               c  := b ;
        c = b;                 }
end
```

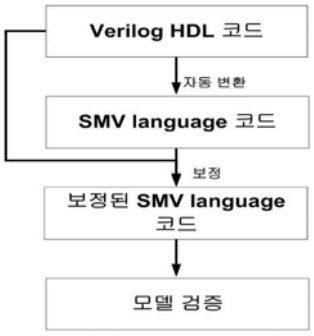
(그림 3) 그림 2 코드에서 sensitivity list 중 입력신호 b 가 빠진 코드



(그림 4) 그림 2 코드 타이밍 시뮬레이션(상), 그림 3 코드 타이밍 시뮬레이션(하)

3.3. 개선된 변환 규칙

그림 5 는 제안하는 변환 보정 방법에 대한 절차이다. 검증할 Verilog HDL 코드를 Cadence SMV 를 이용해 SMV 코드로 변환하고, 다시 SMV 코드를 Verilog HDL 코드의 sensitivity list 를 이용하여 본 논문에서 제안하는 변환 방법으로 보정한다. 본 논문에서 제안한 변환 보정 방법은 래치를 고려한 SMV 변환처럼[8] 변환된 SMV 파일에 신호의 이전 상태(state)의 값을 저장하는 새로운 신호를 추가하여 변환하였고, 규칙은 다음과 같다.



(그림 5) 제안하는 변환 절차

- 변환 규칙

1. Do 문 이전에 sensitivity list 에 있는 신호와 출력 신호의 이전 상태의 값을 저장하는 신호 생성
2. 첫 번째 상태에서 sensitivity list 신호의 이전 상태의 값을 현재 상태의 값의 반전으로 (High→Low, Low→High) 한번만 초기화
3. Do 문 안에서 출력신호의 이전 상태의 값을 현재 상태의 값으로 할당
4. Sensitivity list 를 체크하는 if 문을 추가하고, sensitivity 조건을 만족하면 기존의 동작 코드를 수행
5. Do 문을 벗어나 규칙 1 에서 생성한 신호의 다음 상태의 값으로 현재 상태의 sensitivity list 에 있는 신호와 출력신호의 값을 할당

그림 6은 변환 규칙을 적용한 SMV 코드이다. 괄호 안의 번호는 적용된 규칙의 번호이다. 규칙 1 은 sensitivity list 를 검사하는 것과 latch register 가 발생하는 것을 대비해 이전 상태의 입력신호와 출력신호의 값을 저장하기 위해 추가하였다. 규칙 2 는 Cadence SMV 를 이용해 정형 검증을 할 때, 첫 번째 상태에서부터 sensitivity list 를 만족하지 않는 경우를 제외하기 위해 추가하였다. 규칙 3 은 의도치 않은 latch register 가 발생하는 상황을 고려하기 위해 추가하였다[8]. 규칙 4 는 본 논문에서 제안한 sensitivity list 를 검사하기 위해 추가하였다. 마지막으로 규칙 5 는 현재 상태의 값이 다음 상태에서 이전 상태의 값으로 사용되기 때문에 다음 상태로 넘겨주기 위해 추가하였다.

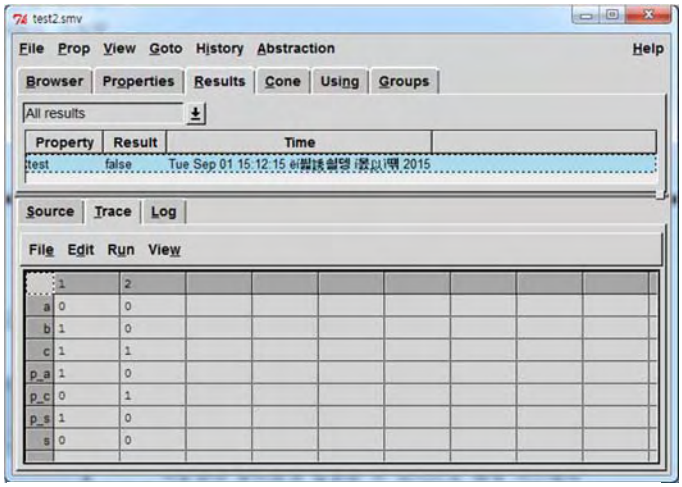
```
p_s : boolean resolve;   [1]
p_a : boolean resolve;   [1]
p_c : boolean resolve;   [1]
init(p_a) := ~a;         [2]
init(p_s) := ~s;         [2]
do {
c := p_c;                [3]
if((p_a~=a)||(p_s~=s))   [4]
{
    if((s =1))
        c  := a ;
    else
        c  := b ;
} }
next(p_s) := s;          [5]
next(p_a) := a;          [5]
next(p_c) := c;          [5]
```

(그림 6) 규칙을 적용한 SMV 코드

4. 실험

본 논문의 실험은 Intel i7-4790 @ 3.60 GHz, 메모리 8 GB, Windows 7 OS 환경에서 Cadence SMV 를 이용하여 실험하였다. 신뢰성 있는 실험을 위해 Python 3.4 를[9] 이용하여 본 논문에서 제안하는 방법이 적용된 SMV 코드를 자동으로 생성하는 프로그램을 구현하였으며, 구현한 프로그램을 통해 sensitivity list 가 빠진 그림 3 의 코드에서 생성된 SMV 코드를 이용하여, 식 (1)을 검증하였다.

그림 7 은 그림 3 의 코드를 본 논문에서 제안하는 방법을 적용하여 식 (1)을 검증한 결과이다. 실험 결과, 기존의 검증 결과와 달리 본 논 논문에서 제안하는 방법이 적용된 SMV 코드에서는 식 (1)이 유효하지 않으며, 다음과 같은 반례(counterexample)가 나타났다. 그림 7 의 두 번째 상태를 보면 sensitivity list 의 입력신호 a, s 둘 다 이전 상태의 입력신호 p_a, p_s 와 비교해 값의 변화가 없으므로 입력신호 b 의 값이 바뀌어도 출력에는 반영되지 않는다. 따라서 본 논문에서 제안하는 SMV 변환 방법을 통해 조합 논리회로에서 sensitivity list 가 고려된 정확한 검증이 가능한 것을 확인하였다.



(그림 7) 식 (1)에 대한 반례

5. 적용

본 논문에서 제안하는 방법은 (주)정우이엔지의 모듈 검증에 적용되었다. 그림 8 은 (주)정우이엔지의 모듈중 Mpublk 모듈의 Verilog HDL 코드에서 본 논문에서 제안하는 검증 방법을 적용한 always 문이다. 그림 8 의 조합 논리회로의 동작은 입력신호 Cp_Cold_nRst 가 0 일 때 또는 입력신호 nIntrst 가 1 일 때는 4 bits 출력신호 oMpu_A_Lwe, oMpu_B_Lwe, oMpu_C_Lwe, oMpu_D_Lwe 모두 High Impedance 가 된다. 그리고 입력신호 Cp_Cold_nRst 가 1, nIntrst 가 0 일 때는 2 bits 입력신호 Mpu_Mst_Sel 의 값이 '00', '01', '10', '11'일 때에 따라 각각 4 bits 출력신호 oMpu_A_Lwe, oMpu_B_Lwe, oMpu_C_Lwe, oMpu_D_Lwe 가 '0111'이 된다. 표 1 은 그림 8 의 조합 논리회로에서 기존의 Cadence SMV 는 sensitivity list 에 입력 신호

Mpu_Mst_Sel 이 빠진 설계 오류를 발견하지 못하지만 본 논문에서 제안하는 방법을 적용했을 때 설계 오류를 찾을 수 있는 CTL 검증 속성과 검증 결과이다. 검증 1 은 4bits 출력신호 oMpu_A_Lwe 가 '0111'이 되는 조건에서 oMpu_A_Lwe 가 항상 '0111'으로 동작하는지를 검증하는 식이다. 검증 2 는 4bits 출력신호 oMpu_B_Lwe 가 '0111'이 되는 조건에서 oMpu_B_Lwe 가 항상 '0111'으로 동작하는지를, 검증 3 은 4bits 출력신호 oMpu_C_Lwe 가 '0111'이 되는 조건에서 oMpu_C_Lwe 가 항상 '0111'으로 동작하는지를, 검증 4 는 4bits 출력신호 oMpu_D_Lwe 가 '0111'이 되는 조건에서 oMpu_D_Lwe 가 항상 '0111'으로 동작하는지를 검증하는 식이다.

그림 8 의 코드에서 입력신호 Mpu_Mst_Sel 은 sensitivity list 에 포함되지 않았다. 그래서 입력신호 Cp_Cold_nRst 가 1, nIntrst 가 0 인 상태가 계속되면 입력신호 Mpu_Mst_Sel 이 바뀌어도 출력신호는 바뀌지 않아서 표 1 의 모든 검증 속성은 False 가 됨을 검증하였다.

```
always@(Cp_Cold_nRst or nIntrst)
begin
    if(!Cp_Cold_nRst)
    begin
        oMpu_A_Lwe = 4'hZ;
        oMpu_B_Lwe = 4'hZ;
        oMpu_C_Lwe = 4'hZ;
        oMpu_D_Lwe = 4'hZ;
    end
    else if(!nIntrst)
    begin
        if(Mpu_Mst_Sel == 0)
            oMpu_A_Lwe <= 7;

        if(Mpu_Mst_Sel == 1)
            oMpu_B_Lwe <= 7;

        if(Mpu_Mst_Sel == 2)
            oMpu_C_Lwe <= 7;

        if(Mpu_Mst_Sel == 3)
            oMpu_D_Lwe <= 7;
    end
    else
    begin
        oMpu_A_Lwe <= 4'hz;
        oMpu_B_Lwe <= 4'hz;
        oMpu_C_Lwe <= 4'hz;
        oMpu_D_Lwe <= 4'hz;
    end
end
```

(그림 8) Mpublk 모듈의 코드 일부

<표 1> Mpublk 모듈의 검증 속성 및 결과

	검증 속성	결과
검증 1	AG((Cp_Cold_nRst=1)&(nIntrst=0)& (Mpu_Mst_Sel=0) → (oMpu_A_Lwe=7));	False
검증 2	AG((Cp_Cold_nRst=1)&(nIntrst=0)& (Mpu_Mst_Sel=1) → (oMpu_B_Lwe=7));	False
검증 3	AG((Cp_Cold_nRst=1)&(nIntrst=0)& (Mpu_Mst_Sel=2) → (oMpu_C_Lwe=7));	False
검증 4	AG((Cp_Cold_nRst=1)&(nIntrst=0)& (Mpu_Mst_Sel=3) → (oMpu_D_Lwe=7));	False

6. 결론

본 논문에서는 Cadence SMV 를 이용해 디지털회로 중 하나인 조합 논리회로에서 sensitivity list 를 고려한 검증이 가능하도록 하는 방법을 제안하였다. 실험 결과, 기존의 Cadence SMV 에서는 찾아내지 못하는 설계상의 오류를 발견할 수 있었으며, 또한 제안하는 방법이 적용된 자동 변환 프로그램 개발을 통해 추가적인 변환 작업을 최소화 하였다. 이러한 연구는, 시스템 설계 단계에서 정형 검증 기법을 이용한 초기 오류 발견 가능성을 더욱 높일 수 있으며, 이로 인한 설계 사이클 단축 및 개발 비용 절감 효과는 매우 클 것으로 기대된다. 앞으로 조합 논리회로뿐만 아니라 순차 논리회로(sequential logic circuit)에서 sensitivity list 를 고려한 검증이 가능하도록 변환 방법을 확장하는 연구를 계속 진행 할 계획이다.

< Acknowledgement >

이 논문은 2014 년도 국방기술품질원의 재원으로 (주)정우이엔지의 위탁과제로 수행된 연구의 결과임

참고문헌

[1] 이승호, 이현룡, 장종권, "SMV 를 이용한 Pipeline 시스템의 설계 검증," *대한전자공학회 2003 년 하계종합학술대회*, 2003, pp. 939-942.

[2] H. Jain, D. Kroening, N. Sharygina, and E. Clarke, "Word level predicate abstraction and refinement for verifying RTL verilog," in *Proc. of the 42nd annual Design Automation Conference*, 2005, pp. 445-450.

[3] M. Huth and M. Ryan, *Logic in Computer Science: Modelling and reasoning about systems*: Cambridge University Press, 2004.

[4] K. L. McMillan. *Cadence SMV*. Available: http://www-cad.eecs.berkeley.edu/~kenmcmil

[5] S. Palnitkar, *Verilog HDL: a guide to digital design and synthesis*: Prentice Hall, 2003.

[6] R. H. Katz and G. Borriello, *Contemporary logic design*: Prentice Hall, 2005.

[7] "IEEE Standard for Verilog® Register Transfer Level Synthesis," *IEEE Std 1364.1™-2002*, 2002.

[8] T. Kratochvíla, V. Řehák, and P. Šimeček, "Verification of COMBO6 VHDL Design," Technical report, 2003.

[9] *Python3.4*. Available: https://docs.python.org/3.4/