모델 체킹을 통한 조합 회로의 정형 검증

안영정°, 송관호, 최진영
고려대학교 컴퓨터학과
{yjahn°, ghsong, choi}@formal.korea.ac.kr

A Study for Formal Verification of Combinational Circuit Using Model Checking

Young-Jung Ahn°, Gwan-Ho Song, Jin-Young Choi
Dept. of Computer Science & Engineering, Korea University

요약

하드웨어 개발에 있어서 데이터의 신속한 처리와 공정의 저렴한 비용을 위해 많은 부분이 조합회로로 설계된다. 기능 검사는 하드웨어 개발에 있어서 설계의 기능을 분석하는 중요한 설계 흐름이다. 하지만 복잡한 기존의 기능 검사의 절차는 사용자의 요구에 의해 하드웨어 시스템이 복잡해지고 정보산업의 발전에 따라 개발 주기가 점점 빨라지는 시장의 특성으로 인해 설계자에게 많은 시간적 경제적인 부담감을 준다. 본 논문에서는 설계자에게 가중되는 부담을 극복하고 효율적인 조합회로의 기능 검사를 위한 정형적 방법을 제시하고자 한다.

1. 서론

정보 산업의 발전함에 따라 컴퓨터 하드웨어 및 소프트웨어 시스템의 개발 주기가 매우 짧아지고 있다. 그리고 사용자의 요구가 다양해짐에 따라 기존에 개발된 시스템에 많은 부가적인 기능들이 추가되어 시스템이 점차 복잡해지는 추세이다. 이러한 발전에 따라 시스템의 정확성 확인을 위한 노력이 꾸준히 이루어지고 있다.

기존의 전통적인 하드웨어 시스템의 설계 과정에서 보면 정확성 확인을 위해 기능 검사를 반드시 거치게 된다. 일반적으로 기능 검사는 하드웨어 명세 언어를 사용하여 시스템을 설계하고 시뮬레이터를 이용하여 테스트 벡터에 대한 출력 신호, 즉 파형을 검사하는 방법이다. 이 경우 기능 검사를 할 설계자는 설계된 시스템에 대해 정확하게 파악하고 있어야 하고 기능 검사에 사용될 입력을 준비해야 하며 생성될 출력 파형에 대해 예지하고 출력된 파형을 분석해야 한다. 또한 기능 검사할 시스템이 복잡해질수록 출력 신호에 대한 분석이 매우 어려워진다. 만일 기능 검사를 통해 오류가 발생하면 그 오류가 설계의 어느 부분에서 발생하였는지 판단하기 위해 설계 명세 언어의 소스를 모두 대응 시켜 오류를 찾아가는 불편함이 있다.

결국, 기능 검사를 수행하기 위해 시간적 경제적인 부담이 가중된다. 이러한 이유로 하드웨어를 검증 할 수 있는 보다 효율적이며 신뢰할 수 있는 방법의 필요성이 제기되었다. 모델 체킹 (Model Checking) [1]은 도구를 이용하여 손쉽고 빠르게 검증할 수 있으며 자동화되는 장점이 있지만, 적용분야는 유한 시스템으로 표현되는 순차회로로 제한된다.

본 논문에서는 모델 체킹을 이용하여 조합회로의 기능 검사를 수행하기 위해서, 어떠한 전자 회로가 언제나 주어진 속성을 만족하는지에 사용되는 동치성 검사 (Equivalence Checking)[2]를 모델 체킹에 적용함으로써 모델 체킹의 적용분야를 확장하고자 한다.

본 논문은 연구의 동기와 정형적인 하드웨어의 검증에 관련되어 진행되고 있는 연구에 대해 논하고, 모델 체킹에 동치성 검사를 적용하여 조합회로의 기능 검사를 수행하는 연구 방법과 결과 및 적용 사례를 보여준다. 마지막으로 본 연구의 향후 연구에 대해 논하면서 논문을 마무리 한다.

2. 관련 연구

하드웨어 시스템의 기능 검사에 대한 정형적인 연구는 동치성 검사, 모델 체킹, Bounded Model Checking (BMC)[3], 등 다양한 측면에서 연구되고 있다.

먼저 동치성 검사는 Satisfiability Problem (SAT)를 이용하여 두 회로가 동치임을 증명한다. 두 회로가 같은 신호의 입력에 대해서 동일한 신호를 출력한다는 것을 증명하기 위해 miter를 구성하고 SAT를 통해 두 회로가 동치인지 아닌지를 증명한다. miter는 두 회로의 대칭되는 입력들을 연결하고 대칭되는 출력들을 XOR, OR 게이트를 이용하여 회로들이 동치이면 '0', 동치가 아니면 '1'을 출력하도록 구성한다. SAT는 어떠한 표현식-회로를 표현하는 Conjunctive Normal Form

(CNF)-이 있을 때, 그 표현식을 '1'이 되게 하는 값의 조합이 존재하면 그 표현식은 "satisfiable (SAT)"이라고 얘기하고, 없으면 "unsatisfiable (UNSAT)"이라고 얘기한다. 하드웨어 시스템의 개발 단계는 Abstract - RTL - Structural - Transistor 4단계로 이루어지고, Abstract 단계에서 기존의 기능 검사를 통해 회로의 정확성을 확인한다. 동치성 검사는 RTL - Structural - Transistor 단계에서 이전 단계의 모델과 동치를 확인하여 각 단계의 기능 검사를 수행한다.[4] 동치성 검사는 Abstract 단계에서 올바른 모델을 얻기 전까지 기능 검사를 수행할 수 없는 단점이 있다. 동치성 검사를 하는 도구로는 BerkMin[5], MiniSat[6], GRASP[7]등이 있다.

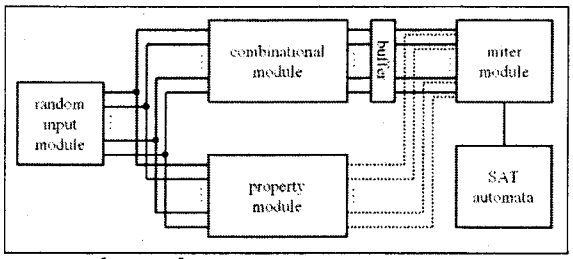
모델 체킹은 상태 전이 시스템과 특성이 주어지면, 주어진 시스템이 검증하고자 하는 특성을 만족하는지 알아보기 위해 전체 상태 공간을 검사하고 그 결과를 알려준다. 조합회로는 기억 소자가 없어서 상태 전이 시스템으로 표현하지 못하므로 모델 체킹할 수 없다. 모델 체킹을 수행하는 도구로는 VIS[8], SMV[9]등이 있다.

BMC는 CNF로 회로를 표현하고 SAT를 이용하여 기능 검사를 수행한다. 한 입력에 대한 신호를 출력하는 것을 step이라 하고 사용자가 지정한 step까지 SAT를 통해 입력에 대한 올바른 신호를 출력하는지 확인한다. CNF의 표현은 positive/negative literal (variable)들의 disjunction으로 이루어진 claus들의 conjunction으로 구성되다. 회로의 명세에 있는 대입문들을 disjunction과 conjunction으로 이루어진 CNF으로 표현하기 때문에 많은 literal과 claus가 생성된다. BMC로 조합 회로의 기능 검사 수행에 있어서 조합 회로는 초기값이 없기 때문에 매 step마다 전체 회로의 CNF를 생성하고 입력에 대한 출력을 확인해서 비교적 긴 수행시간을 가지고 있다. 하드웨어의 BMC 도구는 hw-CBMC[10]가 있다. 실험을 통해 기존의 기능 검사와 BMC 그리고 본 논문 제안한 방법의 탐색 공간과 수행시간을 비교하고자 한다.

3. 모델 체킹을 이용한 조합회로 연구 방법

조합회로는 기억 소자를 제외한 게이트들을 조합하여 만든 논리회로로서 입력에 따라 출력이 결정되며, 이 때 그 출력은 그 전 입력의 영향을 받지 않는다.

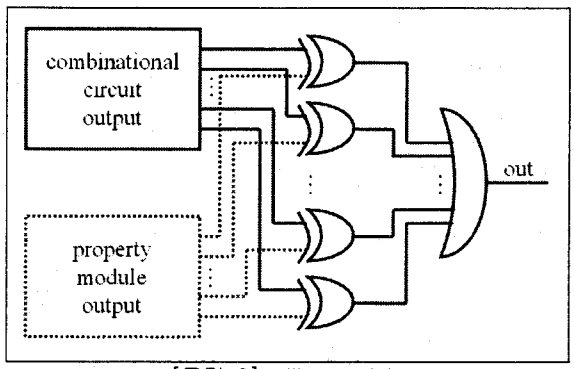
본 논문에서는 [그림 1]과 같이 정확한 모델을 기억소자를 가진 모델로 구현하고 개발하고자 하는 조합회로와 miter 구조를 구성한다. 그리고 모든 입력에 대해 두 회로가 동치임을 모델 체킹을 통해서 확인한다. 모델 체킹 도구는 하드웨어 명세 언어인 verilog로 구현된 모델을 상태 전이 시스템으로 받아들이고 Computational Tree Logic (CTL)을 검증하고자 하는 속성으로 받아들이는 VIS를 사용한다.



[그림 1] 조합회로의 모델 체킹 구조

random input module은 회로의 입력에 대해서 조합 가능한 모든 값을 생성하고 이 값들을 동시에 두 회로의 입력으로 전달한다. VIS에서 "assign input = $ND(0,1);"와 같이 전체 상태 공간을 검사하기 위해서 입력마다 난수를 대입한다. 여기서 "input"은 1비트 wire인 데이터 타입을 가진다. 기능 검사할 조합회로는 combinational module에 구현된다.

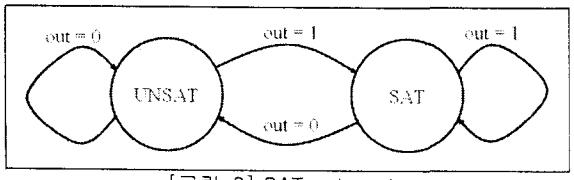
property module은 검증하고자 하는 속성을 기억소자를 이용하여 표현한다. 회로의 출력값들을 오토마타의 상태들로 표현하고 조합회로의 입력들은 오토마타의 전이로 구현한다. 조합회로는 이전 입력의 영향을 받지 않기 때문에 오토마타의 어느 상태에서든 입력에 따라서 가져야할 출력값을 나타내는 상태로 전이되도록 설계한다. verilog의 case문을 이용하면 간단하게 위에서 설명한 오토마타를 구현할 수 있다. buffer module은 combinational module과 property module의 sync를 맞추기 위한 모듈이다.



[그림 2] millet module

miter module은 combinational module과 property module이 조합 가능한 같은 입력에 대해 출력이 동치이면 '0', 동치가 아니면 '1'을 출력하도록 구성한다. [그림 2]와 같이 combinational module과

property module의 대칭되는 출력들을 XOR 연산한 후, 이 결과를 OR 연산한다. XOR 연산에 의해서 두 회로가 같은 입력에 대해서 서로 다른 출력값을 가지면 '1'로 나타내고 동일한 값에 대해서는 '0'을 나타낸다. OR 연산은 두 회로의 출력들이 하나라도 틀린 값을 가지면 '1'로 나타낸다.



[그림 3] SAT automata

SAT automata는 [그림 3]과 같이 miter module의 결과 (out)에 따라서 상태가 "SAT"과 "UNSAT"으로 나타내도록 구현한다. 두 회로가 동치면 "UNSAT", 동치가 아니면 "SAT"를 나타낸다. "AG(state = UNSAT)"란 CTL 속성을 통해서 두 회로가 같은 입력에 대해서 항상 같은 신호를 출력하는지 모델 체킹을 통해 검증한다.

4. 적용 사례

본 논문에서는 차세대 Advanced Encryption Standard (AES)인 Rijndael 알고리즘[11]의 Sbox를 순차회로로 구현하고 제안된 방법에 의해 기능 검사를 수행했다. 대칭형 암호 알고리즘의 표준으로 사용되어 왔던 DES 알고리즘이 오늘날의 컴퓨터 성능의 발달로 인해 암호가 해독되는 등 보안에 대한 취약성이 발견되었다. 그래서 미국 국립표준기술연구소 (NIST)에서는 DES 알고리즘을 대체할 AES를 세계적으로 공모하여 Rijndael 알고리즘을 표준으로 채택했다.

Sbox는 데이터의 무결성을 보장하는 부분이다. Sbox는 $GF(2^8)$에서 곱셈의 역원을 구하고 affine 변환을 수행한다. $GF(2^8)$에서 곱셈의 역원을 구하는 알고리즘은 입력에 따라 알고리즘의 반복 횟수가 다르다. 따라서 NIST에서는 Sbox를 알고리즘에 의해 구현하지 말고 ROM으로 구현된 sbox를 이용하도록 권장하고 있다. ROM은 데이터를 읽을 때마다 1 clock씩 소모한다. Rijndael 알고리즘은 10 round를 통해 128비트의 데이터를 암호화하고 128비트를 하나의 Sbox를 통해 변환하면 16 clock이 소모된다. 물론 16개의 Sbox를 병렬적으로 사용하면 1 clock이 소모된다. 본 논문에서는 신속한 데이터 처리를 위해 시간의 소모가 없는 조합회로로 Sbox를 구현한 후, 기존의 기능 검사와 BMC 그리고 논문에서 제안한 방식의 결과를 비교하였다. 기존의 기능 검사는 리눅스 환경의 verilog-HDL 시뮬레이터인 icarus-verilog [12]를 사용하고 BMC는 hw-CBMC를 사용했다. 그리고 제안한 방식이 적용된 도구는 CTL 모델 체커인 VIS를 사용하여 구현된 Sbox의 기능을 검증하였다.

본 논문의 실험 환경은 [표 1]과 같다. 각 도구의 공정한 비교를 위하여 CPU는 Pentium3 733㎒, RAM은 512M 그리고 운영체제는 Linux 2.6인 서버에서 공동으로 실험을 수행했다.

	실험 환경
CPU	Pentium3 733㎒
RAM	512M
운영체제	Linux 2.6
실험 도구	icarus-verilog ver.0.8 hw-CBMC ver.2.3 VIS ver.2.1

[표 1] 실험 환경

[표 2]은 조합회로로 구현된 Sbox와 NIST에서 제공하는 ROM으로 구현된 Sbox가 동치임을 보인 실험 결과이다. Event-Driven 방식을 사용하는 icarus-verilog의 1548 assign 문에 대한 실행시간은 0.609 초이다. hw-CBMC는 1548개의 assign 문을 CNF로 변환했을 때, 495773개의 literal과 1479843개의 claus가 생성된 것을 확인할 수 있고 수행시간은 10719 초가 소요됐다. VIS에서는 구현된 모델을 상태 기반 시스템으로 변환했을 때 18164개의 상태가 생성되었고 수행시간은 0.11초이다.

실험 도구	탐색 공간	수행 시간(초)
icarus-verilog	1548 assignments	0.609
hw-CBMC	495773 literals 1479843 claus	10719
VIS	18164 states	0.11

[표 2] 실험 결과

4. 결론 및 향후 연구 방법

설계된 하드웨어 개발에 있어서 시스템의 안전성 보장을 위해 기능 검사에 대한 연구가 활발히 진행되고 있다. 그러나 조합회로에 대한 기능 검사를 위한 도구 개발이나 연구들은 초기의 환경을 벗어나지 못하고 있다. 새로 제안되는 정형적 방법에 의한 기능 검사들이 모델의 표현에 자원의 한계를 넘어서야 하는 큰 제약점이 있기 때문이다. 기존의 기능 검사는 사용자의 요구에 의해 하드웨어 시스템이 복잡해지고 정보산업의 발전에 따라 개발 주기가 점점 빨라지는 시장의 특성으로 인해 설계자에게 많은 시간적 경제적인 부담감을 준다. 이러한 부담을 줄이기 위해 많은 EDA 업체들은 시뮬레이터 가속기와 같은 도구를 시장에 내놓고 있지만 여전히 설계의 정확성 확인

절차는 설계자에 의해 판단이 되도록 하는 틀을 벗어나지 못하고 있다.

본 연구에서는 설계자에게 가중되는 부담을 극복하고 보다 효율적인 기능 검사를 위해 동치성 검사를 모델 체킹에 적용하는 방법을 제안하고 있다. 동치성 검사를 적용한 모델 체킹은 다음과 같은 특징을 지닌다.

첫째, 조합회로에 대한 정확성 확인을 자동으로 수행할 수 있는 방법론을 제시함으로 기존의 하드웨어 시스템 개발 절차를 간결하게 하였다.

둘째, 오토마타로 구현되는 property module에 대한 모델 체킹을 통해 정확한 속성을 검증한다.

셋째, property module, miter module을 BIST와 같은 테스트에 적용할 수 있다.

그러나 논문에서 제안된 방식으로 조합회로의 기능 검사를 수행하기 위해서는 비교적 복잡한 miter 구조를 구현해야 된다.

향후 연구로는 조합회로의 기능검사를 수행하기 위한 miter 구조를 자동으로 구현해주는 방법을 모색하겠다.

Reference

[1] Edmund M. Clarke jr., Orna Grumberg, and Doron A. Peled, *"Model Checking"*, The MIT Press, 1999.

[2] R. Drechsler and D. Horeth, *Gatecomp: Equivalence checking of digital circuits in an industrial environment.* In Int'l Workshop on Boolean Problems, page 195-200, 2002.

[3] A. Biere, A. Cimatti, E.M. Clarke, M. Fujita, and Y. Zhu. *Symbolic model checking using SAT procedures instead of BDDs.* In Design Automation Conf., pp.317-320, 1999.

[4] Jawahar Jain, Amit Narayan, M. Fujita, A. Sangiovanni-Vincentelli, *Formal Verification of Combinational Circuits.* In VLSI Design Conf., pp.218-225, 1997.

[5] Goldberg E., Novikov Ya. *BerkeMin: A fast and robust SAT-solver.* Design, Automation, and Test in Europe (DATE '02), pp. 142-149, March 2002.

[6] MiniSat web page. http://www.cs.chalmers.se/Cs/Research/FormalMethods/MiniSat/Main.html

[7] Silva j., Sakallah K. GRASP: *A Search Algorithm for Propositional Satisfiability.* IEEE Transitions of Computers, 1999, Vol. 48, pp.506-521.

[8] VIS web page. http://vlsi.colorado.edu/~vis/

[9] SMV web page. http://nusmv.irst.itc.it/

[10] Edmund Clarke, Daniel Kroening, Karen Yorav, *Behavioral Consistency of C and Verilog Programs Using Bounded Model Checking*, DAC 2003, pp.368-371.

[11] Joan Deamen, Vincent Rijmen, *"The Design of Rijndael"*, Springer-Verlag Berlin Heidelberg 2002.

[12] icarus-verilog web page. http://www.icarus.com/eda/verilog/index.html