SMV를 이용한 Pipeline 시스템의 설계 검증

이 승 호, 이 현 룡, 장 종 권
울산대학교 컴퓨터정보통신공학과
전화 : 052-259-1637 / 핸드폰 : 011-9500-9214

On a Design Verification of the Pipelined Digital System Using SMV

Seung-ho Lee, Hyun-ryong Lee, Jong-kwon Chang
Dept. of Computer Engineering and Information Technology, University of Ulsan
E-mail : dltmdgh4@daum.net

Abstract

Design verification problem is emerging as an important issue to detect any design errors at the early stage of the design. Conventionally, design verifications have been done using a simulation technique. However, this technique has been proved not to cover all potential design errors. Therefore, formal technique is often used to verify digital circuits as an alternative.

In this paper we adopted formal verification technique and verified some important properties derived from our pipelined digital systems, using SMV (Symbolic Model Verifier). Our example shows that model checking method (one of formal verification techniques) can be effectively performed in verifying the large digital systems.

I. 서론

VLSI 시스템의 복잡도가 증가함에 따라 설계 오류 가능성이 한층 커지고 있어 오류 검증에 많은 시간과 비용이 든다. 전체 설계 공정 상에서 초기 단계의 설계 오류 발견은 설계 사이클의 단축과 경비 절감을 가져오므로 설계 증명은 매우 중요한 문제로 부상하고 있다. 시스템의 정확한 동작을 검증하기 위해 사용되어온 기존의 시뮬레이션 검증(simulation verification) 방법은 VLSI 시스템이 대형화함에 따라 검사해야할 입력 값이 기하 급수적으로 증가하여 모든 경우를 다룰 수 없는 단점이 있다. 이와 달리, 정형 검정(formal verification)방법은 주어진 특성에 대해 입력 값에 무관하게 수학적으로 정확성을 보장하며, 그 특성이 정확하지 않을 경우 반증(counter-example)을 제시함으로써 설계 에러 추적을 할 수 있다. 이러한 특성으로 인하여 정형 검증방법은 시뮬레이션 검증의 보완 방법으로 사용되고 있다.

본 논문은 정형 검증 방법 중, 모델 체킹 알고리즘을 채택하고 있는 SMV(Symbolic Model Verifier)[1]검증 도구를 사용하였다. 검증 대상 시스템으로는 3단계 파이프라인을 Verilog 언어로 재구성하여 설계하고, CTL(Computation Tree Logic)[2.3.4]로 그 특성들을 기술하고 검증하였다.

II. 관련 연구

2.1 정형 검증

정형 검증은 정형 논리(formal logic)나 수리 논리(mathematical logic)를 이용하여 설계대상 시스템(target system)이 설계 명세서(specification)와 일치하

여 동작하는 지를 검증하는 방법이다. 정형 검증은 BDD(Binary Decision Diagram)[5,6]를 기반으로 하고, 등가성 검사(Equivalence Checking), 모델 검사(Model Checking)[2,3] 그리고 정리 증명(Theorem Proving)으로 구분된다. 각각의 검증 방법은 다음과 같다.

등가성검사 - 두 회로의 논리기능이 모든 입력 조건에 대하여 항상 동일한 지의 여부를 검사하는 방법이다.

모델검사 - 주어진 시스템이 검증하고자 하는 특성을 만족하는지를 알아보기 위해서 전체 상태 공간을 검사하는 방법이다.

정리증명 - 시스템과 증명하고자 하는 특성을 수학적 논리를 이용한 논리식으로 표현하고, 공리들과 규칙들을 사용하여 시스템의 정확성을 검사하는 방법이다.

2.2 CTL

CTL(Computation Tree Logic)[2.3.4]은 유한 모델에서 임의의 상태가 주어진 논리식을 만족하는지를 효율적으로 검증하는 고정 특성을 갖추고있고, 경로 지시자(path quantifier)에 시제 연산자(temporal logic)가 결합한 형태로 이루어진다.

시제 연산자에는 **G**(globally or invariantly), **F**(sometime in the future), **X**(next time), **U**(until)가 있다. 경로 지시자는 트리(tree)의 가지 구조(branching structure)를 표현하는 것으로, **A**(for all computation path) 와 **E**(for some computation path)가 있다.

CTL의 정형 구성(formal syntax)은 다음과 같다.

P가 atomic propositions의 집합이라 하면,

1. 모든 atomic proposition $p \in P$는 CTL formula이다.
2. f와 g가 CTL formula 이면, $\neg f$, $f \wedge g$, **AX***f*, **EX***f*, **A**(*f***U***g*), **E**(*f***U***g*) 등도 CTL formula이다.

또, CTL formula로 다음의 약어들(abbreviations)을 쓸 수 있다.

AF*f* ≡ **A**(true **U***f*), **EF***f* ≡ **E**(true **U***f*)

EG*f* ≡ ¬**AF**(¬*f*), **AG***f* ≡ ¬**EF**(¬*f*)

2.3 SMV

SMV(Symbolic Model Checker)[1]는 BDDs(Binary Decision Diagrams)[5,6]를 기반으로 하여 유한 상태 시스템(finite state system)이 CTL로 표현된 요구 명세서와 일치하는지를 자동적으로 검증하는 정형 검증 도구이다. SMV는 시스템을 모델링 하는 언어로 SMVL(SMV language)과 Verilog language를 지원한다. SMVL은 시스템의 상태 전이(state transition)의 기술이 용이하고, Verilog language는 동작 기술이 용이하다.

우리가 시스템을 설계하고, SMV를 이용하여 검증한 과정을 그림 1에 나타내었다.

시스템은 RTL(Register Transfer Level)로 Verilog language를 사용하여 설계하였고, SMV에서 제공하는 vl2smv utility(Verilog 언어를 SMVL로 변환하여 주는 utility)로 SMVL로 변환 하였다. 검증할 특성들은 CTL fomula로 기술하고, SMV를 사용하여 설계와의 일치 여부를 검증한다. 일치하지 않을 경우는 SMV는 반증을 보여준다.

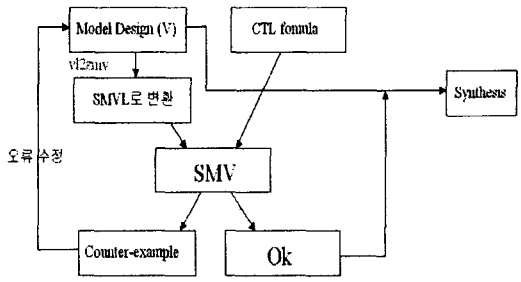

그림 1. 설계 검증 과정

III. Pipeline 재구성

검증 대상 시스템으로 하버드 아키텍처 형태의 5단계 파이프라인 프로세서에서 데이터 메모리와 프로그램 메모리 인터페이스를 제거하고 간략화 시킨 3단계 파이프라인을 재구성 설계하였다.[4,7] 전체 block diagram은 그림 2에 나타내었다.

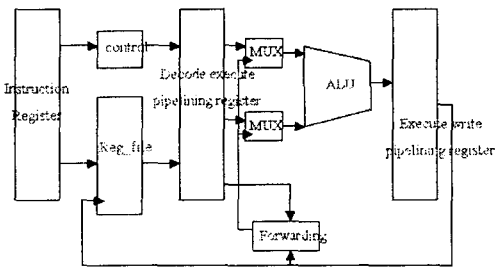

그림 2. 3단계 pipeline 시스템의 Block diagram

각 단계는 decode(read), execute, write로 이루어진

다. 외부 input은 instruction code, clk, reset, stall이 있으며, output은 없다. instruction code의 값에 따라 산술, 논리 연산과 데이터 전송(data transfer)의 두 가지 중 하나의 format이 결정된다. 8-bit instruction format은 그림 3에 나타내었다. clk는 시스템 전체의 외부 클럭(clock)을 나타낸다. reset은 시스템의 모든 register를 reset시킨다. stall은 instruction stream이 멈추었을 때를 표시하는 것으로, 신호 발생 시 pipe-register들을 통하여 어떠한 operation도 전송되지 않는다. 또, Data Hazard의 발생을 막기 위해 Forwarding Unit을 설계하였다. 그림 4는 재구성 설계의 합성 결과를 보여준다.

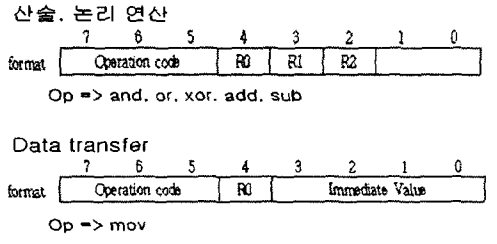

그림 3. 8-bit instruction format

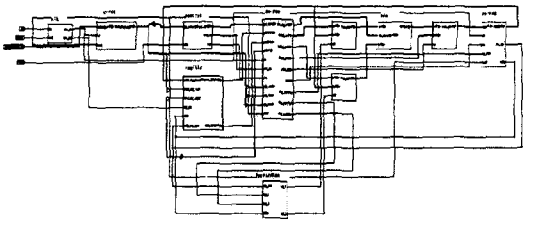

그림 4. Xilinx 5.1i tool을 사용하여 합성한 결과

Ⅳ. 설계 검증

다음의 두 가지 특성(property)을 검증하였다.

4.1 register-file에서 data의 correctness

식(1)은 register-file에서 현재의 data들의 연산 결과가 2-cycle 이후 write 되는가를 검증하는 식이다.

$$\mathbf{AG}((\text{reg\_file.read\_data1} \;\&\; \text{reg\_file.read\_data2}) \rightarrow (\mathbf{AX}(\mathbf{AX}(\text{reg\_file.write\_reg\_data})))) \quad (1)$$

여기서 stall, reset, forwarding은 일어나지 않는다고 가정하고, operation은 AND인 경우를 검증한 것이다. 왜냐하면, stall이 발생하면 어떠한 operation도 전송되지 않으므로 현재의 regiter-file의 data는 2-cycle 이후에도 불변이고, reset이 발생하면 register-file의 모든 data는 0 이다. 또, forwarding이 발생하면 현재의 register-file의 data 값이 아닌 현재의 write할 결과 data값이 연산된다. AND는 instruction format에서 산술, 논리 연산의 Operation중 임의의 선택이다.

4.2 Data Hazard의 발생 시 동작 특성

식(2),(3)은 execute 단계에서의 연산 하고자 하는 register의 data를 write 단계의 목적 register의 data를 요구할 때, write단계의 결과 data를 재 연산하는지를 검증하는 식이다.

$$\mathbf{AG}((\text{ew\_register.reg\_des} = \text{de\_register.reg\_num1\_e}) \rightarrow \mathbf{AF}((\text{ew\_register.result} \;\&\; \text{de\_register.data2\_e}) = \text{ew\_register.data}))) \quad (2)$$

$$\mathbf{AG}((\text{ew\_register.reg\_des} = \text{de\_register.reg\_num2\_e}) \rightarrow \mathbf{AF}((\text{ew\_register.result} \;\&\; \text{de\_register.data1\_e}) = \text{ew\_register.data}))) \quad (3)$$

여기서 stall, reset은 발생하지 않는다고 가정하고, operation은 AND인 경우를 검증하였다.

4.3 검증결과

그림 5은 식(2),(3)을 SMV를 이용하여 검증한 결과이다. 검증 특성과 설계한 시스템이 일치하지 않을 경우는 그림 6과 같은 반증을 보여준다.

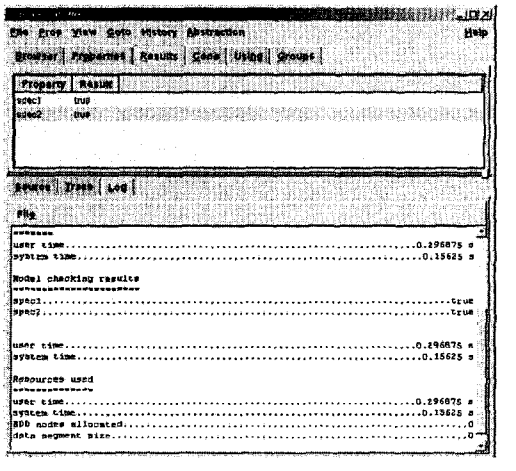

그림 5. SMV를 이용한 검증 결과

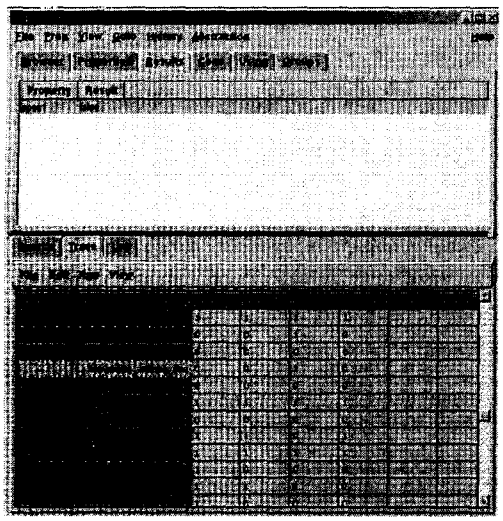

그림 6. counter-example

표 1은 register-file에서 data의 correctness 특성을 AND, XOR, ADD operation에서의 결과를 요약하였다.

Op	word size	BDD nodes allocated	number of states reached	user time
AND	1	9089	8.38e+06	0.078125
	2	13190	1.07e+09	0.109375
	4	34930	4.39e+12	0.140625
	8	61991	7.37e+19	0.53125
XOR	1	9064	8.38e+06	0.0625
	2	13191	1.07e+09	0.125
	4	35243	4.39e+12	0.109375
	8	61990	7.37e+19	0.359375
ADD	1	9057	8.38e+06	0.015625
	2	16876	1.07e+09	0.15625
	4	35966	4.39e+12	0.171875
	8	85292	7.37e+19	0.628125

표 1. register-file의 특성 검정 결과

인텔 펜티엄4(CPU-2.5GHz, Memory-512Mbytes)환경에서 워드의 크기를 증가시키면서 검증하였다. 여기서, state의 지수적 증가에 반해 BDD는 선형적 증가를 하고 있음을 알 수 있다.

V. 결론

정형 검증방법은 정형 논리와 수리 논리를 이용하여 설계 오류를 검증한다. 이 방법은 시스템의 입력 값과는 무관하게 주어진 특성을 검증하며, 검증된 특성에 대해서는 완전성을 보장한다. 정형 검증 방법 중 모델 체킹 방법은 검증 대상 시스템의 설계 사양을 CTL을 이용하여 용이하게 표현할 수 있으므로 대형 시스템의 설계검증에 잘 적용되어진다.

본 논문은 3단계 파이프라인을 재구성 설계하고, 몇 가지 특성들을 SMV를 이용하여 검증하는 사례를 제시하였다. 본 논문의 사례는 다양한 VLSI 시스템 설계 검증에 SMV를 이용한 모델 체킹 방법이 효율적으로 이용될 수 있음을 보여준다.

참고문헌

[1] K. L. McMillan, Cadence SMV, available at http://www-cad.eecs.berkeley.edu/~kenmcmil.

[2] E.M. Clake, E.A. Emerson, and A.P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. ACM Transactions on Programming Languages and Systems, 8(2):244-263, 1986.

[3] Browne, M.C., Clarke, E.M., Dill, D.L., & Mishra, B., Automatic verification of sequential circuits using temporal logic, IEEE Trans. on Computers, Vol. C-35, No. 12, December 1986, pp. 1035-1043.

[4] J. R. Burch, E. M. Clarke, K. L. McMillan, and D. L. Dill. Sequential circuit verification using symbolic model checking. Technical report, Carnegie Mellon University, School of Computer Science, 1990. In Preparation.

[5] Sheldon. B. Akers. Binary Decision Diagrams. IEEE Trans. Comput., vol. C-27, pp. 509-516, June 1978.

[6] Randal. E. Bryant. Graph-based algorithms for boolean function manipulation. IEEE Transactions on Computers, C-35(8), 1986.

[7] David A. Patterson and John L. Hennessy, Computer Organization & Design, Morgan Kaufmann Publisers Inc, pp. 436-529.