Pipeline 시스템의 Hazard 검출기를 위한 BIST 설계

이 한 권, 이 현 룡, 장 종 권
울산대학교 컴퓨터정보통신공학부
전화 : 052-259-1637 / 핸드폰 : 016-9740-2362

BIST Design for Hazard controller in Pipeline System

Han-gweon Lee, Hyun-ryong Lee, Jong-kwon Chang
Dept. of Computer Engineering and Information Technology, University of Ulsan
E-mail : pilineddie@hotmail.com

Abstract

The recent technology developments introduce new difficulties into the test process by the increased complexity of the chip. Most widely used method for testing high complexity and embedded systems is built-in self-test(BIST).

In this paper, we describe 5-stage pipeline system as circuit under testing(CUT) and proposed a BIST scheme for the hazard detection unit of the pipeline system. The proposed BIST scheme can generate sequential instruction sets by pseudo-random pattern generator that can detect all hazard issues and compare the expected hazard signals with those of the pipelined system. Although BIST schemes require additional area in the system, it proves to provide a low-cost test solution and significantly reduce the test time.

I. 서론

최근 VLSI 시스템 집적도의 지속적인 증가로 반도체 단일 칩에 프로세서 및 메모리, 기능 블럭을 내장하고, 이 단일 칩을 이용한 SOC(System On Chip) 설계가 각광을 받고 있다. 이는 시스템 설계 시 발생하는 시간 및 비용을 줄일 수 있고 최소의 하드웨어 overhead로 효율을 얻을 수 있기 때문이다. 반면, 집적도가 증가함에 따라 설계 시 오류 발생 가능성이 한층 커지고 있다. 이러한 문제를 해결하기 위하여 가장 많이 쓰이고 있는 방법이 자체 내장 테스트인 Built-In Self-Test(BIST) 기법이다[1,2,3]. BIST 기법은 제품에 대해 추가적으로 발생할 수 있는 테스트 비용 및 시간을 최소화하고, 시스템 특성에 맞춰 구현이 가능하여 내장형 시스템 설계 검증에 많이 사용되고 있다.

본 논문에서는 BIST 기법을 적용하기 위해 5단계 파이프라인 시스템을 구현하였다. 파이프라인 시스템에서 가장 발생하는 쉬운 오류는 Hazard의 발생에 의한 시스템의 오동작이다. 이러한 Hazard 발생을 검출하고, 오동작을 방지하기 위해 forwarding unit을 구현하였다. 본 논문에서 BIST 회로는 Hazard 발생을 유발하기 위한 Instruction Set을 생성하고, 테스트 대상 회로(CUT, Circuit Under Test)에서 인가하여, forwarding unit에서 올바르게 감지하고 처리되는지를 확인한다.

2장에서는 테스트 대상인 파이프라인 시스템 구조에 대해 설명하고 설계 과정에 대해 기술한다. 3장에서는 시스템에서 hazard 발생 조건에 대해 알아보고, 이를 검출할 수 있는 Instruction Set에 대해 기술한다. BIST 회로의 세부적인 기능과 동작, 테스트 수행 과정은 4장에서 다룬다. 5장에서는 ModelSim tool을 이용한 시뮬레이션 결과를 확인하고, 6장에서 결론을 맺고 향후 연구 방향을 제시한다.

II. 5단계 Pipeline 시스템 설계

테스트 대상 시스템으로 havard 아키텍처 형태의 5단계 파이프라인 프로세서를 설계하였다[4,5]. 개략적인

block diagram은 그림 1에 나타내었다.

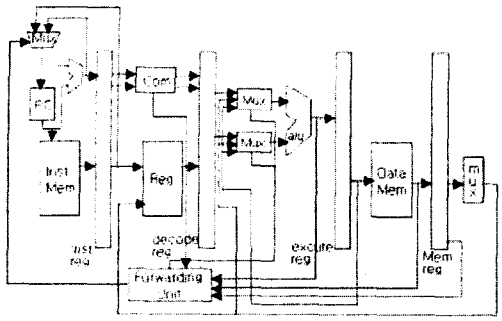

그림 1. 5단계 pipeline 시스템의 Block diagram

각 단계는 fetch, decode(read), execute, memory, write로 이루어진다. 외부 input은 clk, reset, test_in이 있으며, output 신호는 테스트수행 결과인 sucess 신호가 있다. Instruction code에 따라 산술 및 논리 연산과 메모리 load / store, 데이터 전송(data transfer), branch 중 하나의 format이 결정된다.

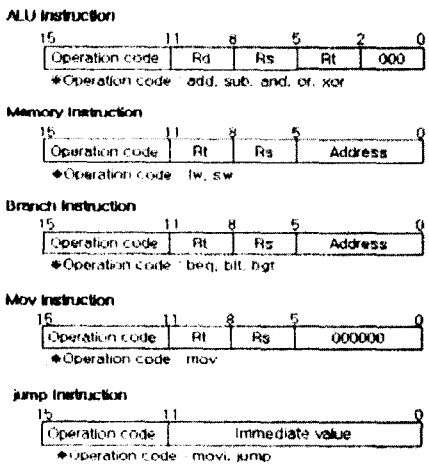

그림 2. 16-bit instruction format

16-bit Instruction format은 그림 2에 나타내었다. clk는 시스템 전체의 외부 클럭(clock)을 나타내며 memory read 및 write를 구분하기 위해 2 clock cycle로 동작된다. reset은 시스템의 모든 register를 reset시키고, test_in 신호는 시스템을 테스트 모드로 전환하기 위한 입력으로 사용된다.

시스템 구현 시 data hazard의 발생을 감지하고 이를 처리하기 위해 forwarding unit을 설계하였다.

III. Hazard 발생 조건

파이프라인 시스템에서 발생 가능한 hazard는 크게 Data hazard와 Structural hazard, Control hazard로 나눌 수 있다[5,6].

Data hazard는 RAW(Read After Write)의 경우에 발생한다. 즉, register에 write하기 이전에 해당 데이터 값을 read하고자할 때 hazard가 발생하게된다.

각 단계에서 발생 가능한 조건은 다음과 같다.

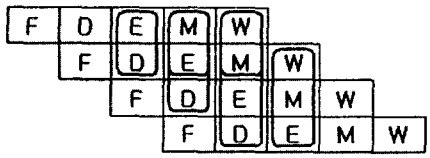

그림 3. 연속된 Instruction 수행 시 Hazard 발생 조건

Data hazard가 발생하는 경우는 그림 3에서의 hazard 발생 조건과 같이 6가지 경우에 해당한다.

1) M->E hazard

현재 excute 단계에서 수행한 결과 값을 write 단계에서 register에 저장하기 이전에 다음 Instruction의 excute 단계에서 해당 register 값을 연산하고자하는 경우 hazard가 발생하며 excute 단계에서 연산결과를 forwarding한다.

2) W->M hazard

write 단계에서 다음 Instruction이 해당 register를 값을 동시에 참조하여 load 또는 store하려는 경우 발생한다.

3) W->E hazard

write 단계에서 다음 Instruction의 excute 단계에서 해당 register를 참조하여 연산하고자 하는 경우 발생한다.

4) E->D hazard

excute 단계에서 연산한 결과값을 register에 write하기 이전에 branch Instruction으로 인해 decode 단계에서 해당 register를 read할 때 hazard가 발생하게된다.

5) M->D hazard

register에 write하기 이전에 branch Instruction이 발생 시 memory 단계에서 forwarding을 수행한다.

6) W->D hazard

register에 write하는 동시에 branch Instruction에서 동일 register를 참조할 때 발생한다.

일반적으로 excute 단계에서 hazard가 발생하며, branch Instruction 수행시에는 decode 단계에서 hazard가 발생한다. lw/sw Instruction에서는 Rs 값과 Address

값을 연산하기 때문에 일반적으로 excute 단계에서 forwarding을 수행하지만, Rt 값이 이전 alu Instruction의 Rd 또는 lw/sw 및 mov Instruction의 Rt 값과 동일 시 W->M hazard로 인해 memory 단계에서 forwarding을 실시한다.

Structural hazard는 시스템의 Instruction 메모리와 Data 메모리를 동일하게 사용하는 경우인 Von Neumann 구조에서 발생하며, 해당 메모리에서 Instruction read 신호와 Data write 신호가 동시 발생할 경우 hazard가 발생하게 된다. 즉, Instruction fetch와 data access의 충돌에 의해 발생한다. 본 논문에서 구현한 시스템은 Instruction 메모리와 Data 메모리를 별도로 사용하여 원천적으로 Structural hazard를 발생시키지 않는 havard 아키텍처를 기반으로 설계하였다.

Control hazard는 분기(branch, jump) Instruction에서 발생하며 해당 조건에 따라 지정된 PC(Program Counter)를 증가시킨 Instruction이 수행된다. 이 경우 일반적으로 flush/stall 신호를 발생하여 Instruction register로 더 이상의 Instruction이 전송되는 것을 방지한다[5]. 본 논문에서 구현한 시스템에서는 2 clock cycle의 원리를 이용하여 branch / jump instruction 수행 시 decode 단계의 첫 번째 clock에서 조건을 확인하여 branch / jump 신호를 발생하고, 두 번째 clock에서 PC 값을 증가시켜 지정된 instruction이 수행된다. 그러므로 flush 신호에 의해 Instruction이 낭비되지 않고 연속된 Instruction을 수행할 수 있다.

IV. BIST 구현

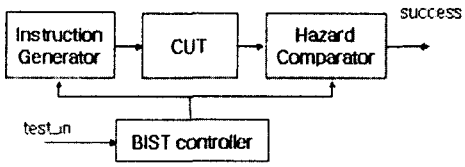

그림 4. BIST 회로구현

BIST 회로는 그림 4와 같이 구성되었다. Instruction generator에서 생성한 연속된 Instruction을 테스트 대상 회로에 인가하고 예상되는 Hazard 신호와 실제 시스템의 forwarding unit에서 발생하는 hazard control 신호를 비교하여 시스템에서 hazard에 의한 오동작 여부를 확인한다.

BIST 회로에서 가장 주요한 부분은 발생 가능한 모든 hazard를 유발하기 위한 Instruction Set을 생성하는 Instruction generator부분이다.

Instruction generator는 Op_code generator와 Register generator로 나누어지며 hazard 발생을 유발하기 위한 연속된 Instruction을 생성한다. 테스트 패턴을 최소화하기 위해 op_code를 기능에 따라 alu, lw/sw, branch, mov의 functional group으로 구분하였으며 그림 5와 같이 각각의 generator에 의해 opcode가 생성된다. op_code 및 register는 pseudo-random test pattern generation 기법에 의해 생성된다[1,3].

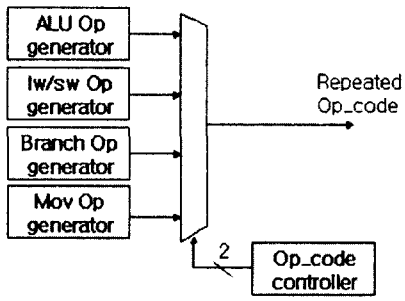

그림 5. Op_code generator 구조

hazard는 register에 write하기 이전에 해당 register를 read하는 경우에 발생한다. 그러므로 연속된 Op_code에 적합한 register를 지정하는 것이 Instruction Set을 구성하는데 주요한 사항이다. 그림 6에서와 같이 각각의 Register 값은 generator에 의해 생성된다. R1, R2, R3는 Instruction에 따라 Rd, Rs, Rt 값으로 지정되며 R1값을 hazard를 발생시키고자 하는 Instruction의 R2 또는 R3에 삽입하여 hazard 발생을 유도하였다.

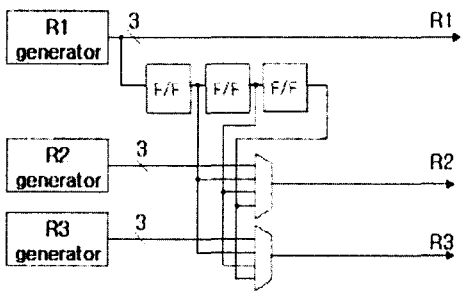

그림 6. Register generator 구조

그림 7은 instruction generator에 의해 생성되어진 연속된 instruction을 보여준다. 최초 3개의 Instruction은 register를 초기화하기 위해 Initial Instruction generator에서 생성되며 그 이후 12개의 Instruction은 control 신호에 의해 반복적으로 생성되어 hazard가 발생할 수 있는 모든 경우의 Instruction set을 검증한다.

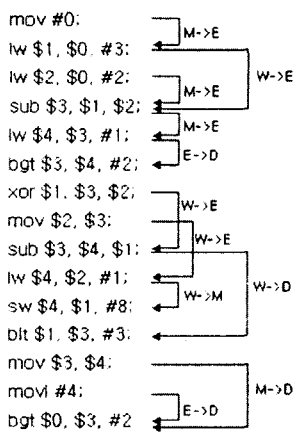

그림 7. Instruction Set

V. 시뮬레이션 결과

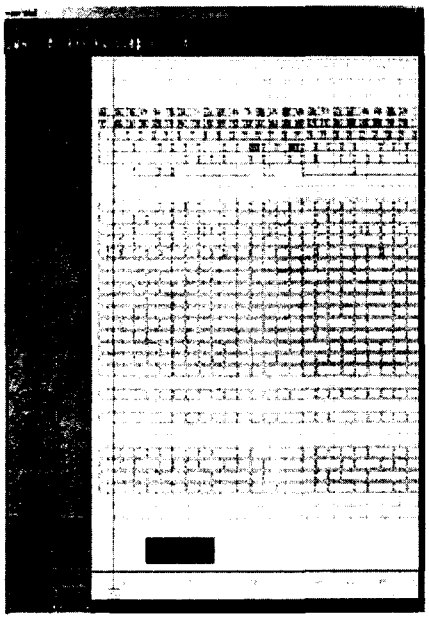

그림 8. Modelsim을 이용한 simulation 결과

그림 8은 ModelSim을 이용한 시뮬레이션 결과이다. 파이프라인 시스템에서 각각의 모듈과 interconnection line 및 bus에서는 고장이 발생하지 않는 것으로 가정하였다. BIST 회로에서 comparator 모듈은 예상되는 hazard 신호에 대해 forward sequence 신호를 발생하여 forwarding unit에서 발생하는 신호와 비교하여 동일 신호가 발생 시 sucess 신호를 내보낸다. 시뮬레이션 결과 success 신호로써 테스트 대상 시스템에서 Instruction 조건에 의해 hazard 발생 시 정상적으로 감지하고 처리하고 있음을 확인할 수 있다.

VI. 결론

FPGA나 ASIC, SOC(System on Chip)와 같이 최근의 VLSI 설계는 고속/고집적도의 반도체를 기반으로 설계가 이루어지고 있다. 이러한 시스템에서의 테스트는 더 많은 기술을 요구하고 있으며, 테스트를 위한 설계(DFT)나 Scan chain 기법, Built-in Self-test(BIST) 기법들이 널리 사용되고 있고 다양한 알고리즘을 바탕으로 테스트가 수행되고 있다.

본 논문은 Havard 아키텍처를 기반으로 5단계 파이프라인 시스템을 설계하고, 파이프라인 시스템에서 오동작을 유발하는 hazard을 검출하기 위해 forwarding unit을 추가하였다. 이 모듈의 동작여부를 테스트하기 위해 BIST 회로를 구성하여 hazard가 발생할 수 있는 Instruction Set을 생성하고, 대상 회로에 인가함으로써 hazard 검출여부를 확인하여 시스템이 정상적으로 동작하는지를 확인하였다. 본 논문의 사례를 통하여 다양한 VLSI 시스템 설계에서 BIST 회로를 이용하여 시스템을 자체 테스트하여 고장 여부를 검증할 수 있음을 확인하였고, 차후 유동적인 register의 개수나 확장된 state에 적용 가능한 BIST 회로를 설계할 수 있다.

참고문헌

[1] Miron Abramovici and Melvin A. Breuer and Arthur D. Friedman Digital system testing and testable design, pp.457-531

[2] Stanley L. Hurst VLSI Testing digital and mixed analogue/digital techniques

[3] Parag K. Lala Digital Circuit Testing and Testability

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

[6] J.L.Hennessy and D.A.Patterson, Computer architecture, a Quantitative Approach

[7] Hideo Fujiwara, Logic Testing and Design for Testabiity

[8] 부산대학교 반도체설계 교육지역센터, Verilog HDL을 사용한 디지털 시스템 설계 및 실습