ISSN: 1226-7244 (Print)
ISSN: 2288-243X (Online)
논문번호 14-03-02
8

j.inst.Korean.electr.electron.eng.Vol.18,No.3,305~312,September 2014
http://dx.doi.org/10.7471/ikeee.2014.18.3.305

FPGA기반 원전용 제어기 코드커버리지 개선

Improving Code Coverage for the FPGA Based Nuclear Power Plant Controller

허 형 석*, 오 승 록*★, 김 규 철**

Hyung-suk Heo*, Seungrohk Oh*★, Kyuchull Kim**

Abstract

IIt takes a lot of time and needs the workloads to verify the RTL code used in complex system like a nuclear control system which is required high level reliability using simple testbench. UVM has a layered testbench architecture and it is easy to modify the testbench to improve the code coverage. A test vector can be easily constructed in the UVM, since a constrained random test vector can be used even though the construction of testbench using UVM. We showed that the UVM testbench is easier than the verilog testbench for the analysis and improvement of code coverage.

요 약

기존의 Verilog테스트벤치로 원전용 안정등급 제어기기와 같이 복잡하고 높은 신뢰도를 요구하는 모듈의 테스트는 수작업으로만 수행된 결과를 가지고 RTL단계의 검증을 마무리하기에는 현실적으로 많은 시간과 노력이 필요하다. UVM은 기존의 테스트벤치의 한계점을 보완하는 계층적 테스트벤치의 구조를 갖고 있어 DUT의 검증을 위한 테스트개선에 대해 테스트벤치의 수정을 간편하게 할 수 있다. 비록 구축과정이 다소 복잡하긴 하지만 테스트벤치의 컴포넌트들인 driver나 sequence 등을 사용함으로 constraint random test를 가능하게 하여 test vector작성을 편리하게 한다. 본 논문에서는 기존의 테스트벤치와 계층적 테스트벤치인 UVM테스트벤치를 사용하여 실제 시뮬레이션 하고 커버리지를 분석하여 코드커버리지를 간편하게 향상 할 수 있음을 보였다.

Key words : FPGA, Code coverage, Nuclear, UVM, Testbench

Ⅰ. 서론

원자력 발전소의 계측제어 시스템은 빠른 처리속도를 만족하는 마이크로프로세서를 기반으로 한 시스템으로 구현이 되어왔다. 그러나 최근엔 원자력 발전소의 안전계통 시스템 구현에 소프트웨어 기반의 시스템에 대한 다양성 확보 방안의 방법으로 프로그래머블 논리 소자 중 하나인 FPGA(Field Programmable Gate Array)가 사용되고 있다[1]. 마이크로프로세서 기반의 소프트웨어에 대한 검증 및 확인에 대한 기술기준은 있지만 FPGA와 같은 프로그래머블 논리소자의 경우 기술 기준이 없었다. 최근 IEC(International Electrotechnical Commission)는 개발 공정이나 특징이 소프트웨어와 다른 HC(HDL-based Iintegrated

* Dept. of Electronics and Electrical Engineering, Dankook University
** Dept. of Applied Computer Engineering, Dankook. University
★ Corresponding author
Dept. of Electronics and Electrical Engineering, Dankook University,
ohrk@dankook.ac.kr, 031-8005-3634
※ Acknowledgment
This work was supported by Korea Ministry of Trade, Industry & Energy(Development of Licensing and Validation Technology)
Manuscript received May. 16, 2014; revised Aug. 1, 2014 ; accepted Aug. 1, .2014

Circuit)에 대하여 원전 안전계통의 개발에 사용할 수 있는 기술 기준인 IEC62566(Nuclear Power Plants-Instrument and control important to safety - Development of HDL-based integrated circuit for systems performing category A function)을 공표하였고 그림 1과 같이 HC 개발 관련 기술 중 하나인 FPGA에 대해 설계 구현 절차에 따른 검증과 확인이 필요한 부분을 명시 하였다.

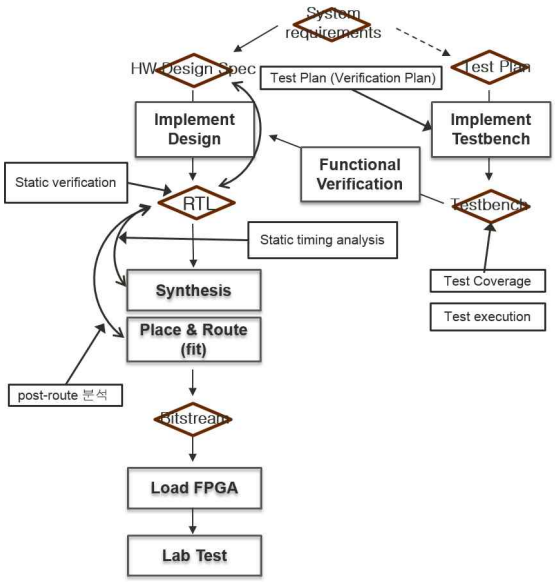


Fig. 1. The verification process of FPGA design
그림 1. FPGA 설계 검증 절차[2,3]

설계자는 하드웨어 시스템의 요구사항과 설계 사양에 따라 하드웨어를 설계한다. HDL(Hardware Design Language)를 사용하여 RTL(Register Transfer Level)의 소스코드형태로 구현하게 되며 이 RTL코드는 합성과정을 통해 게이트들로 표현된다. 이후 합성 결과로 Place&Route를 거치고 생성된 Bitsteam을 FPGA에 로드하면 FPGA의 하드웨어 구현은 마무리 된다. 검증은 각 단계가 전 단계와 기능적으로 일치하는가를 확인하는 것이다. RTL과 설계 사양이 일치하는지를 확인하기 위해서 테스트 입출력을 처리하여 분석할 수 있는 테스트벤치 구축이 필요하다[2]. 검증을 하는 하나의 방법으로 코드커버리지(Code Coverage)와 기능확인(Functional Verification)이 있다. 테스트벤치는 DUT의 기능에 대해 디자인의 오류를 판별하고 검증을 위해 사용되는 모든 환경을 의미한다. 코드커버리지는 구현된 RTL코드가 수행되는지 여부를 분석하는 것으로 기능의 정확성을 검증하지는 않는다[4,5,6]. 기능 확인은 기능의 정확성(Functional Correctness)을 확인하는 것과 기능커버리지(Functional Coverage)로 나눌 수 있다[4]. 기능의 정확성은 테스트벤치의 출력과 예상되는 출력을 비교하여 정확성을 분석 한다. 기능 커버리지는 RTL코드에 포함된 기능이 수행되는지 여부를 테스트하지만 기능의 정확성은 분석하지 않는다. 이와 같은 RTL 수준에서의 검증은 그 검증 대상이 원전, 항공우주와 같이 규모가 크고 안정성과 정확성을 요구하는 하드웨어 일수록 더욱 중요하다. RTL설계 아래 단계에서 오류를 찾아 수정하는 것은 내려갈수록 기하급수적으로 늘어나기 때문에 하드웨어 구현의 초기단계인 상위수준 디자인구현, 즉 RTL 코딩에서의 검증이 만족할 만큼 충분히 이루어져야 전체 설계시간을 절약할 수 있다[7,8]. 본 논문에서는 RTL단계에서의 검증에서 코드커버리지 분석을 위한 테스트벤치 구축 시, 편리하고 효율적인 테스트벤치 구축을 제시하기 위해 Verilog기반의 테스트벤치와 UVM(Universal Verification Methodology)기반의 테스트벤치에 대해 효율성을 비교 분석 하고자 한다.

II. 테스트벤치 종류 및 검증 방법

1. 일반적인 테스트벤치를 이용한 검증

기본적인 테스트벤치는 Verilog나 VHDL 기반으로 구축된 테스트벤치이다. DUT에 신호를 가할 테스트 벡터 생성기가 필요하며 단순히 DUT에 입력을 가하고 그 반응을 봄으로써 가장 기본적인 Blackbox 형태의 하드웨어 검증을 수행한다. 그림 2 에서 보인 간단한 구조의 테스트벤치는 DUT로부터 단순히 입출력을 주고받을 수 있는 구조 이다. 이러한 테스벤치는 복잡하고 대형화된 DUT에 대하여 테스트 신호를 가하기에 많은 시간과 노력이 필요하다. 또한 테스트 시나리오 변화에 따른 테스트벤치의 수정에 많은 코딩작업이 필요하다. 이것은 코드 커버리지를 높이는 작업에도 많은 노력이 필요하다[9].

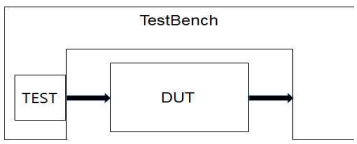


Fig. 2. The simple testbench
그림 2. 간단한 구조의 테스트벤치

2. 계층적 테스트벤치를 이용한 검증

계층적 테스트벤치는 근본적으로 검증 관련 컴포넌트들의 모듈화 그리고 재사용 가능한 테스트벤치의 생성을 지원하기 위해 개발되었으며 검증자로 하여금 스티뮬러스 생성기, 드라이버, 스코어보드, 모니터와 같은 개별적인 검증 컴포넌트에 집중하도록 도와준다. 계층적 테스트벤치의 구조는 그림 3과 같은 구조를 가지고 있다[4].

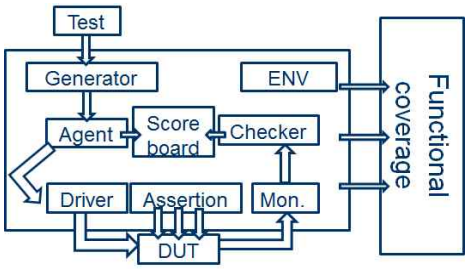


Fig. 3. The layered testbench
그림 3. 계층적 구조의 테스트벤치

계층적 구조를 가진 테스트벤치의 컴포넌트들은 검증자의 테스트 목적에 따라 테스트벤치에서 배제될 수도 있고 사용될 수도 있다. 모니터는 DUT의 버스(bus)를 감시하는 역할을 하며 DUT의 입력과 출력을 감시하여 받은 신호를 스코어보드나 커버리지를 측정하는 곳에 보내게 된다. 스코어보드는 DUT 기능의 정확성을 결정하는 데 사용한다[4]. 전달된 신호들을 분석하여 입력에 따른 예상된 출력과 DUT의 실제 출력을 비교 분석한다. 계층적 구조의 테스트벤치 환경을 구축하는 기초 과정은 테스트 컴포넌트들의 분업화를 이루고 있기 때문에 일반적인 테스트벤치 환경의 구축 과정보다 복잡하다.

가. UVM을 이용한 검증

UVM이란 Universal Verification Methodology 의 약자이며 검증을 위한 효율적인 개발 및 재사용을 가능하게 하는 테스트벤치 구축 방법론 중 하나이다. 표준기구인 Accellera에서 UVM을 위한 API 표준과 SystemVerilog에 기반을 두고 정의된 클래스 라이브러리인 reference implementation을 제공하고 있다[9]. UVM은 객체지향 프로그래밍을 바탕으로 계층적 구조의 테스트벤치를 사용하여 계층적 구조의 테스트벤치가 가지는 장점을 가지고 있다. 객체지향 프로그래밍의 장점은 테스트벤치를 구축하면 테스트에 대한 테스트벤치의 재사용, 수정 및 관리가 용이하다는 것이다[6]. 또한 클래스와 모듈을 자유롭게 혼합함으로 RTL과 DUT의 연결이 매우 쉽고 HVL(Hardware Verification Language)을 사용하는 검증자들이 요구하는 유연성과 재사용성을 만족한다[9].

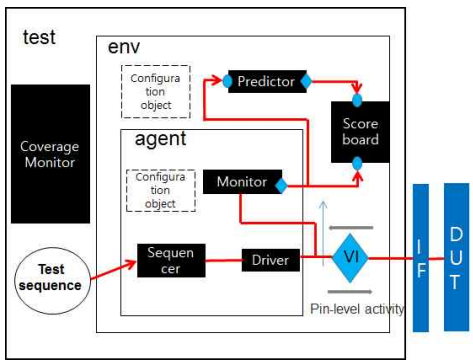


Fig. 4. UVM testbench
그림 4. UVM testbench]

UVM은 SystemVerilog, Verilog VHDL을 사용한 RTL 모델에 동일한 테스트벤치 구축환경을 사용할 수 있다.

나. UVM 테스트 벤치

UVM을 사용한 하드웨어 테스트에 대한 장점은 계층적 구조의 테스트벤치를 적용함으로써 여러 개의 테스트 시나리오에 맞게 테스트를 진행 할 수 있으며 테스트 벡터에 대한 constratined random테스트, coverage_driven verification이 가능하다.

또한 테스트 클래스를 생성함으로 여러 개의 테스트를 모아 관리할 수 있다. 테스트 클래스에서 공통적으로 수행해야하는 일은 테스트환경 생성, 테스트 환경설정 내의 virtual interface 연결, 테스트 시나리오

에 따른 sequence 선택 등 이다. 그림 5는 virtual sequence를 이용하여 테스트 입력의 요소인 sequence item들을 테스트 시나리오에 맞게 만든 후 테스트 입력으로 사용하는 개념도 이다.

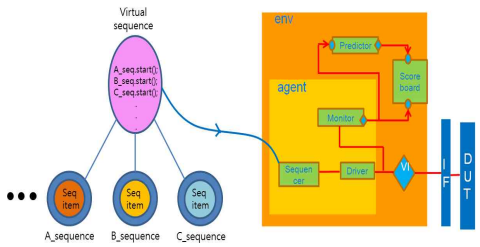


Fig. 5. Test vector using Virtual equence
그림 5. Virtual Sequence의 관리에 의한 테스트 벡터

3. 코드 커버리지(Coverage)

코드 커버리지 종류는 Toggle Coverage. Branch Coverage, Path Coverage, Statement Coverage 등이 있다. 코드 커버리지는 만들어진 RTL코드 중 수행되지 않는 코드가 있는지 확인하는 방법이다. 만약에 수행되지 않은 코드가 존재하면 일상적인 입력에 대해서는 오류를 발생시키지 않지만 예상치 못한 입력이 인가되면 오류가 발생할 수 있기 때문에 수행되지 않는 코드가 존재하는지 확인하는 것이다. 코드 커버리지는 기능의 정확성을 확인하는 것은 아니고, 코드의 수행유무만을 검증하는 것이다.

Ⅲ. Verilog기반 테스트벤치와 UVM기반 테스트벤치의 비교분석

본 논문에서는 원자력 보호계통 제어기에 사용되는 모듈에 대한 테스트벤치를 예시로 이상적인 테스트 커버리지를 목표로 한 Verilog기반 테스트벤치와 UVM기반 테스트벤치의 구축, 수정, 관리에 관하여 시뮬레이션을 하고, 비교분석 한다.

1. DUT 설명

그림 6은 원자로 보호계통 제어기에 사용되는 입/출력 모듈로서, 연산 모듈로부터 값을 입력 받은 후 데이터 처리를 통해 적절한 값을 외부로 출력하는 기능을 한다. 연산모듈로부터 오는 명령은 set, out, chk_id, reset, clear, enable, disable, user0, user1 등이 있으며 이들 명령 중 직렬버스로 다시 출력하는 명령은 set, out, chk_id 등이 해당한다. 나머지 명령들은 입/출력 모듈을 제어하기 위한 제어신호로써 사용된다.

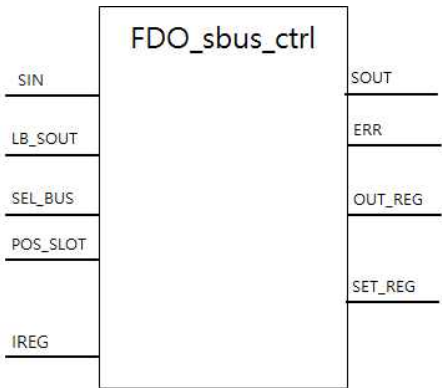


Fig. 6. sbus_ctrl module in FDO
그림 6. FDO sbus_ctrl 모듈

DUT는 다음 그림 7과 같은 직렬 버스 프로토콜을 갖고 있다. 각 필드는 16bit로 구성된다.

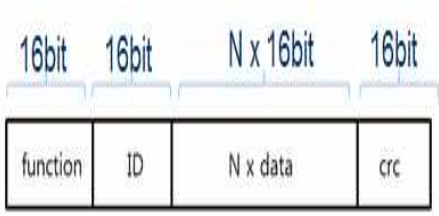


Fig. 7. The protocol of FDO module
그림 7. FDO 모듈의 프로토콜

2. 테스트벤치구축 비교분석

가. Verilog 테스트벤치

Verilog를 기반으로 구축한 테스트벤치는 해당 DUT의 입력 포트에 값과 시간지연을 할당해 주며 직관적인 테스트가 가능하다. 통신 프로토콜을 테스트벡터로 주기위해 top 테스트벤치에 직접적인 입력포트에 값을 할당함으로 그림 8과 같이 작성함으로 테스트입력을 줄 수 있다. 이때 초기 테스트를 위해 대략 300줄 이상의 코딩이 수작업으로 필요하였다.

```
initial begin
     repeat(1000)begin
#200ns;
SBUS.sin=1
i = 0;
parity =0;
SBUS.sin = 1'b0;
#192ns;
crc_int[15:0] = 16'hffff;
cmd_random = $random()
                              if(cmd_random == 1)begin
                                   for_random=0;
                              end else begin
                                   for_random=1;
                              end
funcfifo[15:0] = {1'b0, CFG.sel_sbus , 1'b0,
CFG.pos_slot, 1'b0,cmd_random,for_random,1'b0,4'h0 };
midfifo[15:0] = MID;
cmd=funcfifo[7:0];
                    repeat(16) begin
                    SBUS.sin = funcfifo[i];
                    parity ^= SBUS.sin;
                    xors = crc_int[15]^SBUS.sin;
                    xorf = xors^crc_int[14];
                    xort  = xors^crc_int[1];
                    xorz  = xors;
                    crc_int[15:0] = {crc_int[14:12], xorf,
                    crc_int[10:2], xort, crc_int[0], xorz };
                    i++;
                    #192ns;
                    end
                    .
                    .
```

Fig. 8. A part of source code in verilog testbench
그림 8. verilog 기반 소스코드의 일부분

```
Coverage Report Summary Data by DU

Design Unit: work.fdo01_sbus_ctrl(behavioral)
    Enabled Coverage        Active      Hits    Misses % Covered
    ----------------        ------      ----    ------ ---------
    Stmts                      230       201        29      87.3
    Branches                   204       170        34      83.3
    FEC Condition Terms         44        17        27      38.6
    FEC Expression Terms        14         7         7      50.0
    States                      14        14         0     100.0
    Transitions                 35        25        10      71.4
    Toggle Bins               1498       841       657      56.1
```

Fig. 9. The code coverage of FDO
그림 9. FDO 의 코드커버리지 결과

그림 9는 일반적인 테스트벤치로 DUT의 기능검증을 한 커버리지 결과이다. 1000개의 테스트 패턴을 가하고 결과를 확인하였다. 커버리지중 중요한 지표인 Statement 커버리지와 Branch 커버리지는 각각 87.3%, 83.3%이다. 테스트벤치 전체 코드 커버리지는 75.8%로 커버리지를 높이기 위해 테스트에 개선이 필요하다.

나. UVM 테스트벤치

계층적 구조의 테스트벤치로 SystemVerilog 기반의 UVM 테스트벤치는 초기 구축 시 일반적인 테스트벤치보다 복잡하지만 테스트시나리오 변경과 관리에 용이하여 정밀하고 세부적인검증에 보다 간편하게 접근할 수 있다. 그림 10은 UVM 테스트벤치 컴포넌트중 하나인 드라이버이다. 드라이버는 DUT와 직렬버스의 통신 프로토콜을 지정해 줌으로써 테스트백터에 대해 일일이 통신프로토콜을 명시할 필요가 없이 sequence 들만 랜덤화 하면 자동으로 프로토콜에 대한 테스트 벡터가 생성이 된다.

```
class sbus_driver extends uvm_driver #(sbus_seq_item,
sbus_seq_item);
`uvm_component_utils(sbus_driver)
virtual sbus_if sbus
sbus_agent_config m_cfg
.
.
endclass: sbus_driver
task sbus_driver::run_phase(uvm_phase phase);
  sbus_seq_item req
forever
begin
seq_item_port.get_next_item(req);
.....
                crc_int[15:0] = 16'hffff;
                funcfifo[15:0] = {1'b0, req.sel_sbus , 1'b0,
req.pos_slot, req.cmdid};
                midfifo[15:0] = req.midreg
                              req.data=funcfifo
                              req.addr=16'h0000;
                              repeat(16) begin
                              sbus.sin = funcfifo[i];
                              end
                              .
                              .
                              .
    seq_item_port.item_done();
  end

endtask: run_phase
......
  end
```

Fig. 10. The driver of UVM
그림 10. UVM 드라이버

UVM을 사용한 테스트벤치는 드라이버의 입력 sequence의 조합을 사용하므로 테스트 벤치 코드 사이즈를 Verilog기반 테스트벤치에 비해 100줄 가량 줄일 수 있었다. Verilog기반 테스트벤치는 매번 테스트입력을 만들어서 사용하게 되어 UVM 테스트벤치보다 많은 노력이 필요하고 테스트 대상이 복잡하고 대형화 된 모듈일수록 그 노력은 배가 된다. 그림 11은 UVM 테스트벤치로 DUT의 기능검증을 한 커버리지 결과이다. 1000개의 테스트 패턴을 가하고 결과를 확인하였다. 커버리지중 중요한 지표인 Statement 커버리지와 Branch 커버리지는 각각 87.3% 83.3%로 Verilog 테스트벤치와 동일하고 Toggle Bins에 작은 차이만 보이고 있다. 전체 코드 커버리지는 76.1%로 역시 커버리지를 높이기 위해 테스트에 개선이 필요하다.

Coverage Report Summary Data by DU

Design Unit: work.fdo01_sbus_ctrl(behavioral)

Enabled Coverage	Active	Hits	Misses	% Covered
Stmts	230	201	29	87.3
Branches	204	170	34	83.3
FEC Condition Terms	44	17	27	38.6
FEC Expression Terms	14	7	7	50.0
States	14	14	0	100.0
Transitions	35	25	10	71.4
Toggle Bins	1498	841	657	56.1

Fig. 11. FDO code coverage in UVM testbench
그림 11. UVM testbench을 이용한 FDO 코드커버리지

4. 커버리지개선을 위한 테스트벤치 비교 분석

가. Verilog 테스트벤치 커버리지 개선

일반적인 테스트벤치로 커버리지를 높이기 위해선 구성한 테스트에 기능검증 중 놓친 부분을 일일이 찾고 코드에 추가함으로서 커버리지를 높일 수 있다. 이러한 작업은 디자인이 복잡한 하드웨어의 모든 내부 경로를 테스트하기 위해 많은 시간과 노력이 필요하다.

Coverage Report Summary Data by DU

Design Unit: work.fdo01_sbus_ctrl(behavioral)

Enabled Coverage	Active	Hits	Misses	% Covered
Stmts	230	222	8	96.5
Branches	204	185	19	90.6
FEC Condition Terms	44	17	27	38.6
FEC Expression Terms	14	14	0	100.0
States	14	14	0	100.0
Transitions	35	25	10	71.4
Toggle Bins	1498	1303	195	86.9

Fig. 12. Improved code coverage in verilog testbench
그림 12. verilog 테스트벤치에서 개선된 코드커버리지

그림 12는 개선된 일반 테스트벤치의 결과를 보인다. Statement, Branch 커버리지는 각각 96,5%, 90,6%로 증가 하였으며 전체 코드 커버리지는 84.7%로 증가하였다. 이와 같이 개선된 결과를 얻기 위해 일반 테스트 벤치는 100줄 이상의 추가적인 코드의 수정과 삽입이 필요로 했다.

나. UVM 테스트벤치 커버리지 개선

그림 13은 Virtual_sequence가 입력테스트 벡터의 변수로 사용될 sequence item을 고정시키거나 부분 randomize하여 적은 양의 코드 수정을 통해 여러 테스트시나리오를 쉽게 추가, 관리 하여 테스트의 커버리지를 높이는 작업을 보여 준다. 그림 13에서 각각의 sequence는 테스트 벤치 초기 구축 시 만들어진 프로토콜이나 테스트패턴을 호출하여 사용한다. 기존 sequence를 호출할 때마다 sequence item의 파라미터를 수정하여 테스트 시나리오에 맞는 테스트 sequence를 손쉽게 추가 하였다. 이러한 UVM의 특성은 커버리지를 높이기 위한 테스트벡터의 수정과 생성을 기존의 Verilog 테스트벤치와 다르게 적은 코드의 양만 손쉽게 추가, 수정함으로 개선된 테스트를 할 수 있다. 1000개의 테스트패턴을 입력으로 가했을 때 91.5%의 기능 커버리지를 보였다. UVM은 이와 같은 결과를 얻기 위해 테스트케이스 추가, 수정에 큰 어려움이 없으며 테스트를 쉽게 관리 할 수 있다. 테스트코드의 증가는 50줄 내외로 거의 없었으며 수정작업이 주로 필요하였다.

```
function new(string name = "sbus_v_seq");
  super.new(name);
endfunction
task body;
  // Sequences to be used
  ireg_seq A_seq = ireg_seq::type_id::create("A_seq");
  cfg_seq B_seq = cfg_seq::type_id::create("B_seq");
  sbus_seq C_seq = sbus_seq::type_id::create("C_seq");
  basic_seq E_seq = basic_seq::type_id::create("E_seq");
  super.body;
                #10000ns;
repeat(1000)begin
fork
            B_seq.randomize();
            C_seq.sel_sbus=B_seq.sel_sbus;
            C_seq.pos_slot=B_seq.pos_slot;
            A_seq.randomize();
            B_seq.start(cfg_SQR);
        C_seq.randomize();
            C_seq.start(sbus_SQR);
            A_seq.start(ireg_SQR);
        .
        .
        .
        C_seq.sel_sbus=randombit[1:0];
            C_seq.pos_slot=4'hf;
            C_seq.randomize(cmdid, midreg);
            A_seq.randomize();
        .
        .

join
.
.
end
```

Fig. 13 Modification of virtual sequence
그림 13. virtual sequence 의 수정

Coverage Report Summary Data by DU

Design Unit: work.fdo01_sbus_ctrl(behavioral)

Enabled Coverage	Active	Hits	Misses	% Covered
Stmts	230	224	6	97.3
Branches	204	188	16	92.1
FEC Condition Terms	44	17	27	38.6
FEC Expression Terms	14	14	0	100.0
States	14	14	0	100.0
Transitions	35	25	10	71.4
Toggle Bins	1498	1350	148	90.1

Fig. 14. Improved code coverage in UVM
그림 14. UVM 테스트 벤치에서 개선된 코드커버리지

Ⅳ. 결 론

기존의 Verilog테스트벤치로 원전용 안정등급 제어기기와 같은 복잡하고 높은 신뢰도를 요구하는 모듈의 테스트는 수작업으로만 수행된 결과를 가지고 RTL단계의 검증을 마무리하기에는 현실적으로 많은 시간과 노력이 필요하다. 그리하여 본 논문에서는 기존의 테스트벤치와 차별성을 둔 계층적 테스트벤치인 UVM 테스트벤치를 사용하여 실제 시뮬레이션 하고 커버리지를 분석함으로 해당 테스트가 원전용 시스템에 어느 정도 수행이 되었는지를 판단하고 지속적인 커버리지의 조정을 위해 테스트 케이스/시나리오에 대한 개선 노력을 기존 Verilog테스트벤치와 비교 분석 하였다. UVM의 검증환경은 기존의 방식보다 테스트의 변화에 대한 테스트벤치의 수정을 줄일 수 있고 간편함을 보였으며 구축과정이 다소 복잡하긴 하지만 테스트벤치의 컴포넌트들인 driver나 sequence 등을 사용함으로 constraint random test를 가능하게 하여 test vector작성에 대한 유연함을 보였다. 향후 신뢰성이 중요한 하드웨어에 대한 검증에 UVM테스트벤치를 사용함으로서 RTL단계에서의 충분한 검증이 가능하고 테스트 벤치 구축에 필요한 많은 노력과 시간들이 절감될 것이며 신뢰성 높은 하드웨어설계와 구현을 할 수 있다.

References

[1] Jong Gyun Choi, Dong young Lee, “Development of RPS Trip Logic based on PLDTechnology”, Journal of KNS, vol. 44, no. 6, pp. 697-699 , 2011
[2] International Electrotechnical Commission (IEC), “IEC 62566-2012 Nuclear power plants - Instrumentation and control important to safety - Development of HDL-programmed integrated circuits for systems performing category A functions”, 01/01/12
[3] Kyuhong Kim, “Integrated Verification Method”, Cadence Korea, 2003
[4] Andreas Meyer, “Principles of Functional Verification”, Nownos, 2003
[5] Lionel Bening, Harry D. Foster, “Principles of Verifiable RTLDesign Second Edition - A Functional Coding Style SupportingVerification Processes in Verilog”, Kluwer Academic Publishers, 2001
[6] Christian B. Spear, “SystemVerilog for Verification: A Guide to Learning the Testbench Language Features”, Springer, 2006
[7] Young Jin Oh, Gi Yong Song, “Implementation of a Verification Environment using Layered Testbench”, Journal of KISPS, vol. 12, no. 2, pp. 145-146 ,2011
[8] Ill-ho Bae, "Implementation of Verification Methodology for A FPGA based Conrol System “, Dankook University, 2012
[9] Mark Glasser, "UVM: The Next Gerneration in Verification Methodology", Methodology Architect, 2011

BIOGRAPHY

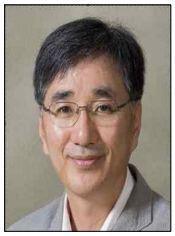
Heo Hyung-suk (Student Member)

2012 : BS degree in Electrical Engineering, Dankook University.
2014 : MS degree in Electrical Engineering, Dankook University.
2014~ : PhD degree course in Electrical Engineering, Dankook University.

Seungrohk Oh (Member)

1980 : BS degree in Electrical Engineering, Hanyang Unv.
1988 : MS degree in Electrical Engineering, Polytechnic Univ.(New York)
1994 : Ph D degree in Electrical Engineering, Michigan State Univ.
1996~ : Professor in Dept. of Electronics and Electrical Engineering, Dankook University.

Kyuchull Kim (Member)



1978 : BS degree in Physics, Seoul National University.
1986 : MS degree in Electrical Engineering, University of Wisconsin at Madison
1992 : PhD Degree in Electrical Engineering, University of Wisconsin at Madison
1993~ : Professor, Dankook University