ISSN: 1226-7244 (Print)
ISSN: 2288-243X (Online)
논문번호 14-04-18

j.inst.Korean.electr.electron.eng.Vol.18,No.4,566~571,December 2014
http://dx.doi.org/10.7471/ikeee.2014.18.4.566

Testbench Implementation for FPGA based Nuclear Safety Class System using OVM

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

Abstract

A safety class field programmable gate array based system in nuclear power plant has been developed to improve the diversity. Testbench is necessary to satisfy the technical reference, IEC-62566, for verification and validation of register transfer level code. We use the open verification methodology(OVM) developed by standard body. We show that our testbench can use random input for test. And also we show that reusability of block level testbench for the integration level testbench, which is very efficient for large scale system like nuclear reactor protection system.

Key words: integration testbench, ovm, functional verification, code coverage, nuclear, control system

I. Introduction

Micro-processor based system has been used for digital nuclear reactor protection system. Recently field programmable gate array(FPGA) based safety class system has been developed for the protection system as a diversity method to improve the safety of nuclear power plant. The development process of FPGA is different from that of micro-process based system. Therefore the verification and validation of FPGA should be different with software verification and validation which is used for micro-process based system. There is no technical reference for the verification and validation of FPGA until 2012. International Electrotechnical Commission(IEC) issued the technical reference of verification and validation for FPGA named "Nuclear power plants - Instrumentation and control important to safety - Development of HDL-programmed integrated circuits for systems performing category A functions(IEC-62566)"[1]. IEC-62566 required the testbench for the verification and validation which is used to get the code coverage and functional coverage[2,3,4,5,6] for the code written by the hardware descriptive language(HDL). We develop the testbench for the actual reactor protection system VHDL code developed by Korea Atomic Energy Research Institute and Doosan Heavy Industrials & Construction. The testbench uses the Open Verification Methodology(OVM)[7,8] which is developed by a standard organization, Accellera. The methodology is open source and very efficient for large scale system like a reactor protection system. OVM provides the library for the testbench components and the library is written by SYSTEMVERILOG which has property that can generate the random test signal. After building the block level testbenchs, we can build the integrated level testbench using the block level testbenchs. The reusability of block level test bench component

* 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 Nov. 25, 2014; revised Dec. 8, 2014 ; accepted Dec. 8, .2014

gives us saving the workload for integrated test. We show that how to build the block level testbenchs for the early version RTL code of a safety class system developed by Doosan Heavy Industrials & Construction and Korea Atomic Energy Research Institute and test results using OVM. We demonstrate the reusability of testbench component used in block level testbench for the integration level testbench. We also have shown that we can get the test results of each block during the integration test without the modification of Design Under Test(DUT).

II. Testbench Build Up

The purpose of a testbench is to analyze the correctness of Design Under Test(DUT). This can be done by the following steps[3]:

- Generate stimulus
- Apply stimulus to the DUT
- Obtain the response from DUT
- Check the properties of the response of DUT to be hold

To this end, one can build the testbench as shown in Fig. 1.

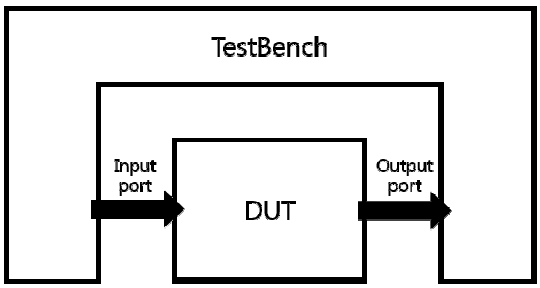


Fig. 1 A simple testbench[3]

We called a simple testbench such as shown in Fig. 1. A test input is applied to input port of DUT and simulate the DUT to check the specifications. A simple testbench is very efficient for small size of DUT. However, if the DUT is large and complex, it requires a long time and lots of workloads[3]. The layered testbech[3] as shown in Fig. 2 uses the smaller pieces of testbench components, namely signal layer component, command layer component, functional layer component, scenario layer component.

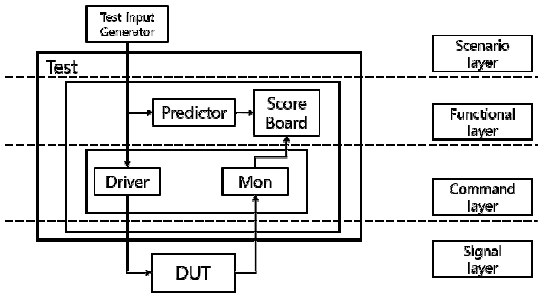


Fig. 2 Layered testbench[3]

Each component of testbench can be developed separately. Also most of modules can be reused when DUT is changed. Standard body, Accellera, develops a layered testbench architecture called Open Verification Methodology(OVM)[7,8]. OVM structure is shown in Fig 3.

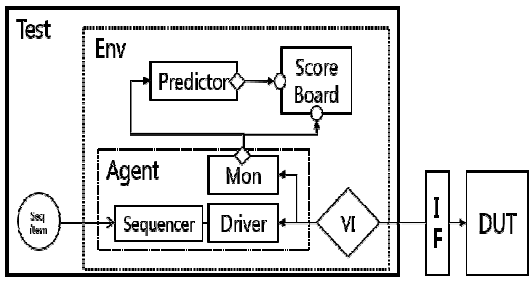


Fig. 3 Open Verification Methodology structure

OVM uses SYSTEMVERILOG which is suitable hardware descriptive language for testbench. The advantages of OVM are the use of random test signal and reusability for the integrated test as we will show later on. Thus the workloads are saved for the test.

1. OVM Structure[2,3]

OVM structure is shown in Fig. 3. The Agent is a set of testbench components which allows to generate and to monitor the pin level transaction. Each component of Agent includes following:

- Sequence item : set of test input vector for DUT
- Driver : covert the sequence item data to pin level transaction
- Sequencer: deliver the sequence item to driver
- Monitor : observe the pin level activity and sent to analysis component such as scoreboard which compare the expected value and DUT output
- Interface : connect the driver and DUT

Analysis component of OVM include the following:
- Scoreboard : check the DUT behavior correctly by comparing the expected value and DUT output
- Predictor : generate the expected value given sequence item
- Coverage collector : monitor the functional coverage using covergoup

2. Testbench build-up using OVM

In order to build up a testbench, we analyze the DUT and identify the testable components of DUT called block level component and set up the strategy how we integrate block level components. As an example we consider a part of early version of digital output module in the FPGA based reactor protection system which is developed by Doosan Heavy Industrials & Construction and Korea Atomic Energy Research Institute in the Fig. 4.

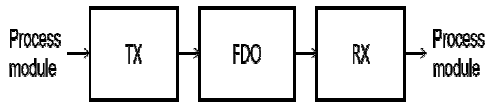


Fig. 4 Integration of the digital output module

TX module convert the parallel data of process module to serial data and its output consists of function, module identification number, data, CRC data and is fed to the digital output module named FDO module. The output of FDO module is function, module identification number, data to be transferred process modules, and CRC data. The output of FDO module is fed to RX module to transfer the data to process module. The outputs of RX module are the data, CRC, and memory address to be stored. Testbench for FDO block is shown in Fig. 5.

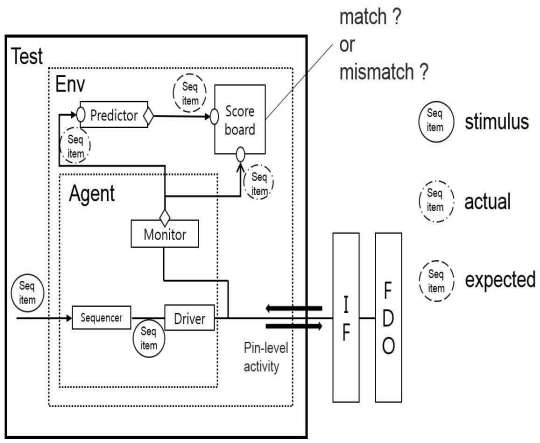


Fig. 5 Testbench for FDO module

We use the sequence items consisting of function, module identification number, data, CRC data as in Fig 6.

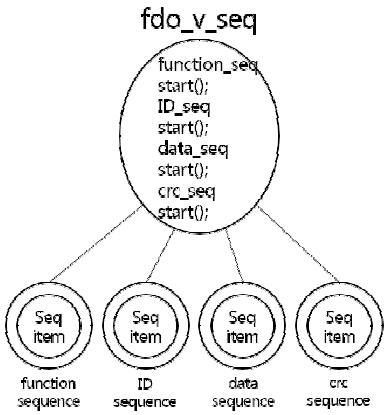


Fig. 6 Sequence item for FDO module

As for test input we generate constrained random sequence for function sequence since the number of function is limited. We generate the random sequences for data. ID sequence is the identification of slot and it is fixed. We compare the DUT output with expected output in the scoreboard. The result of test is shown in Fig 7.
All of 3415 sequence items matched with the expected value. Similarly we develop the testbench for TX block and RX block.

```
#
##############FDO-BLOCKLEVEL SCOREBOARD#########################
#
#
# CMD = 20
#  INPUT Serial Word[0] : 2b10  EXPECTED OUTPUT[0] : 2b10
#  INPUT Serial Word[1] : 44f1  EXPECTED OUTPUT[1] : 44f1
#  INPUT Serial Word[2] : 54d2  EXPECTED OUTPUT[2] : 11f0
#  INPUT Serial Word[3] : 4a0a  EXPECTED OUTPUT[3] : 11f1
#  INPUT Serial Word[4] : 5a5e  EXPECTED OUTPUT[4] : 11f2
#  INPUT Serial Word[5] : 45e6  EXPECTED OUTPUT[5] : 11f3
#  INPUT Serial Word[6] : 55ef  EXPECTED OUTPUT[6] : 11f4
#  INPUT Serial Word[7] : 0d5c  EXPECTED OUTPUT[7] : f8dc
#
#
# ***   Test Pattern Passed : 1000      ***
# ***   Expected values matched by CMD : 3415    mismatched : 0        ***
#
```

Fig. 7 Test result for FDO

3. Integration of block level testbenchs

We develop the testbench for the integration of RX, TX, and FDO block as shown in Fig. 8
TX, FDO, and RX block is connected through the interfaces as shown in Fig. 9. Note that the interfaces used in the integration level testbench are defined and used in the block level testbench and we simply connect the TX interface and FDO interface, FDO interface and RX interface for the integration testbench.

```
always @(TX.sout) begin
        FDO.sin=TX.sout;
end
always @(FDO.sout) begin
        RX.sin=FDO.sout;
end
```

Fig. 9 interface connection for the integration level testbench

Since FDO input is fed from TX output and RX input is fed from FDO output in integration level test, we disable the drivers in the FDO agent and RX agent as shown in Fig. 8. The test input is applied to TX Agent, TX output feed to FDO, and FDO output feed to RX. The expected value of RX output and the actual value of RX output is compared in the integration scoreboard in Fig. 8.

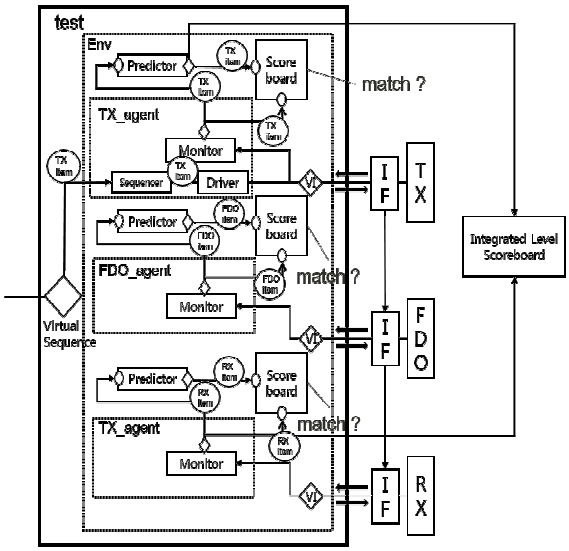


Fig. 8 Integration level testbench

The integration level test results in the integration scoreboard are shown in Fig. 10. We use the 1000 test inputs and there are no errors.

```
#############INTEGRATION-LEVEL SCOREBOARD#########################
#
#
######################
# -Operation Mode = WRITE
######################
# Module ID = Digital Out
######################
#   tx INPUT Reg : 2b20  OUTPUT Serial data : 2b20  fdo INPUT Serial data : 2b20
#                : 44f1                     : 44f1                      : 44f1
#                : 6a71                     : 6a71                      : 6a71
#                : 7848                     : 7848                      : 7848
#                : cc63                     : cc63                      : cc63
#                : xxxx                     : xxxx                      : xxxx
#                : xxxx                     : xxxx                      : xxxx
#                : xxxx                     : xxxx                      : xxxx
#
#OUTPUT Serial data : 2b20  rx INPUT Serial data : 2b20  OUTPUT Reg data : 11f0
#                   : 44f1                      : 44f1                   : 11f1
#                   : 11f0                      : 11f0                   : 11f2
#                   : 11f1                      : 11f1                   : 11f3
#                   : 11f2                      : 11f2                   : 11f4
#                   : 11f3                      : 11f3                   : 9928
#                   : 11f4                      : 11f4                   : xxxx
#                   : 9928                      : 9928                   : xxxx
# **********************test sequence passed : 1000          occured erros : 0*
```

Fig. 10 Integration level test results

The Fig. 11 summarizes the test results in the TX, FDO, and RX scoreboards during the integration test. If we do not verify the block level output during the integration test, it is possible that there is no errors in integration level test even if there

are some errors in the block level. However our integration level testbench can verify the block level test results as shown in Fig. 11 during the integration test without modification of DUT. Note that a modification of DUT is not desirable for a test purpose

```
#############TX-BLOCKLEVEL SCOREBOARD#########################
#
#
# CMD = 20
#  INPUT 16'b Register[0] : 2b20  OUTPUT Serial Word[0] : 2b20
#  INPUT 16'b Register[1] : 44f1  OUTPUT Serial Word[1] : 44f1
#  INPUT 16'b Register[2] : 6a71  OUTPUT Serial Word[2] : 6a71
#  INPUT 16'b Register[3] : 7848  OUTPUT Serial Word[3] : 7848
#  INPUT 16'b Register[4] : cc63  OUTPUT Serial Word[4] : cc63
#  INPUT 16'b Register[5] : xxxx  OUTPUT Serial Word[5] : xxxx
#  INPUT 16'b Register[6] : xxxx  OUTPUT Serial Word[6] : xxxx
#  INPUT 16'b Register[7] : xxxx  OUTPUT Serial Word[7] : xxxx
#
#
# ***   Data matched : 2332    mismatched : 0   ***
```

```
#############FDO-BLOCKLEVEL SCOREBOARD#########################
#
#
# CMD = 20
#  INPUT Serial Word[0] : 2b20  EXPECTED OUTPUT[0] : 2b20
#  INPUT Serial Word[1] : 44f1  EXPECTED OUTPUT[1] : 44f1
#  INPUT Serial Word[2] : 6a71  EXPECTED OUTPUT[2] : 11f0
#  INPUT Serial Word[3] : 7848  EXPECTED OUTPUT[3] : 11f1
#  INPUT Serial Word[4] : cc63  EXPECTED OUTPUT[4] : 11f2
#  INPUT Serial Word[5] : xxxx  EXPECTED OUTPUT[5] : 11f3
#  INPUT Serial Word[6] : xxxx  EXPECTED OUTPUT[6] : 11f4
#  INPUT Serial Word[7] : xxxx  EXPECTED OUTPUT[7] : 9928
#
#
# ***   Expected values matched by CMD : 3355    mismatched : 0        ***
```

```
#############RX-BLOCKLEVEL SCOREBOARD#########################
#
#
# CMD = 20
#  INPUT Serial Word[0] : 2b20  OUTPUT[0] : 11f0
#  INPUT Serial Word[1] : 44f1  OUTPUT[1] : 11f1
#  INPUT Serial Word[2] : 11f0  OUTPUT[2] : 11f2
#  INPUT Serial Word[3] : 11f1  OUTPUT[3] : 11f3
#  INPUT Serial Word[4] : 11f2  OUTPUT[4] : 11f4
#  INPUT Serial Word[5] : 11f3  OUTPUT[5] : 9928
#  INPUT Serial Word[6] : 11f4  OUTPUT[6] : xxxx
#  INPUT Serial Word[7] : 9928  OUTPUT[7] : xxxx
#
#
# ***   Data matched : 3355    mismatched : 0   ***
#
```

Fig. 11 TX, FDO, and RX scoreboard during integration test.

The Fig. 12 shows the code coverage results in the integration level testbenches. We only build the integration scoreboard for integration test, while other components of integration testbench reuse the pre-developed components used in block level testbench. This can save lots of workloads for the integration level testbench of the large scale system like a reactor protection system.

```
Coverage Report Summary Data by DU

Design Unit: work.fdo_in_slot11(behavioral)
    Enabled Coverage        Active      Hits    Misses % Covered
    ----------------        ------      ----    ------ ---------
    Stmts                      214       206         8      96.2
    Branches                   162       154         8      95.0
    FEC Condition Terms         31        26         5      83.8
    FEC Expression Terms         0         0         0     100.0
    States                      10        10         0     100.0
    Transitions                 22        16         6      72.7
    Toggle Bins               1595       839       756      52.6

Design Unit: work.rx(behavioral)
    Enabled Coverage        Active      Hits    Misses % Covered
    ----------------        ------      ----    ------ ---------
    Stmts                       63        56         7      88.8
    Branches                    66        58         8      87.8
    FEC Condition Terms         20        20         0     100.0
    FEC Expression Terms        14         8         6      57.1
    States                       4         4         0     100.0
    Transitions                  7         5         2      71.4
    Toggle Bins                290       273        17      94.1

Design Unit: work.tx(behavioral)
    Enabled Coverage        Active      Hits    Misses % Covered
    ----------------        ------      ----    ------ ---------
    Stmts                       51        47         4      92.1
    Branches                    45        41         4      91.1
    FEC Condition Terms          6         6         0     100.0
    FEC Expression Terms         8         8         0     100.0
    States                       4         4         0     100.0
    Transitions                  7         5         2      71.4
    Toggle Bins                184       183         1      99.4
```

Fig. 12 code coverage results for integration test

Conclusion

In order to satisfy the technical reference for FPGA based safety class system in the nuclear power plant, a testbench for Register Transfer Level(RTL) is necessary to check the properties, i.e., code coverage and functional coverage. We show that a method of building up the testbench using OVM. Since the library of OVM is an open source, someone who is interested in build up a testbench for the safety class system can use OVM without the charge. The proposed method is very efficient for a large scale system such as a reactor protection system because of reusability of block level testbenches in the integration level testbench. And also we can verify the sub-module test results, which imply that we can do the white box test during the integration test. Moreover, the random test capability in the proposed method can increases the code coverage and functional coverage. We show how we construct the test input and

testbench using OVM for actual RTL code used in the FPGA based reactor protection system.

References

[1] International Electrotechnical Commission, *"Nuclear power plants - Instrumentation and control important to safety -Development of HDL-programmed integrated circuits for systems performing category A functions,"* IEC-62566, 2012

[2] Andreas Meyer, *"Principles of Functional Verification"*, Nownos, 2003

[3] Christian B. Spear, *"SystemVerilog for Verification: A Guide to Learning the Testbench Language Features"*, Springer, 2006

[4] Mark Glasser, *"Open Verification Methodology Cookbook"*, Springer, 2009

[5] Lionel Bening, Harry D. Foster, *"Principles of Verifiable RTL Design Second Edition - A Functional Coding Style Supporting Verification Processes in Verilog"*, Kluwer Academic Publishers, 2001

[6] Hyung-suk Heo, Seungrohk Oh, Kyuchull Kim, *"Improving Code Coverage for the FPGA based nuclear power plant controller"*, j.inst.Korean.electr.electron.eng. vol. 18, no. 3, pp. 8-15, 2014

[7] Mentor Graphics Corporation, *"Testbench Guide for OVM"*, Mentor Graphics Corporation, 2011

[8] Mark Glasser, *"Open Verification Methodology Cookbook"*, Springer, 2009

BIOGRAPHY

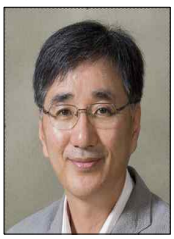
Heo Hyung-suk (Student Member)

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