Improve the quality of SystemC IPs through coverage-driven random verification

Trung Pham, Renesas Electronics Corporation, Ho Chi Minh City, Vietnam (*trung.pham.zn@renesas.com)*

Huy Phan, Renesas Electronics Corporation, Ho Chi Minh City, Vietnam (*huy.phan.wh@renesas.com*)

Masayuki Masuda, Renesas Electronics Corporation, Tokyo, Japan (*masayuki.masuda.gx@renesas.com*)

***Abstract*—The development of SystemC IPs is mainly focused on a short period. Realizing that SystemC IPs can be improved to get higher quality while keeping a good period, we apply UVM to SystemC verification to add coverage-driven random verification besides directed testing. Our solution has the same structure as UVM in SystemVerilog. It provides constraint random by CRAVE and functional coverage by FC4SC. We tried it on a verified SystemC IP. Using directed testing, it originally took 18 man-months and found 127 bugs. We spent about 21 man-months on coverage-driven random verification and found 38 more bugs, 50% of which are hard cases.**

Keywords—SystemC IP, coverage-driven random verification, UVM SystemC, CRAVE, FC4SC

I. Introduction

SystemC IPs [1] are expected to reduce the cost of software development and shortening the development period becomes more important. Of course, the verification environment and verification methodology must not be complicated. Traditional verification based on directed testing was a choice of SystemC IPs verification.

Unfortunately, the limitation of directed testing is indisputable: verifiers need to create individual cases, possibly leading to verification omissions. Rather, a huge effort is necessary to ensure verification coverage (mostly, the effort is reduced at the expense of verification coverage). Besides, verification quality, goals and progress are unclear. It is hard to manage verification requirements and schedules. We realized that the verification method and planning should be changed to improve verification quality while keeping the development period. This is where the UVM SystemC [2] and coverage-driven verification [3] inspired us.

This paper gives an introduction and results of coverage-driven random verification using UVM [4] technologies (verification environment by UVM SystemC library, constrained random stimulus by CRAVE library [5], functional coverage by FC4SC library [6]). Through the results, we can see how efficiently it works.

II. Plan

This paper shows a plan to improve the 2 most typical things which can affect the verification process. We applied coverage-driven random verification using UVM SystemC to SystemC IP to prove that these improvements improve verification quality. The SystemC IP has already been verified using direct testing, and we evaluate the effectiveness of the coverage-driven random verification by comparing the results of both verifications (in IV. RESULT).

A. *Verification plan improvement*

Verifiers need to prepare necessary documents (Target specification) and brainstorm all necessary verification features for the creation of verification planning (planning tied to IP specifications helps to manage verification requirements and schedule, and planning tied to functional coverage clarifies verification goals and progress). Moreover, the coverage of IP specifications also clarifies verification quality. Verification planning is not a one-time effort, it can be refined throughout the course of a project.

B. *Verification environment improvement*

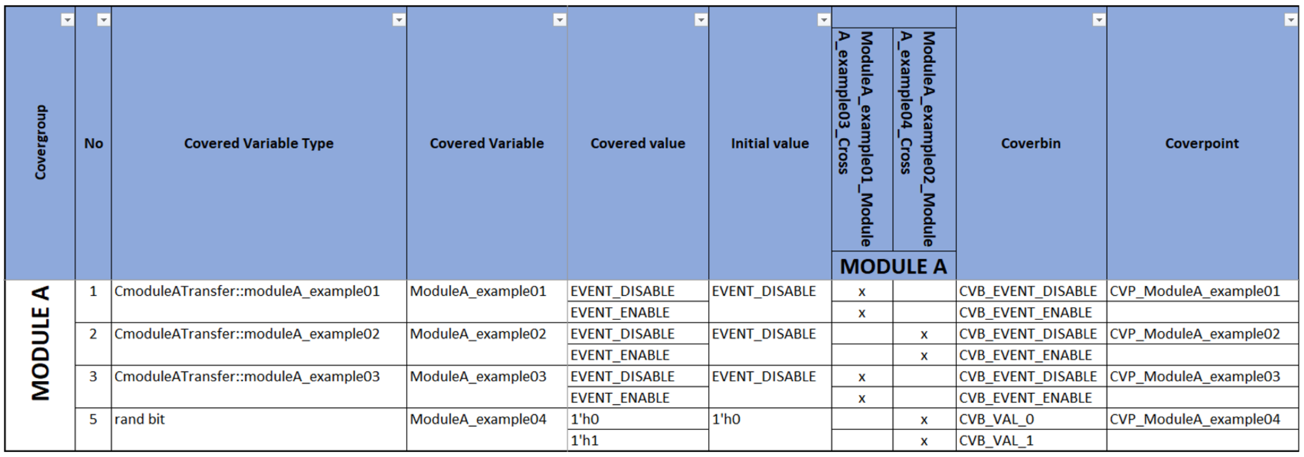
Verifiers need to build a new UVM SystemC IPs environment that followed UVM standardization. It helps to achieve verification goals through effective code reuse (reuse verification components between environments and hierarchies, reuse verification environments between projects). Verifiers can save the verification period by automatic stimulus generation. The constrained random stimulus hits various cases efficiently.

III. IMPLEMENTATION

A. *Verification plan*

The following steps must proceed to ensure improved commitment:

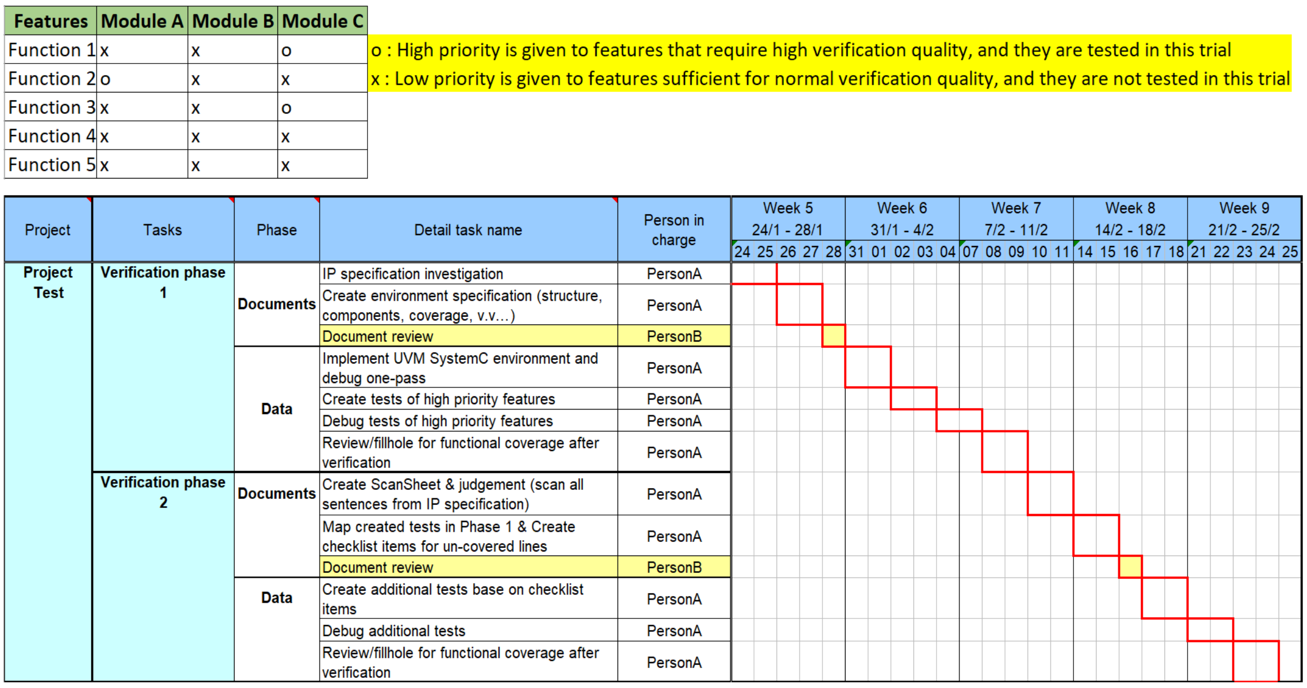
- Investigate verified IP specifications and describe the priority of features in the document.
- Identify supported features and covergroup/coverpoint [7] based on discussion relating to priority between the SystemC IP design team and our verification team. Figure 1 shows an example of covergroup/coverpoint definition.



Covergroup	No	Covered Variable Type	Covered Variable	Covered value	Initial value	MODULE A: ModuleA_example01_ModuleA_example03_Cross	MODULE A: ModuleA_example02_ModuleA_example04_Cross	Coverbin	Coverpoint
MODULE A	1	CmoduleATransfer::moduleA_example01	ModuleA_example01	EVENT_DISABLE	EVENT_DISABLE	x		CVB_EVENT_DISABLE	CVP_ModuleA_example01
				EVENT_ENABLE		x		CVB_EVENT_ENABLE	
	2	CmoduleATransfer::moduleA_example02	ModuleA_example02	EVENT_DISABLE	EVENT_DISABLE		x	CVB_EVENT_DISABLE	CVP_ModuleA_example02
				EVENT_ENABLE			x	CVB_EVENT_ENABLE	
	3	CmoduleATransfer::moduleA_example03	ModuleA_example03	EVENT_DISABLE	EVENT_DISABLE	x		CVB_EVENT_DISABLE	CVP_ModuleA_example03
				EVENT_ENABLE		x		CVB_EVENT_ENABLE	
	5	rand bit	ModuleA_example04	1'h0	1'h0		x	CVB_VAL_0	CVP_ModuleA_example04
				1'h1			x	CVB_VAL_1	

Figure 1. Example of covergroup/coverpoint definition

- Identify verification environment structure (where to implement checkers, components, what attributes should be checked, etc.).
- Create a schedule that separates the verification period into 2 phases to ensure coverage of IP specifications (phase 1 plan to verify all supported features by randomized testing and phase 2 plan to cover remaining points by well-constrained values). Figure 2 shows an example of a priority judgment and verification schedule.



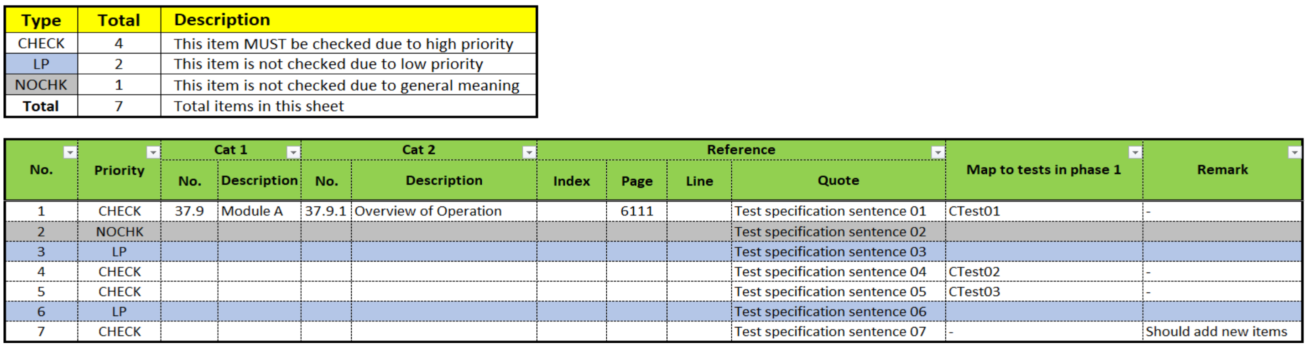
Features	Module A	Module B	Module C
Function 1	x	x	o
Function 2	o	x	x
Function 3	x	x	o
Function 4	x	x	x
Function 5	x	x	x

o : High priority is given to features that require high verification quality, and they are tested in this trial
x : Low priority is given to features sufficient for normal verification quality, and they are not tested in this trial

Project	Tasks	Phase	Detail task name	Person in charge	Week 5 24/1 - 28/1					Week 6 31/1 - 4/2					Week 7 7/2 - 11/2					Week 8 14/2 - 18/2					Week 9 21/2 - 25/2				
					24	25	26	27	28	31	01	02	03	04	07	08	09	10	11	14	15	16	17	18	21	22	23	24	25
Project Test	Verification phase 1	Documents	IP specification investigation	PersonA																									
			Create environment specification (structure, components, coverage, v.v…)	PersonA																									
			Document review	PersonB																									
		Data	Implement UVM SystemC environment and debug one-pass	PersonA																									
			Create tests of high priority features	PersonA																									
			Debug tests of high priority features	PersonA																									
			Review/fillhole for functional coverage after verification	PersonA																									
	Verification phase 2	Documents	Create ScanSheet & judgement (scan all sentences from IP specification)	PersonA																									
			Map created tests in Phase 1 & Create checklist items for un-covered lines	PersonA																									
			Document review	PersonB																									
		Data	Create additional tests base on checklist items	PersonA																									
			Debug additional tests	PersonA																									
			Review/fillhole for functional coverage after verification	PersonA																									

Figure 2. Example of priority judgment and verification schedule

- Create a list of scanned sentences from verified IP specifications and map them to created tests in phase 1, if any items have not been covered yet, record and verify them in phase 2. Figure 3 shows an example of a list of scanned sentences.



Type	Total	Description
CHECK	4	This item MUST be checked due to high priority
LP	2	This item is not checked due to low priority
NOCHK	1	This item is not checked due to general meaning
Total	7	Total items in this sheet

No.	Priority	Cat 1		Cat 2		Reference				Map to tests in phase 1	Remark
		No.	Description	No.	Description	Index	Page	Line	Quote		
1	CHECK	37.9	Module A	37.9.1	Overview of Operation		6111		Test specification sentence 01	CTest01	-
2	NOCHK								Test specification sentence 02		
3	LP								Test specification sentence 03		
4	CHECK								Test specification sentence 04	CTest02	-
5	CHECK								Test specification sentence 05	CTest03	-
6	LP								Test specification sentence 06		
7	CHECK								Test specification sentence 07	-	Should add new items

Figure 3. Example of a list of scanned sentences

Verification can be closed if items created from a list of scanned sentences from verified IP specifications are verified and covergroup/coverpoint are covered (in case of uncovered points/crosses, create more tests to cover if necessary).

B. *Verification environment*

Following the UVM Test Bench architecture [8] of SystemVerilog, the new UVM SystemC IPs environment architecture should be the same. It includes basic components (Top, Test, Environment, Agent, Sequencer, Driver, Monitor and Scoreboard). Because most SystemC IPs have TLM [9] interface for bus access (e.g., registers access), the Driver must support driving TLM sockets through TLM Initiator in addition to driving the Virtual interface. Figure 4 shows a block diagram of the verification environment.

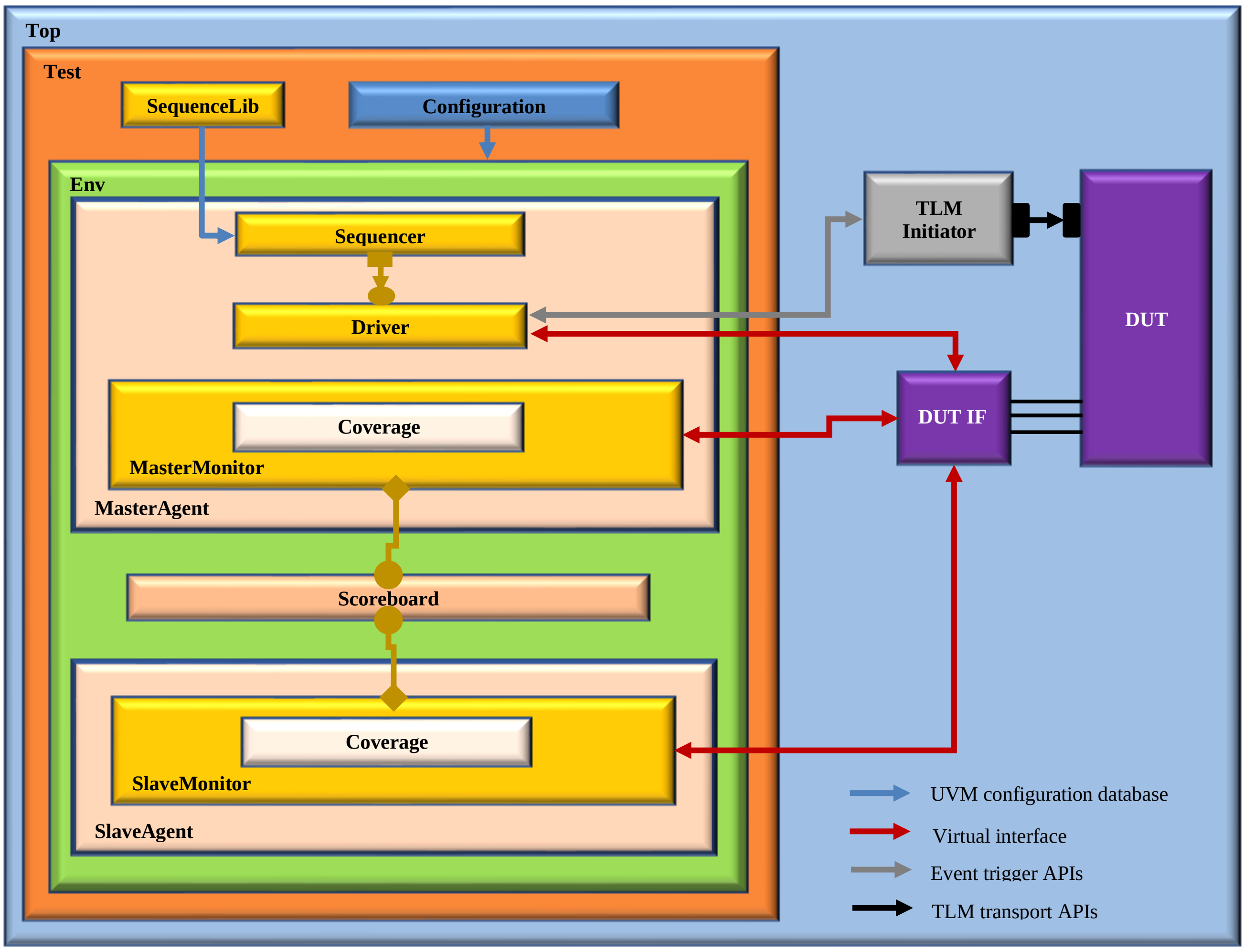
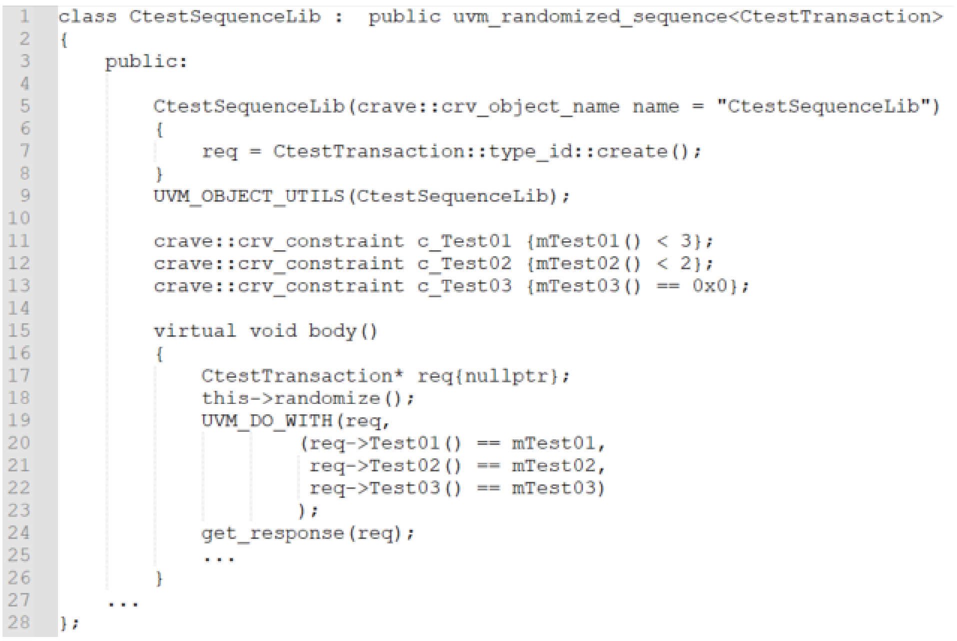


Figure 4. Block diagram of the verification environment

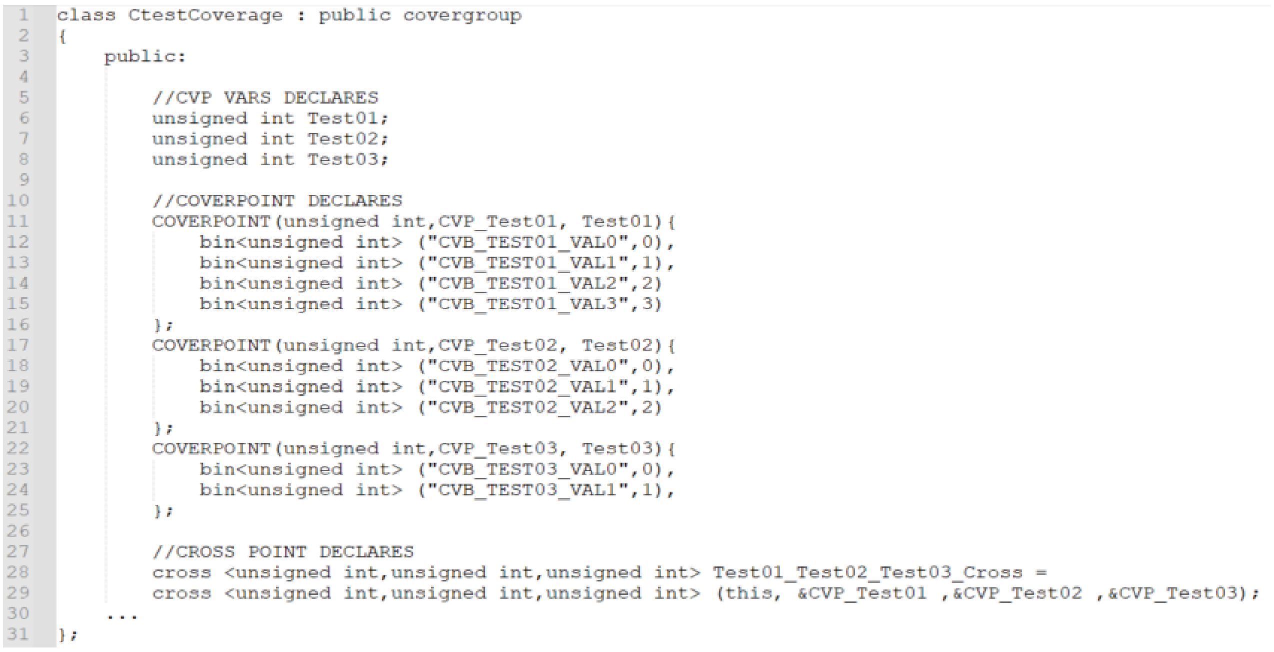
- Except for the "DUT" (Design Under Test) and "TLM Initiator" modules, others must include the UVM SystemC library to build a UVM Test Bench architecture.
- CRAVE library is included in the "SequenceLib" module which contains all tests of verifiers. It allows all tests to be simulated and randomized with constraints using randomize_with() or pre-defined macro UVM_DO_WITH(). Figure 5 shows an example of sending constrained random values.



```
class CtestSequenceLib :  public uvm_randomized_sequence<CtestTransaction>
{
    public:

        CtestSequenceLib(crave::crv_object_name name = "CtestSequenceLib")
        {
            req = CtestTransaction::type_id::create();
        }
        UVM_OBJECT_UTILS(CtestSequenceLib);

        crave::crv_constraint c_Test01 {mTest01() < 3};
        crave::crv_constraint c_Test02 {mTest02() < 2};
        crave::crv_constraint c_Test03 {mTest03() == 0x0};

        virtual void body()
        {
            CtestTransaction* req{nullptr};
            this->randomize();
            UVM_DO_WITH(req,
                    (req->Test01() == mTest01,
                     req->Test02() == mTest02,
                     req->Test03() == mTest03)
                    );
            get_response(req);
            ...
        }
    ...
};
```

Figure 5. Example of sending constrained random value

- FC4SC library is included in the "Coverage" module, the Functional Coverage Group definition (defines covergroup/coverpoint). Each time the transaction/event/signal/data is received from the "Monitors" component, it starts the sampling and records it as HTML coverage report data. Figure 6 shows an example of the Functional Coverage Group definition.



```
class CtestCoverage : public covergroup
{
    public:

        //CVP VARS DECLARES
        unsigned int Test01;
        unsigned int Test02;
        unsigned int Test03;

        //COVERPOINT DECLARES
        COVERPOINT(unsigned int,CVP_Test01, Test01){
            bin<unsigned int> ("CVB_TEST01_VAL0",0),
            bin<unsigned int> ("CVB_TEST01_VAL1",1),
            bin<unsigned int> ("CVB_TEST01_VAL2",2)
            bin<unsigned int> ("CVB_TEST01_VAL3",3)
        };
        COVERPOINT(unsigned int,CVP_Test02, Test02){
            bin<unsigned int> ("CVB_TEST02_VAL0",0),
            bin<unsigned int> ("CVB_TEST02_VAL1",1),
            bin<unsigned int> ("CVB_TEST02_VAL2",2)
        };
        COVERPOINT(unsigned int,CVP_Test03, Test03){
            bin<unsigned int> ("CVB_TEST03_VAL0",0),
            bin<unsigned int> ("CVB_TEST03_VAL1",1),
        };

        //CROSS POINT DECLARES
        cross <unsigned int,unsigned int,unsigned int> Test01_Test02_Test03_Cross =
        cross <unsigned int,unsigned int,unsigned int> (this, &CVP_Test01 ,&CVP_Test02 ,&CVP_Test03);
    ...
};
```

Figure 6. Example of Functional Coverage Group definition

C. *Build options*

Table I shows the build options which were used.

Table I. Build options

Include paths	`INCDIRS = -I$(SYSTEMC)/include \`
	`-I$(UVM_SYSTEMC_HOME)/include \`
	`-I$(CRAVE_HOME)/build/root/include \`
	`-I$(CRAVE_HOME)/metaSMT/src \`
	`-I$(CRAVE_HOME)/deps/cudd-3.0.0/include \`
	`-I$(CRAVE_BOOST_ROOT)/include \`
	`-I$(FC4SC_INCLUDE_DIR) \`
	`-I$(FC4SC_INCLUDE_DIR)/fc4sc_headers`

Library paths	`LIBDIRS = -L$(SYSTEMC)/ lib-linux64 \`
	`-L$(UVM_SYSTEMC_HOME)/lib-linux64 \`
	`-L$(CRAVE_HOME)/build/root/lib \`
	`-L$(CRAVE_HOME)/deps/cudd-3.0.0/lib \`
	`-L$(CRAVE_BOOST_ROOT)/lib`

Library dependencies	`LIBS = -lsystemc \`
	`-luvm-systemc \`
	`-lcrave -lmetaSMT \`
	`-lCUDD_obj -lCUDD_cudd -lCUDD_dddmp -lCUDD_epd -lCUDD_mtr -lCUDD_st -lCUDD_util \`
	`-lboost_filesystem -lboost_system \`
	`-lm -ldl -lutil -lpthread`

D. *Tool Version*

Table II shows versions of tools that were used.

Table II. Version of tools

Tool		Version	Remark
Compiler	GNU/ GCC	4.9.3	-
Library	Accellera/ SystemC	2.3.1a	-
	Accellera/ UVM-SystemC	1.0-beta3	Universal Verification Methodology for SystemC
	CRAVE	2018-06-14	Constrained Random Verification Environment
	FC4SC	2.1.1	Functional Coverage for SystemC

IV. RESULT

Coverage-driven random verification using UVM was applied to a verified SystemC IP. This IP was verified using the directed testing which originally took 18 man-months and found 127 bugs. Figure 7 shows a comparison between Directed testing and Coverage-driven random verification.

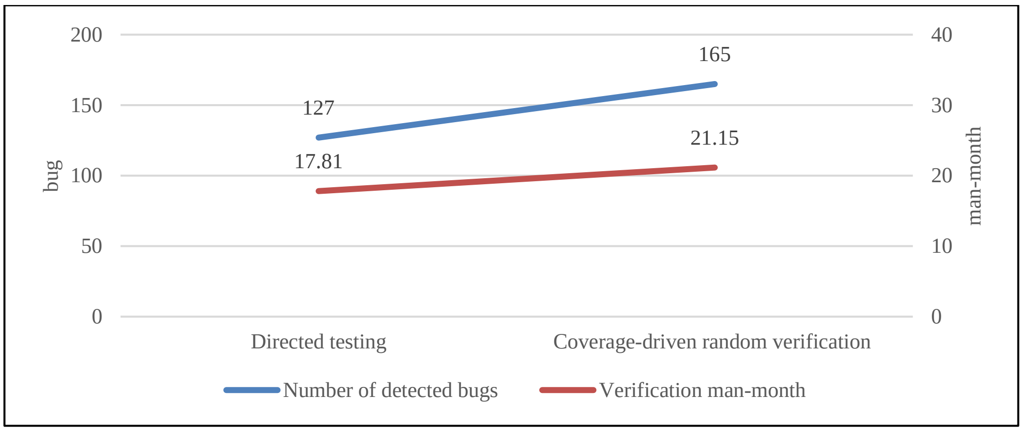


Figure 7. Comparison between Directed testing and Coverage-driven random verification

- The number of detected bugs after applying coverage-driven random verification has increased by 38 more bugs, which is 30% improved compared with directed testing. It proves the efficiency in quality improvement. Moreover, after reviewing new bugs, we concluded that 50% of them would be hard to detect in directed testing.
- Unfortunately, the verification man-month has increased by 18% compared with directed testing. Since the coverage-driven random verification covered a larger verification space than directed testing, it is reasonable that the man-month would increase. In general, it is good that only an 18% man-month increase for 30% quality improvement.

V. Conclusion

In this paper, we have introduced the efficiency of coverage-driven random verification using UVM technologies. It helps to improve the quality of SystemC IPs and solves the limitation of directed testing:

- The tests can be generated automatically, verification coverage target can be fully reached as defined schedule with less effort than directed testing.
- Verification quality can be ensured by coverage of IP specifications.
- Verification goals and progress can be ensured by functional coverage.
- Requirement and schedule can be ensured by a verification plan.

There is an increase in verification engineering resources, but it could be further saved by reusing the verification environment. The coverage-driven random verification using UVM technologies has still a lot of growth potential, we would like to share a proposal for the future:

- Create guidelines for UVM-based coverage-driven random verification.
- Develop SystemC VIPs to improve implementation efficiency and reusability.
- Utilize UVM-ML, which is UVM for multiple languages such as SystemVerilog and SystemC, to incorporate verification technologies from other domains (available VIPs from 3rd party vendors).

Acknowledgment

We would like to take this opportunity to express our gratitude to all our colleagues at Renesas Electronics Corporation. The research on coverage-driven random verification using UVM technologies would not have been successful without their cooperation and input. Moreover, we would like to give thank our managers for reviewing and supporting us in this activity.

References

[1] Juinn-Dar Huang, Ph.D. Assistant Professor, IP Core Design – Lecture 7 Introduction to SystemC, http://twins.ee.nctu.edu.tw/courses/ip_core_04/handout_pdf/07_Introduction_to_SystemC.pdf, National Chiao Tung University, September 2004.

[2] Martin Barnasconi, François Pêcheux, Thilo Vörtler, Advancing system-level verification using UVM in SystemC, https://dvcon-proceedings.org/wp-content/uploads/advancing-system-level-verification-using-uvm-in-systemc.pdf, The Design & Verification Conference & Exhibition United States (DVCon US), 2014.

[3] Doulos, Coverage-Driven Verification Methodology, https://www.doulos.com/knowhow/systemverilog/uvm/easier-uvm/easier-uvm-deeper-explanations/coverage-driven-verification-methodology/, accessed June 2023.

[4] ChipVerify's Blog, UVM introduction, https://www.chipverify.com/uvm/uvm-introduction, accessed June 2023.

[5] Finn Haedicke, Hoang M. Le, Daniel Große, Rolf Drechsler, CRAVE: An Advanced Constrained Random Verification Environment for SystemC, International Symposium on System on Chip (SoC), October 2012.

[6] Dragoș Dospinescu, Teodor Vasilache, Functional Coverage For SystemC (FC4SC), Functional Coverage for SystemC_FC4SC_SCED_EU_2018.pdf (amiq.com), SystemC Evolution Day, October 2018.

[7] ChipVerify's Blog, SystemVerilog Covergroup and Coverpoint, https://www.chipverify.com/systemverilog/systemverilog-covergroup-coverpoint, accessed June 2023.

[8] vlsi4freshers, Basics Of UVM:Testbench Architecture, https://www.vlsi4freshers.com/2020/04/uvm-testbench-architecture.html, accessed June 2023.

[9] Pao-Ann Hsiung, Transaction-Level Modeling in SystemC, https://www.cs.ccu.edu.tw/~pahsiung/courses/soc/notes/SystemC_TLM.pdf, National Chung Cheng University, March 2005.