A UVM Based Testbench Research for ABCStar

Li-Bo Cheng, Francis Anghinolfi, Ke Wang, *Member, IEEE,* Hong-Bo Zhu, Wei-Guo Lu, Zhen-An Liu, *Senior Member, IEEE*

***Abstract*—ABCStar is a front-end readout ASIC under development and aims to read out the silicon strip sensor for ATLAS Phase II upgrade. Its digital part will include two levels of buffering, several trigger modes, cluster identification, data formatting, and etc. To simulate and verify its full functions, we have built a well-constructed testbench with the Universal Verification Methodology (UVM), which represents an advance and powerful verification methodology based on SystemVerilog. Features of the testbench include: functional coverage evaluation, result comparison with reference models, and selected SystemVerilog assertions for validating key design features.**

***Index Terms*—ABCStar, ASIC, Assertion, Functional Coverage, Testbench, UVM, Verification.**

I. Introduction

THE Large Hadron Collider (LHC) in CERN, as the largest collider around the world, is designed for particle physics research. After the successful discovery of higgs boson in 2012 [1][2], the collider is planning a series of upgrade projects in upgrading its energy and luminosity, to uncover new physics in high energy physics.

In Phase-II upgrade, the LHC will run at the center-of-mass energy of 14 TeV, with an integrated luminosity upgraded to 3000 fb^{-1}. To precisely detect particles from this unprecedented high energy and luminosity collider, the inner tracker detector of ATLAS, one of the large detector in LHC, is planing being completely rebuilt with an all-new all-silicon detector, and includes inner pixel detector and outer strip detector. The outer strip detector will contain more than 190 m^2 of n-in-p planar silicon-strip sensors, including a barrel and two endcaps. Each sensor has a size of $97.54\times97.54\ \mathrm{mm}^2$, and 1280 readout strips in total[3].

ABCStar is an under developed ASIC for the strip detector front-end readout, each of which will processing 256 readout strips, with analog part and digital part. The analog part would receive 256 channels of inputs from detector, each channel with a preamplifier, a shaper and a discrimination comparator. And the digital part would get the 256-channels discrimination result (256-channels hits), controlling output these strip hit data by several possible trigger modes, finding all clusters in them, and output these clusters with packets.

Manuscript received May 30, 2016. This work was supported in part by National Natural Science Foundation of China (11435013).

Li-Bo Cheng is with the State Key Laboratory of Particle Detection and Electronic, IHEP, UCAS, Beijing 100049, P.R.China.

Francis Anghinolfi is with CERN, Geneve CH-1211, Switzerland.

Ke Wang, Hong-Bo Zhu, Wei-Guo Lu, are with the State Key Laboratory of Particle Detection and Electronic, IHEP, Beijing 100049, P.R.China.

Zhen-An Liu is with the State Key Laboratory of Particle Detection and Electronic, IHEP, Beijing 100049, P.R.China (telephone: +86-010-88236718, e-mail:liuza@ihep.ac.cn).

Universal Verification Methodology (UVM), as the first standardized verification methodology in 2011, is very powerful in digital ASIC design verification based on SystemVerilog, and split traditional verification testbench with several hierarchical functional units, for extensive and reusable. Main functional components of a typical UVM testbench include a sequence path for producing and driving constraint randomized stimulus to Design Under Test (DUT), a coverage module for collecting specific functional coverage to see whether a certain case has existed or not, and a reference module with the same function as DUT for comparing in a scoreboard module.

In addition, UVM provides a register model, which can track the register content of a DUT by a convenience layer accessing register and memory within the DUT [4][5].

Furthermore, we bring SystemVerilog Assertion(SVA) to the testbench. A SVA is simply a check against the specification of design to make sure never violates in realtime. There are three kinds of assertions in SVA:[6]

- Immediate Assertion: Non-temporal domain assertions that are executed like statements in a procedural block;
- Concurrent Assertion: Temporal domain assertions that allow creation of complex sequences using clock based semantics;
- Deferred Assertion: A special type of immediate assertions with a time stamp.

In this paper we present a UVM based well-constructed testbench for ABCStar, including the common structure of UVM testbench, and some concurrent assertions for checking key features of Input Register of ABCStar, such as mask function, and edge detection function.

II. Digital part design of Current ABCStar

The full ABCStar design simplified block diagram is showed in Fig. 1. The digital part blocks include: Mask and Edge Detection input register, L0Buffer, EvtBuffer, Cluster Finder, Readout logic, Top Logic, and Command Decoder.[7]

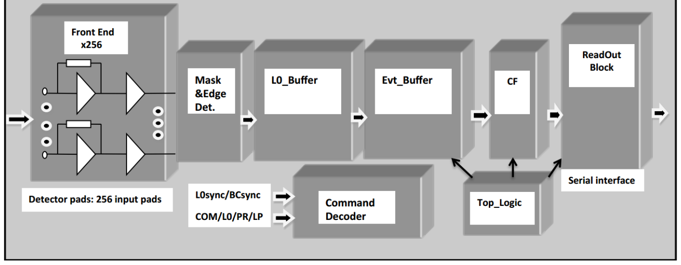


Fig. 1. Block diagram of the ABCStar.

In Input Register, it has four edge detection modes, showed in TABLE. I. Also it supports enabling any bad or noisy channels to be turned off through a mask register:

$$InReg\_out[255:0] = stripdata[255:0] \ \& \ maskbits[255:0]$$

TABLE I
EDGE DETECTION CRITERIA

mode(1:0)	Selection criteria	Hit pattern	Usage
00	Hit	1XX or X1X or XX1	Detector alignment
01	Level	X1X	Normal Data Taking
10	Edge	01X	Normal Data Taking
11	Clear	None	Special Mode

There are three trigger data flow controls in present design: L0, PR, and LP. Distribution of rate and latency of these triggers are showed in Fig. 2. At every occurrence of L0, one event of the L0Buffer is written into EvtBuffer. At every occurrence of PR or LP, one event is read out from EvtBuffer and send to the Cluster Finder block.

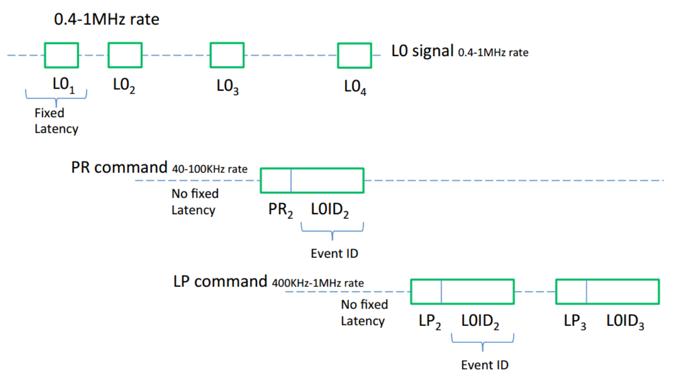
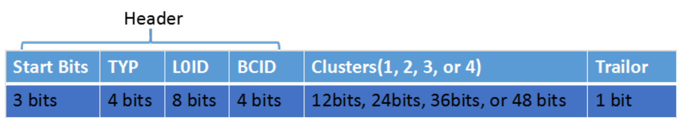


Fig. 2. L0, LP and PR distribution.

The Cluster Finder would take in 256 bits of strip data and report out 12-bit clusters at 40 MHz. All clusters would be read out by packets through readout block. Each packet is transmitted to the fast 160 Mb/s serializer. Packet format is showed in Fig. 3.



Start Bits	TYP	LOID	BCID	Clusters(1, 2, 3, or 4)	Trailor
3 bits	4 bits	8 bits	4 bits	12bits, 24bits, 36bits, or 48 bits	1 bit

Fig. 3. Packet format of the ABCStar

In ABCStar, chip reset and internal registers configuration are controlled by commands. The ports of current ABCStar digital part design are showed in TABLE. II.

III. TESTBENCH DESCRIPTION

In Fig. 4 we show the diagram of the UVM based testbench for ABCStar. There are four main features of the testbench, i.e., five sequence paths, reference model, functional coverage, and assertions.

TABLE II
PORTS LIST OF CURRENT ABCSTAR

Port name	Direction	Describe
RCLK	input	160MHz Clock input primarily intended for Data Readout
BC	input	Beam Crossing Clock at 40MHz
RSTB	input	Asynchronous hard reset
powerUpRstb	input	Reset from Power Up circuit
abcup	input	Serial input reset
chipID	input	Chip identification
L0_CMD	input	L0 Synchronous Trigger with BC falling edge, CMD(command) with BC rising edge
LP_PR	input	LP with BC rising edge, PR with falling edge CMD(command) with BC rising edge
DataOut	output	Serialized packet readout

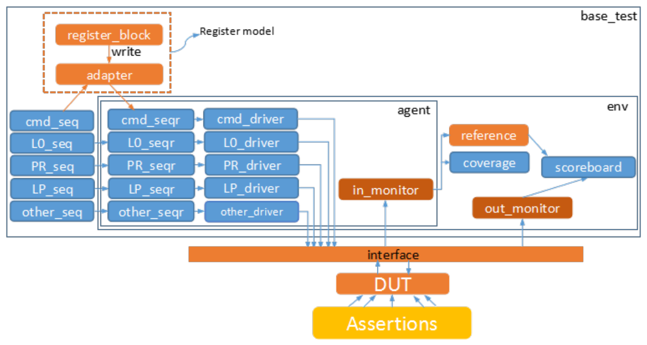


Fig. 4. Diagram of ABCStar testbench based on UVM

A. Five Sequence Paths

Since three triggers, command and stripdata input are all independent in timing, we built five independent sequence paths for them. Each path would send corresponding constraint randomized stimulus to the DUT according to tests. The UVM stimulus generation process is based on sequences controlling the behavior of drivers by generating sequence item and sending them to the driver via a sequencer.

Command sequence path is a special one, which includes a register model. Register model in UVM is proposed for simulating the behavior of configuring internal registers in DUT, including writing and reading. We build a register model including all commands of ABCStar in this testbench.

B. Reference Model

For the reference, it should have the same function with DUT. Since the function of Input Register, L0Buffer, Evt-Buffer, and Command Decoder, are very complex in timing, for their unfixed latency and data integration between different sub-modules. However, function of Cluster Finder and Readout Circuit, are complex in data logical processing, and timing is relatively simple. We built the reference with a mixture C and SystemVerilog languages. C is priviledge in data operation, and we use it for modeling the function of Cluster Finder and Readout circuit. SystemVerilog is extended from verilog, and has some powerful system functions related to timing, so we use it for modeling the function of other blocks of ABCStar with complex timing.

C. Functional Coverage

In coverage module, we built three covergroups for maskbits&stripdata, Input Register output data, and EvtBuffer output data respectively. Each covergroup would collect all 256 channels hit. TABLE. III list all covergroups and their descriptions.

TABLE III
COVERGROUP LIST OF THE TESTBENCH

Covergroup name	Number of bins	Describe
maskbits_cov	256	Statistic for all 256 channels maskbits&stripdata hits
InReg_out_cov	256	Statistic for all 256 channels InReg_out hits
buf_out_cov	256	Statistic for all 256 channels EvtBuffer output data hits

D. Assertions

We also built some assertions for checking key features of ABCStar. In TABLE. IV we list all assertions. Including 8 concurrent assertions and 1 immediate assertion. Concurrent assertions are used for assert complex sequences using clock(sampling) edgebased semantics in temporal domain. The immediate assertion is purely combinatorial, and used in scoreboard comparing two packet results, which one from the reference, and another from the DUT output.

TABLE IV
ASSERTION LIST OF THE TESTBENCH

Assertion name	Useage	type
pip_ThreeBC_lev_ass	For asserting InReg edge detection mode(level)	concurrent
pip_ThreeBC_hit_ass	For asserting InReg edge detection mode(hit)	concurrent
pip_ThreeBC_edge_ass	For asserting InReg edge detection mode(edge)	concurrent
pip_datatakingmod_ass	For asserting InReg working mode(normal data taking)	concurrent
pip_tstprinBCIDmod_ass	For asserting InReg working mode(test printing BCID)	concurrent
pip_loadmaskbitsmod_ass	For asserting InReg working mode(test loading maskbits)	concurrent
pip_pulsetestmod_ass	For asserting InReg working mode(pulse test)	concurrent
L0_Pipeline_ass	For asserting L0 controlling data output from L0Buffer	concurrent
packet_compare	For the scoreboard packets comparing	immediate

IV. Test Results

Based on the testbench we built, we did several tests with random constraint conditions to check the main functions of ABCStar, These are three typical tests, edge detection models test, maskbits test, and trigger combination test.

A. Edge Detection models test

There are three used edge detection models in Input Register of ABCStar, and one kept for special usage. The test includes:

- Limiting the hit channels number of stripdata input less than 6, and transform three modes with each other after each with a long running time.
- Checking the output of the input register with three concurrent assertions respectively for three edge detection model.
- Collecting hit coverage message of every output data channel.

The statistic for the coverage is showed in Fig. 5. It is obvious that all channels are covered (hit at least once). In Fig. 6 we present the results of assertions, which shows that all three assertions are finished without any failure count.

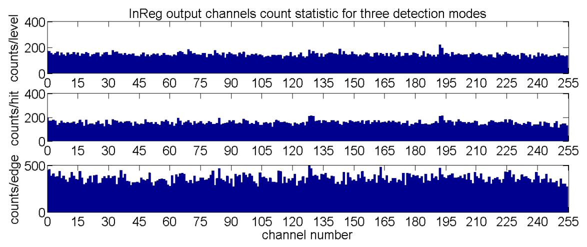
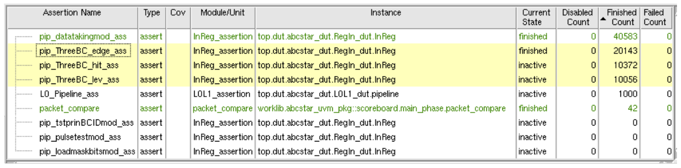


Fig. 5. InReg outputdata channels hits statistic for edge detection modes test.



Assertion Name	Type	Cov	Module/Unit	Instance	Current State	Disabled Count	Finished Count	Failed Count
pip_datatakingmod_ass	assert		InReg_assertion	top.dut.abcstar_dut.RegIn_dut.InReg	finished	0	40583	0
pip_ThreeBC_edge_ass	assert		InReg_assertion	top.dut.abcstar_dut.RegIn_dut.InReg	finished	0	20143	0
pip_ThreeBC_hit_ass	assert		InReg_assertion	top.dut.abcstar_dut.RegIn_dut.InReg	inactive	0	10372	0
pip_ThreeBC_lev_ass	assert		InReg_assertion	top.dut.abcstar_dut.RegIn_dut.InReg	inactive	0	10056	0
L0_Pipeline_ass	assert		L0L1_assertion	top.dut.abcstar_dut.L0L1_dut.pipeline	inactive	0	1000	0
packet_compare	assert		packet_compare	worklib.abcstar_uvm_pkg::scoreboard.main_phase.packet_compare	finished	0	42	0
pip_tstprinBCIDmod_ass	assert		InReg_assertion	top.dut.abcstar_dut.RegIn_dut.InReg	inactive	0	0	0
pip_pulsetestmod_ass	assert		InReg_assertion	top.dut.abcstar_dut.RegIn_dut.InReg	inactive	0	0	0
pip_loadmaskbitsmod_ass	assert		InReg_assertion	top.dut.abcstar_dut.RegIn_dut.InReg	inactive	0	0	0

Fig. 6. Assertions result of edge detection modes test.

B. Maskbits test

Input Register also support a choice to turn off any bad or noisy channels through a mask register. To verify this function, we set:

- Limiting the random hit channels number of stripdata input less than 6, and change maskbits value with the command after each running a long time.
- Checking the output of the input register with a concurrent assertion.
- Collecting hit coverage message of every maskbits&stripdata result channel.

The statistic for the coverage is showed in Fig. 7. From the figure, we can see that all channels are covered (hit at least once). Fig. 8 is the results of assertions, which shows that all assertions are finished without any failure count.

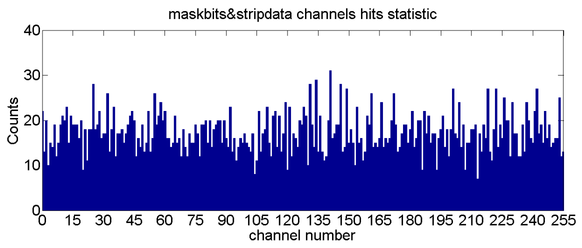
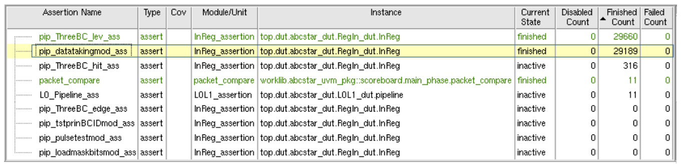


Fig. 7. Block diagram of the ABCStar.



Assertion Name	Type	Cov	Module/Unit	Instance	Current State	Disabled Count	Finished Count	Failed Count
pip_ThreeBC_lev_ass	assert		InReg_assertion	top.dut.abcstar_dut.RegIn_dut.InReg	finished	0	29660	0
pip_datatakingmod_ass	assert		InReg_assertion	top.dut.abcstar_dut.RegIn_dut.InReg	finished	0	29189	0
pip_ThreeBC_hit_ass	assert		InReg_assertion	top.dut.abcstar_dut.RegIn_dut.InReg	inactive	0	316	0
packet_compare	assert		packet_compare	worklib.abcstar_uvm_pkg::scoreboard.main_phase.packet_compare	finished	0	11	0
L0_Pipeline_ass	assert		L0L1_assertion	top.dut.abcstar_dut.L0L1_dut.pipeline	inactive	0	11	0
pip_ThreeBC_edge_ass	assert		InReg_assertion	top.dut.abcstar_dut.RegIn_dut.InReg	inactive	0	0	0
pip_tstprinBCIDmod_ass	assert		InReg_assertion	top.dut.abcstar_dut.RegIn_dut.InReg	inactive	0	0	0
pip_pulsetestmod_ass	assert		InReg_assertion	top.dut.abcstar_dut.RegIn_dut.InReg	inactive	0	0	0
pip_loadmaskbitsmod_ass	assert		InReg_assertion	top.dut.abcstar_dut.RegIn_dut.InReg	inactive	0	0	0

Fig. 8. Assertions result of maskbits test

C. Trigger combination test

There are three trigger controls in present design, L0, LP, and RP, whose distributions are showed in Fig. 2. To simulate the distribution, we use constraint randomized stimulus to set:

- L0 with a fixed latency.
- LP with a Possion distribution latency with mean of 480 BCs (12 us).
- PR with a Possion distribution latency with mean of 480 BCs (12 us).
- Checking all packets output with an immediate assertion in scoreboard with the reference.
- Collecting hit coverage message of all Evtbuffer output data channels.

The intervals between two LOs, LPs, and PRs all subject to Possion distributions with the mean of 40 BCs (1 us, rate is 1 MHz, 1 BC = 25 ns). In Fig. 9 we present the statistic for the intervals of L0 and LP and PR, which shows that the intervals is just produced as expected.

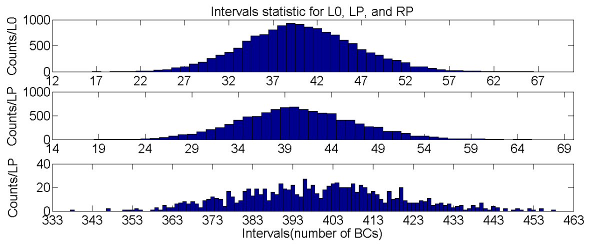


Fig. 9. maskbits&stripdata channels statistic for maskbits test.

In Fig. 10 we present the statistic for the latency of LP and PR, which show that the latency is just produced as demands.

The statistic for the coverage of Evtbuffer output is showed in Fig. 11. It is obvious that all channels are covered.

Fig. 12 is the assertion comparing result for every packet. It shows that all packet comparing finished without any failure count.

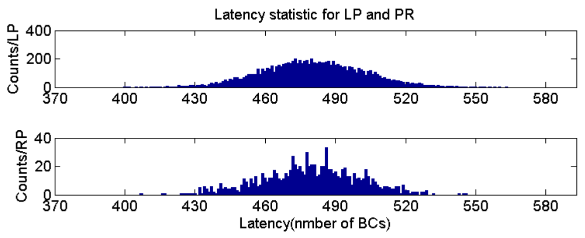


Fig. 10. LP and PR latency statistic for trigger combination test

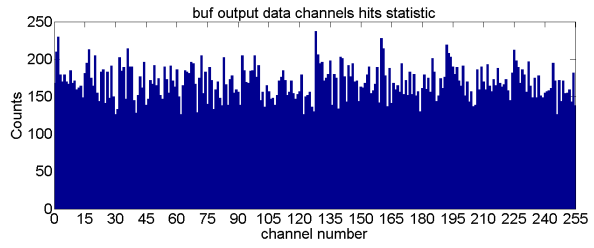
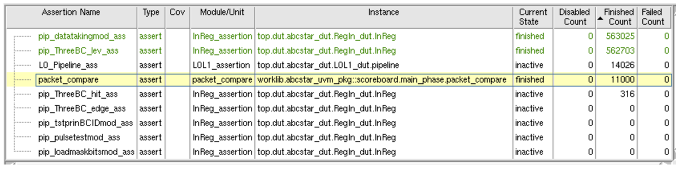


Fig. 11. Channels hits statistic for EvtBuffer output data for trigger combination test.



Assertion Name	Type	Cov	Module/Unit	Instance	Current State	Disabled Count	Finished Count	Failed Count
pip_datatakingmod_ass	assert		InReg_assertion	top.dut.abcstar_dut.RegIn_dut.InReg	finished	0	563025	0
pip_ThreeBC_lev_ass	assert		InReg_assertion	top.dut.abcstar_dut.RegIn_dut.InReg	finished	0	562703	0
L0_Pipeline_ass	assert		L0L1_assertion	top.dut.abcstar_dut.L0L1_dut.pipeline	inactive	0	14026	0
packet_compare	assert		packet_compare	worklib.abcstar_uvm_pkg::scoreboard.main_phase.packet_compare	finished	0	11000	0
pip_ThreeBC_hit_ass	assert		InReg_assertion	top.dut.abcstar_dut.RegIn_dut.InReg	inactive	0	316	0
pip_ThreeBC_edge_ass	assert		InReg_assertion	top.dut.abcstar_dut.RegIn_dut.InReg	inactive	0	0	0
pip_tstprinBCIDmod_ass	assert		InReg_assertion	top.dut.abcstar_dut.RegIn_dut.InReg	inactive	0	0	0
pip_pulsetestmod_ass	assert		InReg_assertion	top.dut.abcstar_dut.RegIn_dut.InReg	inactive	0	0	0
pip_loadmaskbitsmod_ass	assert		InReg_assertion	top.dut.abcstar_dut.RegIn_dut.InReg	inactive	0	0	0

Fig. 12. Assertions results of trigger combination test.

V. Conclusions and Discussions

We built a well-constructed UVM based testbench for ABCStar. It includes functional coverage statistic, a reference for comparing, and some assertions checking in real time. Then we did several tests based on the testbench to check the main design features of ABCStar, by using constraint randomized stimulus theory to simulate the real possible data input activity.

One of the most challenging task in our work is modeling the reference model. At first, we built a testbench only for verifying the ClusterFinder module of ABCStar, and the reference model is written by C language alone. However, when we consider verifying all functions of ABCStar, using C language alone to modele the new complicated reference seems impossible. In this case, we split the reference into two parts, written by C language and SystemVerilog respectively, with one part to model functions without complex timing and the other part to model functions with complex timing, .

Another cumbersome part is producing stimulus for all three triggers, command, and stripdata inputs. It is very difficult to produce stimulus for all these signals in a sequence path, since they all independent with each other in timing. To solve this problem, we built five sequence paths for them respectively.

Our study shows UVM is very powerful in simulating and verifying digital ASIC design. Its constraint randomized stimulus can help simulating the real possible data input activity, and its functional coverage and assertions can ensure believability of the verification.

References

[1] G. Aad *et al.* [ATLAS Collaboration], Phys. Lett. B **716**, 1 (2012) doi:10.1016/j.physletb.2012.08.020 [arXiv:1207.7214 [hep-ex]].

[2] S. Chatrchyan *et al.* [CMS Collaboration], Phys. Lett. B **716**, 30 (2012) doi:10.1016/j.physletb.2012.08.021 [arXiv:1207.7235 [hep-ex]].

[3] The ATLAS Collaboration, *ATLAS Phase-II Upgrade Scoping Document*. url:https://cds.cern.ch/record/2055248, 2015.

[4] Accellera, *UVM 1.1 User's Guide*. 2011.

[5] Ashok B. Mehta. *SystemVerilog Assertions and Functional Coverage*. Springer-Verlag New York, 2014.

[6] Mentor Graphics Verification Methodology Team. *Verification Academy Cookbook*. url:http://verificationacademy.com/uvm-ovm, 2013.

[7] *ATLAS ITK Electronics Specification Component or Facility Name: ABCstar*. (internal ATLAS note), 2015.