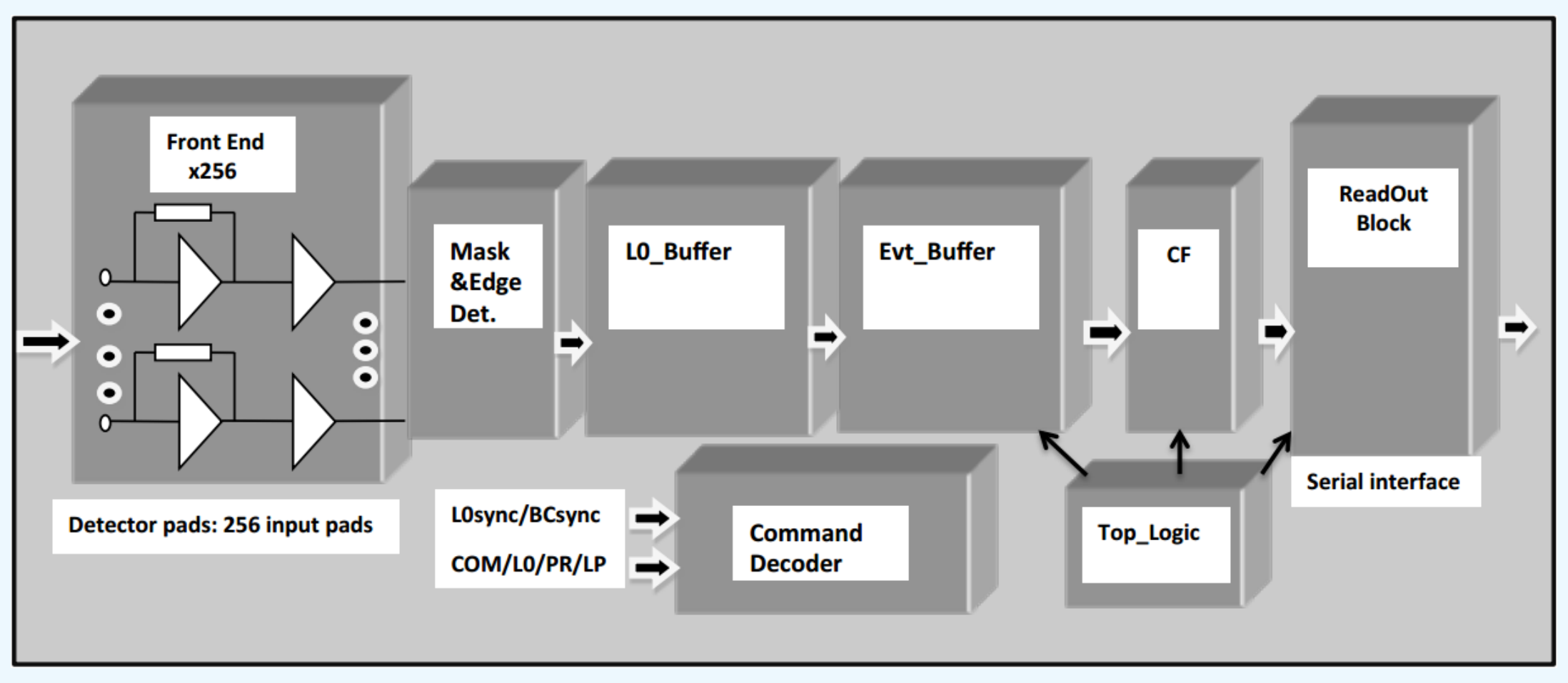
A Testbench Based on UVM Research for ABCStar

Libo Cheng[1], Francis Anghinolfi[2], Ke Wang[1], Zhen-An Liu[1], Hongbo Zhu[1], Weiguo Lu[1]
1, Institute of High Energy Physics, CAS, Beijing 100049, China
2, CERN, Geneve CH-1211, Switzerland

Abstract

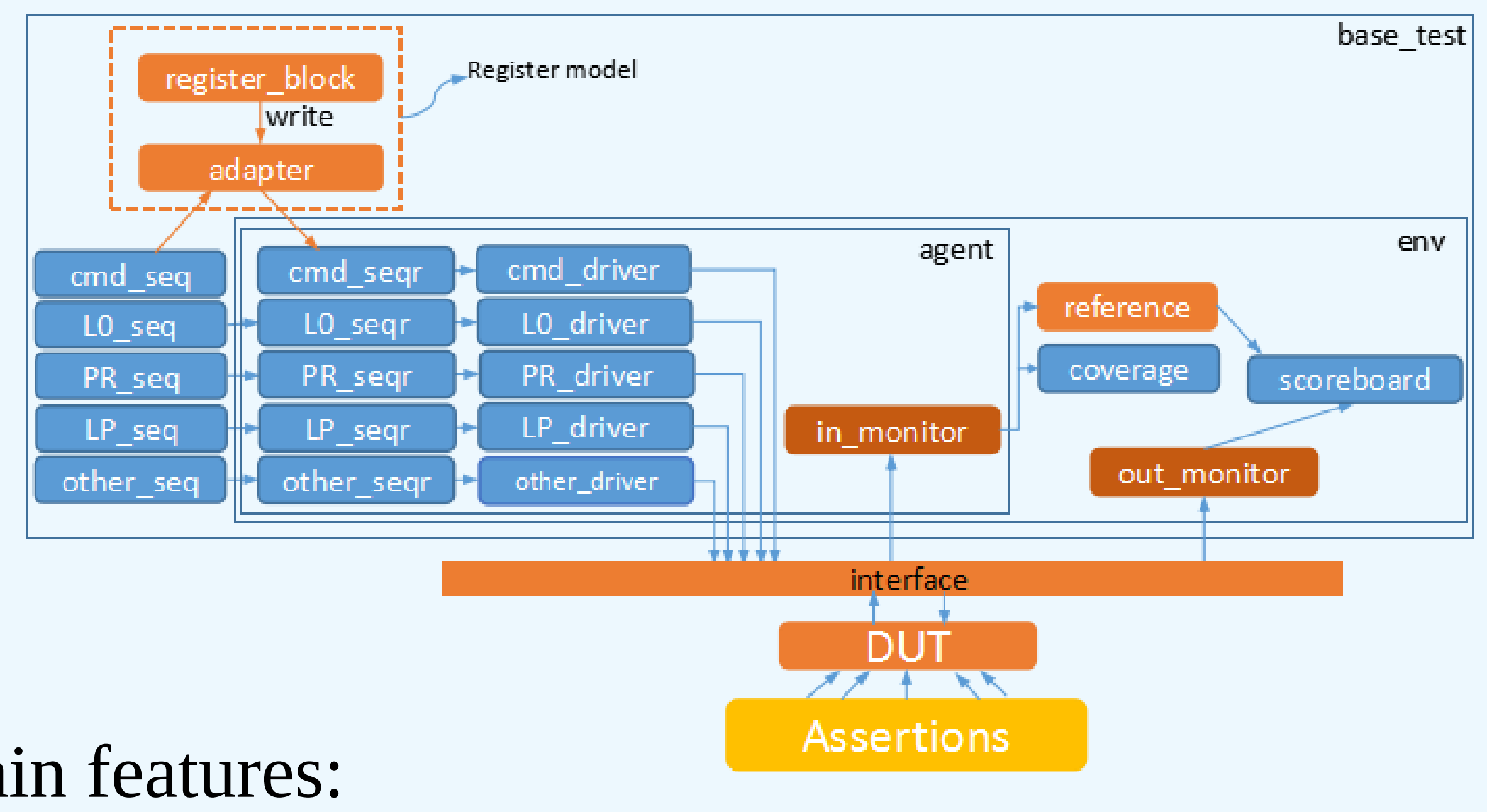
It is demanding to implement more and more functions and algorithms in front-end Application-Specific Integrated Circuit (ASIC), which, in consequence, makes the logic designs more complicated. It necessitates to develop reliable and robust function verification test benches during ASIC designs. ABCStar 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 would 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.

ABCStar Blocks



Main functions in the digital part:
- ➢ 256-channel inputs with mask and edge detection functions;
- ➢ Three trigger data flow controls: L0, PR, and LP;
- ➢ To find clusters and read out them in data packets.
- ➢ Chip reset and internal register configuration by commands.

Testbech Structure



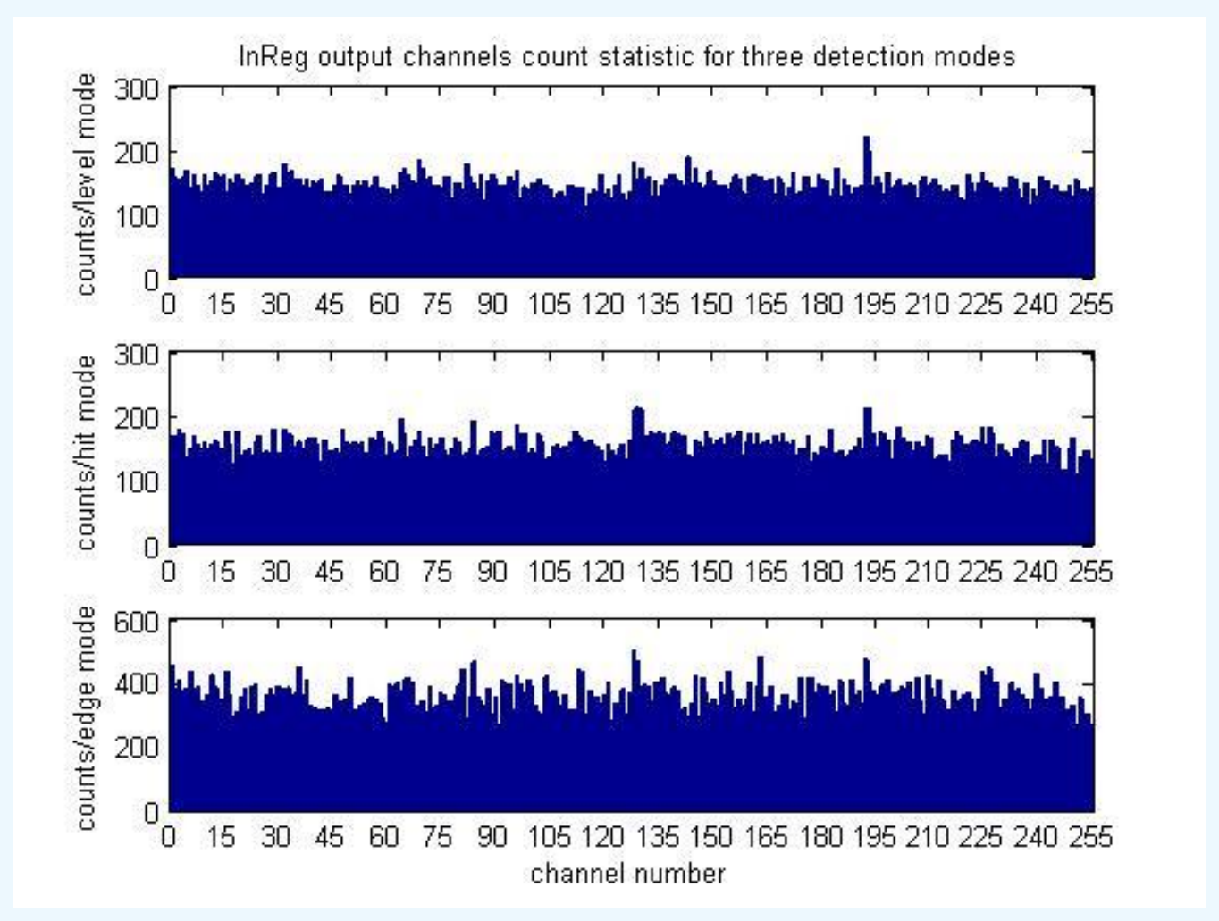
Main features:
- ➢ Five independent sequence paths to generate stimulus;
- ➢ One register model for all commands;
- ➢ One reference model built with combined C and SystemVerilog languages;
- ➢ Coverage for functional statistic, and concurrent assertions for realtime validation.

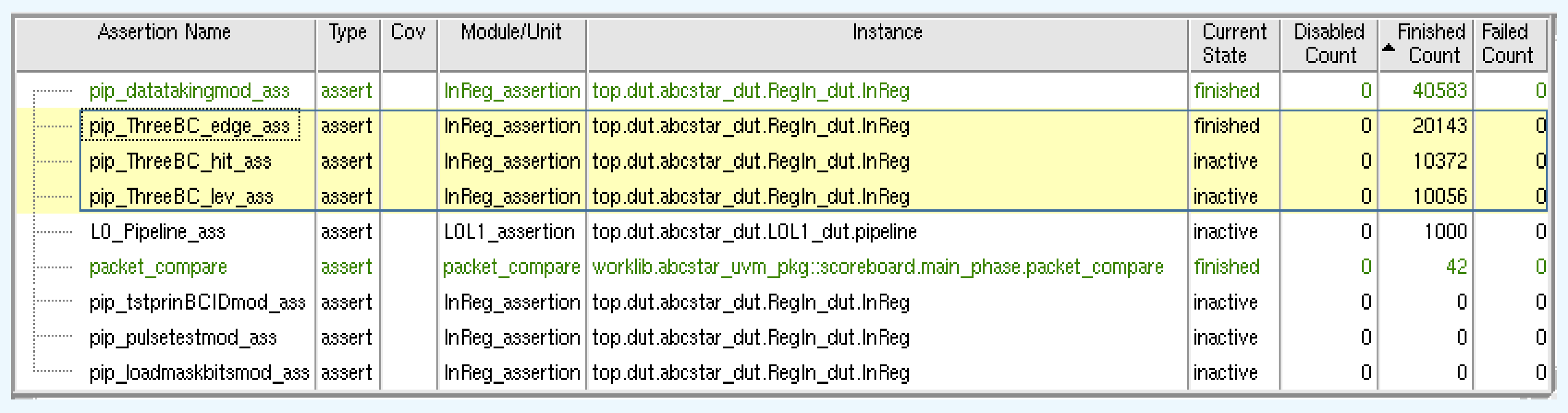
Test Results

1 Input Register detection mode test

There are four detection modes for ABCStar Input Register, two for normal data taking (Level and Edge mode), one for alignment (hit mode), and one reserved for special usage.

With corresponding concurrent assertions, we can check whether the Device Under Test (DUT) works correctly in each mode. We also sample the functional coverage output hits of the Input Register in three modes,



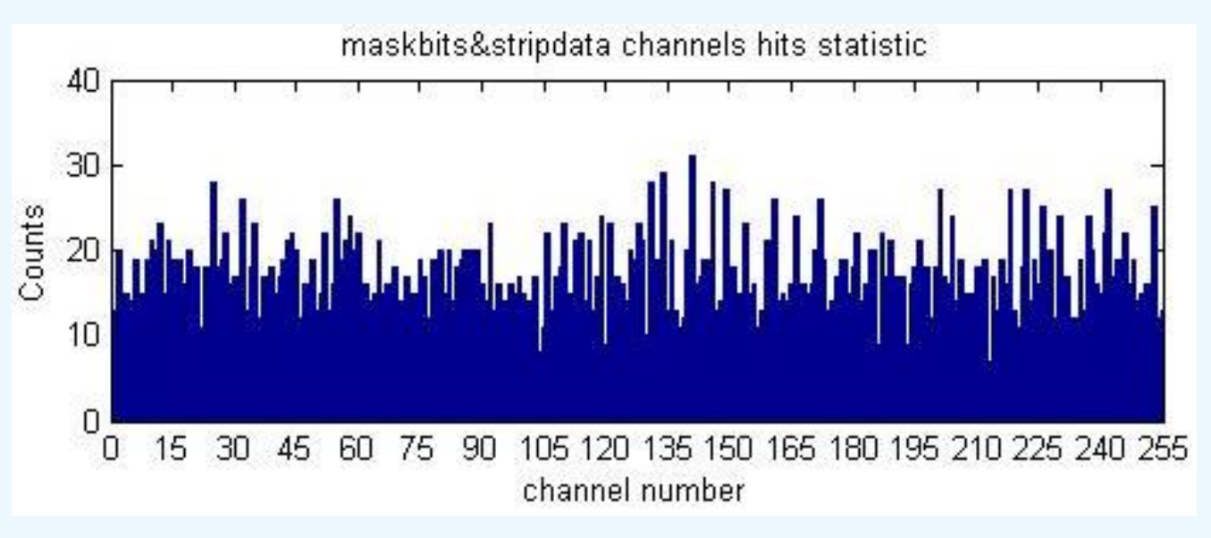


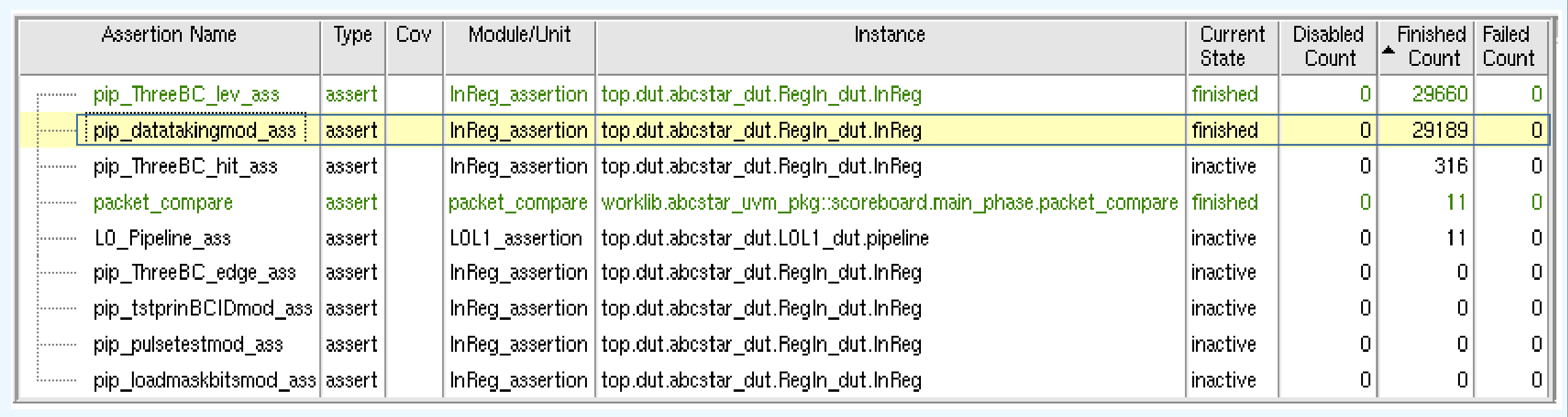
Assertion Name	Type	Cov	Module/Unit	Instance	Current State	Disabled Count	Finished Count	Failed Count
pip_datatakingmod_ass	assert		InReg_assertion	top.dut.abcstar_dut.RegIn_dut.InReg	finished	0	40583	0
pip_ThreeBC_edge_ass	assert		InReg_assertion	top.dut.abcstar_dut.RegIn_dut.InReg	finished	0	20143	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

2 Maskbits test

ABCStar supports turning off bad or noisy channels through a mask register according to the following logic:

output_data(256 bits) = input_data(256 bits) & maskbits(256 bits).





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

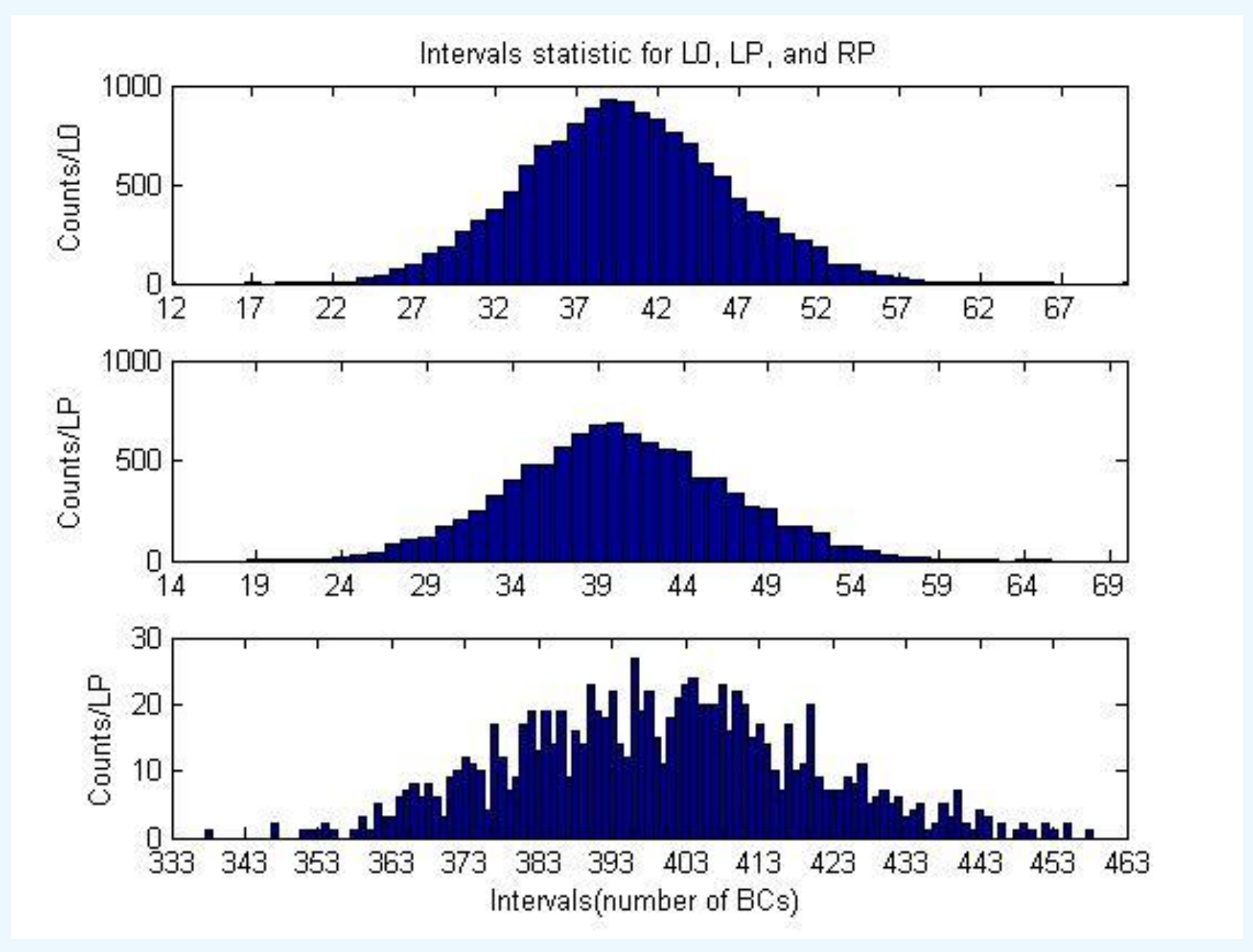
3 Trigger combination test

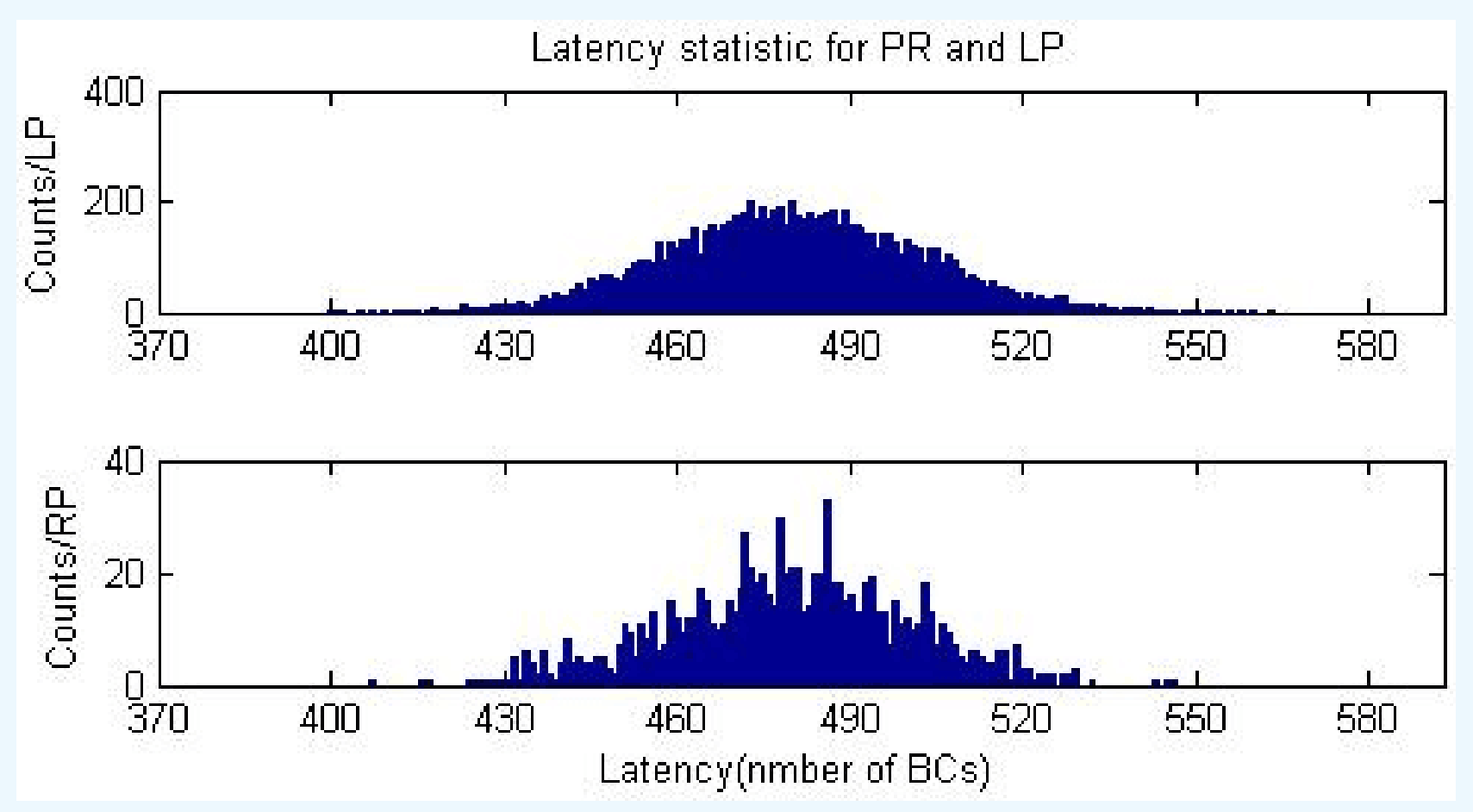
We use constraint randomized stimulus to produce:

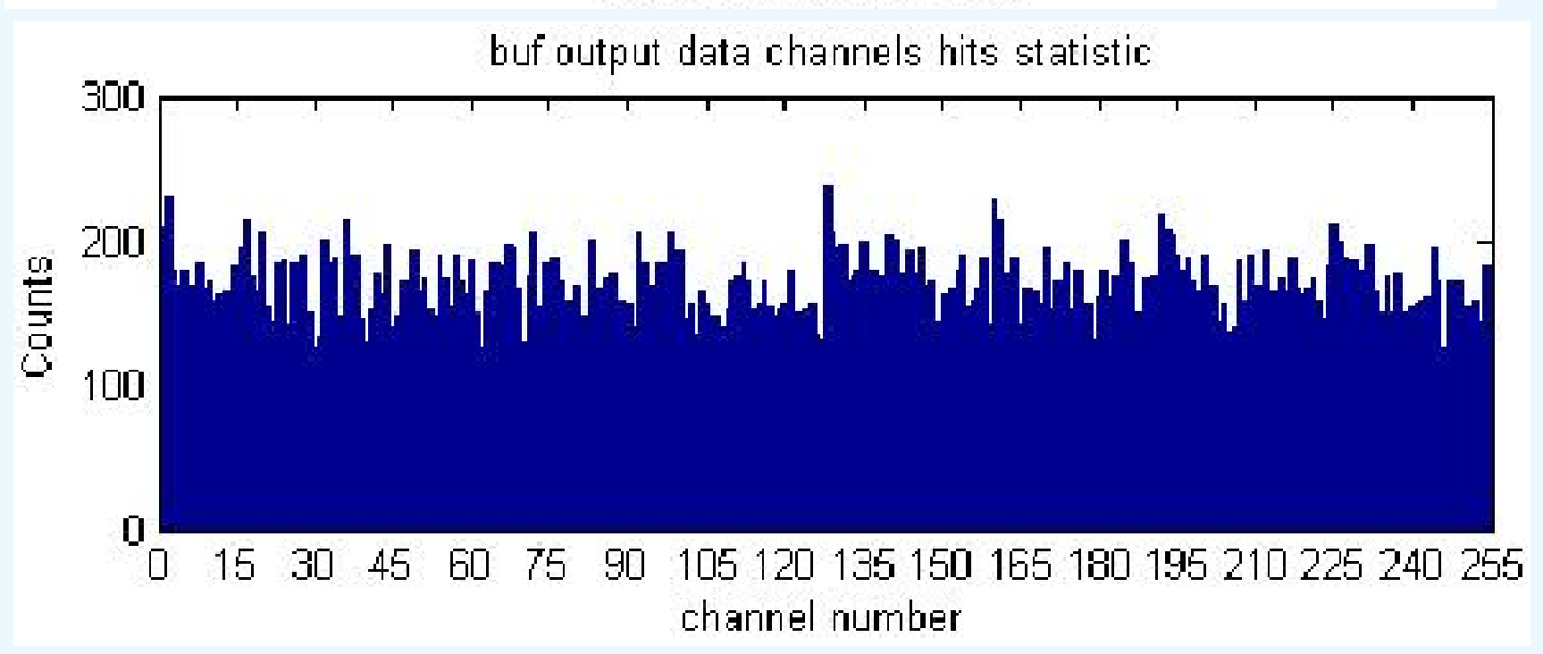
L0 with a fixed latency;

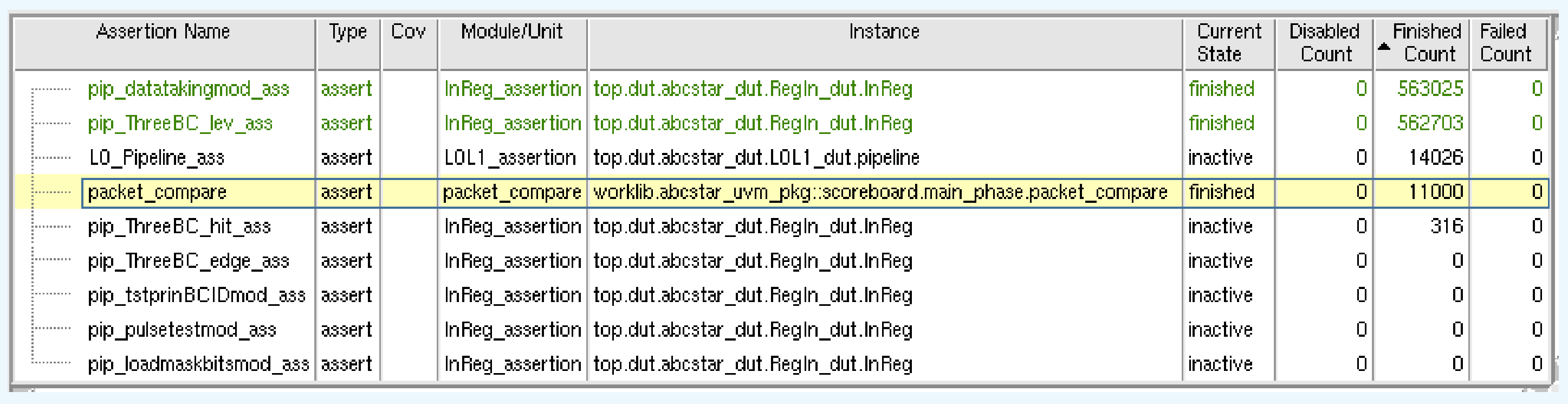
L0_intervals = $dist_possion(seed, 40)	(BCs, rate is 1MHz).
LP_latency=$dist_poisson(seed, 480)	(BCs, mean is 12us).
LP_intervals =$dist_possion(seed, 40)	(BCs, rate is 1MHz).
PR_latency=$dist_poisson(seed, 480)	(BCs, mean is 12us).
PR_intervals =$dist_possion(seed, 400)	(BCs, rate is 100KHz).

**Note: BC is Beam Crossing Clock, 1BC=25ns*









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

Conclusion

◆ We have built a well-structured testbench for the ABCStar design based on UVM. It includes functional coverage statistic, a reference for result comparison, and assertions for real time validation.

◆ We have verified the main features of the ABCStar, by using UVM-constrained and randomized stimulus to simulate the real data input. the coverage and assertions show the design is working correctly.