UVM and SVA Based Functional Verification of an SLM Wear-out Monitoring IP

Shidi Xi
Department of Electrical and Computer Engineering
The University of British Columbia
Vancouver V6T 1Z4, Canada
xsd99@ece.ubc.ca

***Abstract*—Nowadays, many SLM IPs are being integrated into SoCs, and effective functional verification of them is becoming increasingly important. Meanwhile, on-chip communication interfaces are also growing more complex. As an essential component of any SoC IP, this puts additional demands on the IP's verification process. In this report, an SLM wear-out monitoring IP with an AMBA APB interface has been verified using UVM and SVA. UVM testbenches capable of generating constrained random stimuli were developed for different levels of the design, and SVA properties were written for the APB interface. 100% functional coverage has been achieved, demonstrating the effectiveness of the methodology.**

I. Introduction

Universal Verification Methodology (UVM) is a standardized methodology for functional coverage-driven verification of digital designs, primarily in SystemVerilog [1]. It enables the creation of reusable, modular, and scalable testbenches, making it a dominant choice in the IC verification industry. SystemVerilog Assertions (SVA) is an integral part of the SystemVerilog language that provides a well-structured way of integrating assertions into testbenches written in SystemVerilog [2]. SVA can be used to check the temporal and functional properties of the design under test (DUT) and are particularly useful when verifying a communication protocol such as AMBA APB.

Silicon life management (SLM) involves techniques to monitor, predict, and extend the operational life of ICs. It is critical in industries where circuit reliability is a top concern, such as automotive, biomedical, and aerospace [3]. One aspect of SLM is wear-out monitoring, which involves using on-chip sensors to constantly measure the physical and electrical characteristics of an IC to detect signs of circuit degradation (wear-out) caused by wear-out mechanisms. Common parameters to monitor include voltage, temperature, and frequency. Common wear-out mechanisms include bias threshold instability (BTI), hot carrier injection (HCI), electromigration (EM), and stress-induced leakage current (SILC).

A wear-out monitoring IP can be used within a system-on-chip (SoC) to gather on-chip sensor readings and transfer the data back to the CPU. Usually, the IP contains logic to collect and analyze output from ADCs, which digitalize the sensors' analog output. Its communication with the CPU is typically enabled by a standard on-chip bus protocol such as AMBA APB. APB is part of the AMBA protocol family, designed to bridge low-speed and low-power peripherals to the processor.

This report presents the verification methodology and results of an SLM wear-out monitoring IP designed in-house. Using UVM and SVA, the functionality of the IP and its APB interface have been successfully verified with full functional coverage. An overview of the IP and the APB protocol is presented in Section II. Section III describes the verification strategy, followed by the UVM testbench architecture and SVA properties in Section IIII and V. Section VI presents the verification results and Section VII concludes the report.

II. DUT Overview

A. IP Functionalities

Figure 1 presents the IP's block diagram. The IP, depicted by the yellow block, consists of data processing modules on the right to gather and analyze the data from the on-chip wear-out sensors. The IP also has an APB interface to enable communication with the CPU. In addition, there are some internal registers within the IP. The CPU Commands Register takes in commands from the CPU through APB. These commands control the sensor modules to provide data back to the CPU, which will be written into the Sensor Measurement Register before being sent out on the APB interface.

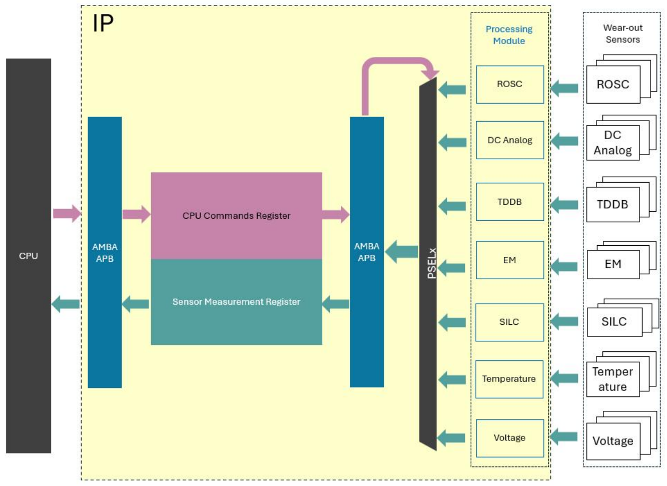


Fig. 1: Block diagram of the SLM wear-out monitoring IP [4].

B. AMBA APB Protocol

APB is a low-cost and easy-to-implement interface protocol that is commonly used to enable data transfer between the CPU and on-chip low-power peripherals [5]. In the protocol, a requester, usually the CPU, requests data from a completer, who responds to the request with data. Oftentimes, the peripherals serve the role of the completers. Both parties must have the same set of interface signals defined by the protocol. Some of the essential signals of APB are listed in Table I.

TABLE I: List of core signals of the APB interface [5].

Signal	Source	Width	Description
PCLK	Clock	1	Clock signal.
PADDR	Requester	ADDR_WIDTH	APB address bus.
PSEL	Requester	1	Select.
PENABLE	Requester	1	Enable.
PWRITE	Requester	1	Direction.
PWDATA	Requester	DATA_WIDTH	Write data.
PREADY	Completer	1	Transaction complete.
PRDATA	Completer	DATA_WIDTH	Read data.

Figure 2 illustrates an APB write transfer. At T1, the requester asserts PSEL to select a completer. PADDR, PWRITE, and PWDATA must be valid at the same time (PWRITE is high during write). Then at T2, PENABLE is asserted, signaling the start of a transaction. All the signals mentioned above must be stable until the completer asserts PREADY at T4, indicating that PWDATA will be accepted at the next cycle. Lastly, at T5, PSEL and PENABLE must be deasserted to finish the transaction.

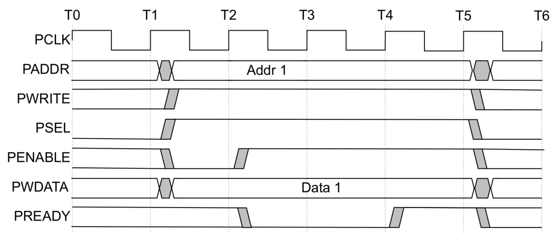


Fig. 2: APB write transfer [5].

Figure 3 illustrates an APB read transfer. The timing of PADDR, PWRITE, PSEL, and PENABLE are the same as the write operation. Notice PWRITE must be low when reading, and when the completer has the valid data, it must drive it on PRDATA and assert PREADY at the same time.

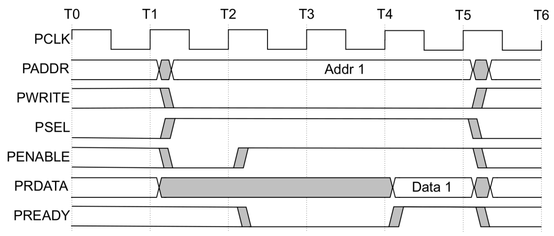


Fig. 3: APB read transfer [5].

III. Verification Strategy

The verification strategy was developed alongside the RTL design of the IP, employing a bottom-up approach to match the design's development. The IP's individual sensor processing modules were designed first, followed by their integration into the IP along with the APB interface. Consequently, the verification process consisted of three distinct stages:

- **Module level**: Verified the functionalities of individual modules as they were developed. Each module had its own UVM testbench, coverpoints, and test cases.
- **IP level**: Conducted full IP verification after integrating all components. A separate testbench was created, combining coverpoints and test cases from individual modules.
- **APB interface**: Ensured that the APB interface adhered to the protocol. SVA was employed for this purpose instead of UVM.

IV. UVM Testbench Architecture

The UVM framework consists of several components that form a structured testbench architecture. Each component, with a unique functionality, is a self-contained class implemented using object-oriented programming (OOP) in SystemVerilog, making them easily reusable across different testbenches or projects. Figure 4 shows a standard architecture of a UVM testbench containing all the common components.

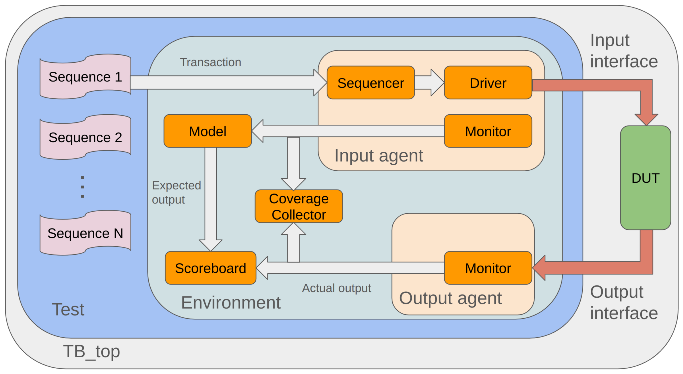


Fig. 4: UVM testbench architecture.

- **TB_top**: The top module of the testbench, not a UVM component. It instantiates a Test object, the DUT, and two interfaces positioned at the input and output, respectively.
- **Test**: Instantiates the Environment and several Sequences. It configures the Environment and controls the test flow by sequentially starting the Sequences.
- **Sequence**: Describes the desired stimuli as a series of Transactions. For example, corner case stimuli reside in one Sequence, while error case stimuli are in another. Completion of all Sequences ensures all desired stimuli are applied to the DUT and test cases are covered.
- **Environment**: A container class consisting of Agents, a Model, a Scoreboard, and a Coverage Collector.

- **Agent**: A container class that can be either active or passive. An active Agent, typically at the input, includes a Sequencer, Driver, and Monitor. A passive Agent, usually at the output, contains only a Monitor.
- **Sequencer**: Receives transactions from Sequences and schedules them for the Driver.
- **Driver**: Converts transactions from the Sequencer into pin-level activities and drives the DUT's pins via an interface.
- **Monitor**: Observes pin activities on the interface, transforms them into transaction objects, and sends them to the Model, Scoreboard, or Coverage Collector.
- **Model**: Fully emulates the DUT's behavior, generating the expected output for given inputs and sending it to the Scoreboard.
- **Scoreboard**: Compares the expected output from the Model with the actual output from the DUT.
- **Coverage Collector**: Defines coverpoints for the DUT based on its specifications and collects coverage information from the input and output Agents.

Each module in the IP, along with the IP itself, had a dedicated testbench based on the architecture described above. These testbenches differed in their Sequences due to varying functionalities and test cases.

V. APB Verification by SVA

Based on the APB protocol described in Section II.B, the following properties must hold during the operation of the IP:

- If PENABLE is asserted, it must remain high until PREADY is also asserted.
- If PENABLE and PREADY are both high in a given cycle, the transaction must be completed, and PENABLE deasserted in the next cycle.
- During a write transaction, while PENABLE is high, PADDR, PWRITE, and PWDATA must remain stable.
- During a read transaction, while PENABLE is high, PADDR and PWRITE must remain stable.

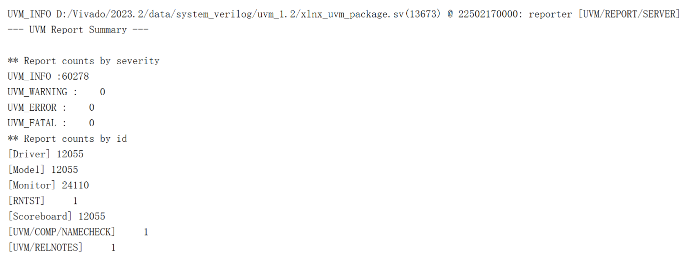
These properties were implemented using SVA and integrated into the IP's UVM testbench. While the DUT responded to stimuli generated by UVM Sequences and its operation was scrutinized by the Monitor and Scoreboard, these assertions were evaluated in parallel.

VI. Results

This section presents the simulation waveforms and the coverage report for the IP level verification. All simulations were conducted using Vivado. Figure 5 is the UVM report generated when the simulation had completed. It shows that there is no warning, error, or fatal error during the entire simulation. This indicates that the IP is functionaly correct.

A. Test Case 1: Random Normal Operation

Figure 6 presents two random transactions between the UVM testbench and the DUT. First, a UVM Sequence generated a random transaction, which was then given to the UVM Sequencer. The Sequencer scheduled the transaction to



```
UVM_INFO D:/Vivado/2023.2/data/system_verilog/uvm_1.2/xlnx_uvm_package.sv(13673) @ 22502170000: reporter [UVM/REPORT/SERVER]
--- UVM Report Summary ---

** Report counts by severity
UVM_INFO :60278
UVM_WARNING :    0
UVM_ERROR :    0
UVM_FATAL :    0
** Report counts by id
[Driver] 12055
[Model] 12055
[Monitor] 24110
[RNTST]     1
[Scoreboard] 12055
[UVM/COMP/NAMECHECK]     1
[UVM/RELNOTES]     1
```

Fig. 5: UVM simulation report.

the UVM Driver. The Driver converted the transaction to pin-activities and drove 'h90723ffc on the CPUCommand bus shown in the figure. At the same time, the UVM Model received the same transaction and produced the expected output for the UVM Scoreboard. Inside the DUT, 'h90723ffc was written into the IP's CPU Commands Register (Figure 1) via the APB interface. Then the command activated a particular sensor module (the analog sensor module in this case) which would start gathering data from its corresponding sensor. Eventually, the result, 'h01732001, was written into the Sensor Measurement Register and driven onto the ResultForCPU bus shown in the figure. This output was collected by the UVM Monitor who then sent it to the Scoreboard to compare with the expected output and to the Coverage Collector for coverage measurement. The check passed in the Scoreboard and a new transaction, marked by 'hb032fffc on the CPUCommand bus, was generated. The process repeated until the entire UVM Sequence had been completed.

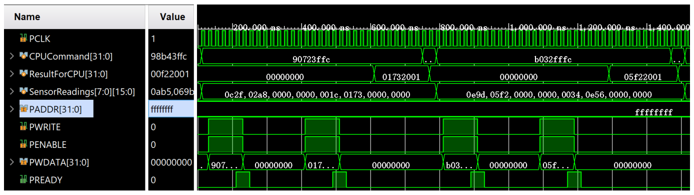


Fig. 6: Simulation waveform of random normal operation.

B. Test Case 2: Corner Case

Figure 7 shows the waveform of a corner case (0 measurement cycles for the SILC module). This is to demonstrate that UVM Sequences can be constrained to generate direct test cases to cover corner scenarios.

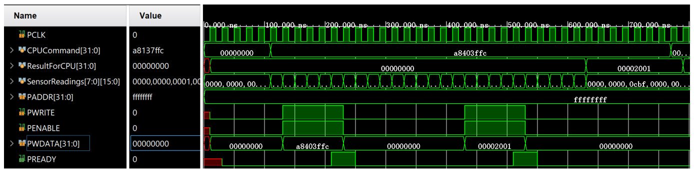


Fig. 7: Simulation waveform of a corner case.

C. Test Case 3: Error Case

Figure 8 shows the waveform of an error case (0 threshold for the SILC module). It demonstrates the DUT's behavior under erroneous inputs. The ResultForCPU bus shows 'h000e001 indicating an error has occurred.

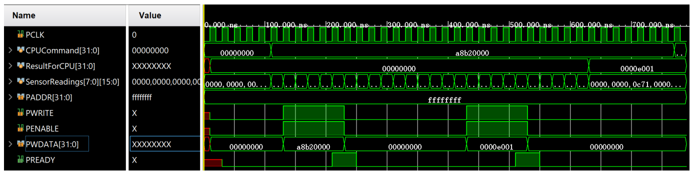


Fig. 8: Simulation waveform of an error case.

D. APB SVA

The simulation waveforms presented earlier also include APB signals. From the waveforms it can be seen that the toggling of these signals adheres to the APB protocol. All of the SVA properties defined in Section V have been successfully asserted. Figure 9 is a snippet of the console messages during simulation. It shows a successful assertion and also a transaction captured by the UVM Monitor whose check passed in the Scoreboard.

E. Functional Coverage

100% functional coverage has been achieved for both the module- and the IP-level verification. Figure 10 presents the IP's coverage report. The total coverage (left) and some of the coverpoints (right) are shown.

VII. Conclusion

An SLM wear-out monitoring IP with an AMBA APB interface has been successfully verified in this project. The proposed UVM testbenches could generate constrained random stimuli to achieve 100% functional coverage. Such finding has proven the effectiveness of using UVM to conduct functional coverage-driven verifications as well as the practicality of using SVA for protocol verification. Similar verification methodology can be used for other designs with similar architecture.

References

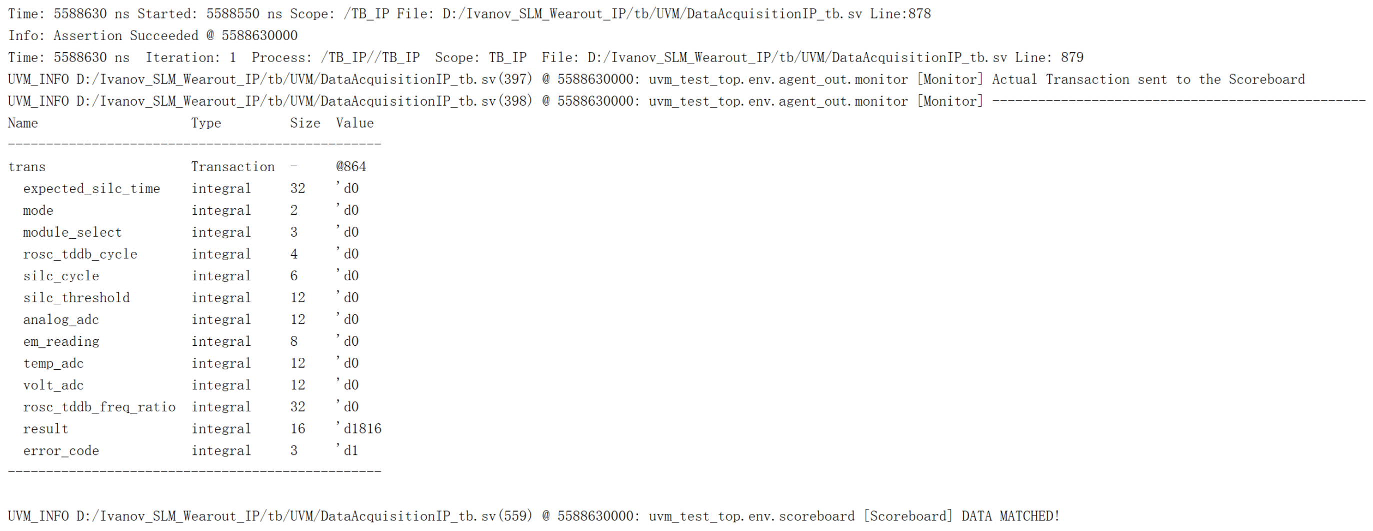
[1] "Ieee standard for universal verification methodology language reference manual," 6 2020. [Online]. Available: https://ieeexplore.ieee.org/document/9195920/

[2] "Ieee standard for systemverilog–unified hardware design, specification, and verification language," 12 2023. [Online]. Available: https://ieeexplore.ieee.org/document/10458102/

[3] I. Hill, M. Rendón, and A. Ivanov, "Enhanced wear-out sensor design in a 12nm process for separable stress regime monitoring," *Proceedings of the IEEE VLSI Test Symposium*, 2024.

[4] A. Peng, "A reliable monitor and read-out ip for on-chip wear-out sensors," 2024.

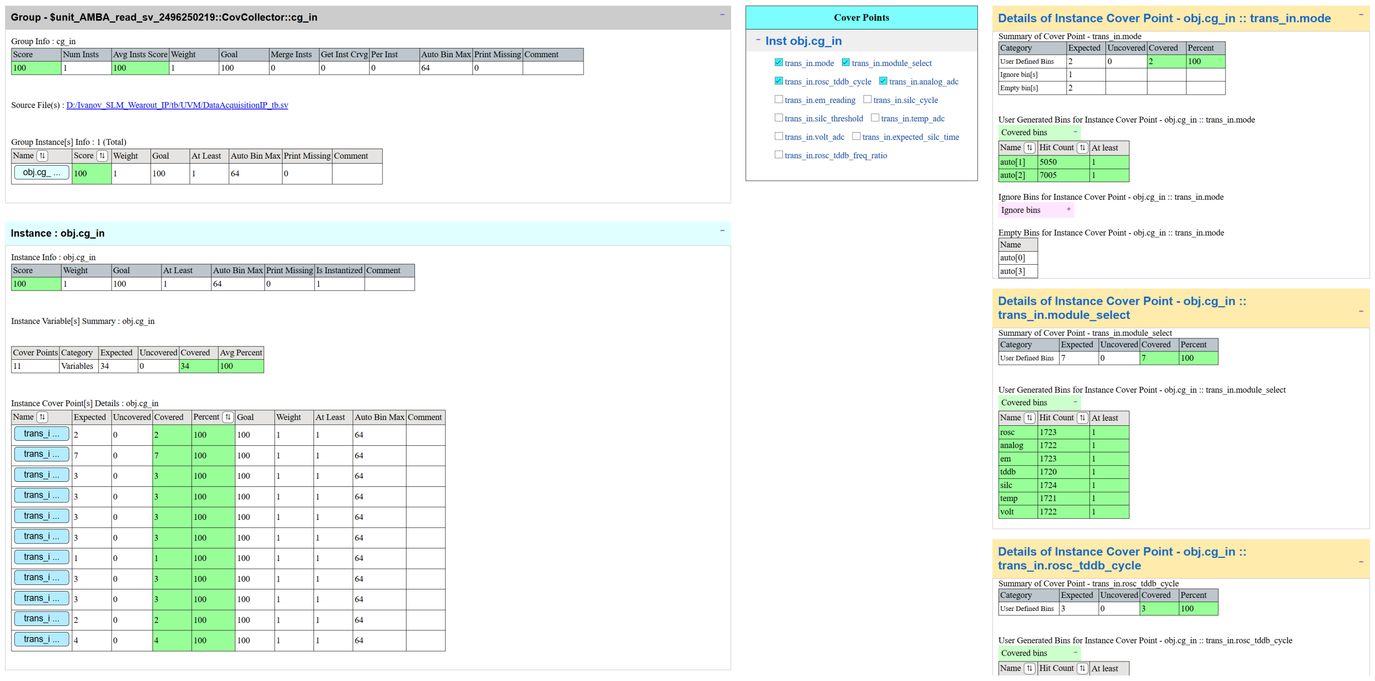
[5] Arm Ltd., "Amba ® apb protocol specification," 2023. [Online]. Available: https://developer.arm.com/documentation/ihi0024/latest/



```
Time: 5588630 ns Started: 5588550 ns Scope: /TB_IP File: D:/Ivanov_SLM_Wearout_IP/tb/UVM/DataAcquisitionIP_tb.sv Line:878
Info: Assertion Succeeded @ 5588630000
Time: 5588630 ns  Iteration: 1  Process: /TB_IP//TB_IP  Scope: TB_IP  File: D:/Ivanov_SLM_Wearout_IP/tb/UVM/DataAcquisitionIP_tb.sv Line: 879
UVM_INFO D:/Ivanov_SLM_Wearout_IP/tb/UVM/DataAcquisitionIP_tb.sv(397) @ 5588630000: uvm_test_top.env.agent_out.monitor [Monitor] Actual Transaction sent to the Scoreboard
UVM_INFO D:/Ivanov_SLM_Wearout_IP/tb/UVM/DataAcquisitionIP_tb.sv(398) @ 5588630000: uvm_test_top.env.agent_out.monitor [Monitor] -----------------------------------------------
Name                     Type         Size  Value
-------------------------------------------------
trans                    Transaction  -     @864
  expected_silc_time     integral     32    'd0
  mode                   integral     2     'd0
  module_select          integral     3     'd0
  rosc_tddb_cycle        integral     4     'd0
  silc_cycle             integral     6     'd0
  silc_threshold         integral     12    'd0
  analog_adc             integral     12    'd0
  em_reading             integral     8     'd0
  temp_adc               integral     12    'd0
  volt_adc               integral     12    'd0
  rosc_tddb_freq_ratio   integral     32    'd0
  result                 integral     16    'd1816
  error_code             integral     3     'd1
-------------------------------------------------

UVM_INFO D:/Ivanov_SLM_Wearout_IP/tb/UVM/DataAcquisitionIP_tb.sv(559) @ 5588630000: uvm_test_top.env.scoreboard [Scoreboard] DATA MATCHED!
```

Fig. 9: Console snippet during simulation.



Group - $unit_AMBA_read_sv_2496250219::CovCollector::cg_in

Group Info : cg_in

Score	Num Insts	Avg Insts Score	Weight	Goal	Merge Insts	Get Inst Crvg	Per Inst	Auto Bin Max	Print Missing	Comment
100	1	100	1	100	0	0	0	64	0	

Source File(s) : D:/Ivanov_SLM_Wearout_IP/tb/UVM/DataAcquisitionIP_tb.sv

Group Instance[s] Info : 1 (Total)

Name	Score	Weight	Goal	At Least	Auto Bin Max	Print Missing	Comment
obj.cg_ ...	100	1	100	1	64	0	

Instance : obj.cg_in

Instance Info : obj.cg_in

Score	Weight	Goal	At Least	Auto Bin Max	Print Missing	Is Instantized	Comment
100	1	100	1	64	0	1	

Instance Variable[s] Summary : obj.cg_in

Cover Points	Category	Expected	Uncovered	Covered	Avg Percent
11	Variables	34	0	34	100

Instance Cover Point[s] Details : obj.cg_in

Name	Expected	Uncovered	Covered	Percent	Goal	Weight	At Least	Auto Bin Max	Comment
trans_i ...	2	0	2	100	100	1	1	64	
trans_i ...	7	0	7	100	100	1	1	64	
trans_i ...	3	0	3	100	100	1	1	64	
trans_i ...	3	0	3	100	100	1	1	64	
trans_i ...	3	0	3	100	100	1	1	64	
trans_i ...	3	0	3	100	100	1	1	64	
trans_i ...	1	0	1	100	100	1	1	64	
trans_i ...	3	0	3	100	100	1	1	64	
trans_i ...	3	0	3	100	100	1	1	64	
trans_i ...	2	0	2	100	100	1	1	64	
trans_i ...	4	0	4	100	100	1	1	64	

Cover Points

Inst obj.cg_in

- ✓ trans_in.mode
- ✓ trans_in.module_select
- ✓ trans_in.rosc_tddb_cycle
- ✓ trans_in.analog_adc
- ☐ trans_in.em_reading
- ☐ trans_in.silc_cycle
- ☐ trans_in.silc_threshold
- ☐ trans_in.temp_adc
- ☐ trans_in.volt_adc
- ☐ trans_in.expected_silc_time
- ☐ trans_in.rosc_tddb_freq_ratio

Details of Instance Cover Point - obj.cg_in :: trans_in.mode

Summary of Cover Point - trans_in.mode

Category	Expected	Uncovered	Covered	Percent
User Defined Bins	2	0	2	100
Ignore bin[s]	1			
Empty bin[s]	2			

User Generated Bins for Instance Cover Point - obj.cg_in :: trans_in.mode

Covered bins

Name	Hit Count	At least
auto[1]	5050	1
auto[2]	7005	1

Ignore Bins for Instance Cover Point - obj.cg_in :: trans_in.mode

Ignore bins

Empty Bins for Instance Cover Point - obj.cg_in :: trans_in.mode

Name
auto[0]
auto[3]

Details of Instance Cover Point - obj.cg_in :: trans_in.module_select

Summary of Cover Point - trans_in.module_select

Category	Expected	Uncovered	Covered	Percent
User Defined Bins	7	0	7	100

User Generated Bins for Instance Cover Point - obj.cg_in :: trans_in.module_select

Covered bins

Name	Hit Count	At least
rosc	1723	1
analog	1722	1
em	1723	1
tddb	1720	1
silc	1724	1
temp	1721	1
volt	1722	1

Details of Instance Cover Point - obj.cg_in :: trans_in.rosc_tddb_cycle

Summary of Cover Point - trans_in.rosc_tddb_cycle

Category	Expected	Uncovered	Covered	Percent
User Defined Bins	3	0	3	100

User Generated Bins for Instance Cover Point - obj.cg_in :: trans_in.rosc_tddb_cycle

Covered bins

Name	Hit Count	At least

Fig. 10: Functional coverage report.