Using a Generic Plug and Play Performance Monitor for SoC Verification

Bhavin Patel

Janak Patel

Kaushal Modi

Ajay Tiwari

eInfochips
Ahmedabad, India
www.einfochips.com

ABSTRACT

Performance validation, while often critical for SoC verification, is usually implemented as an afterthought, leading to inefficiencies. A case study of a complex performance-critical SoC involving a large number of communication paths is presented. The SoC is based on an OCP/AXI/APB Bus Interconnect with 20+ bus masters and 80+ bus slaves. The ad-hoc approaches for analysis and reporting for this SoC became increasingly cumbersome due to the following conditions:

- *Different protocol interfaces at source and destination*
- *Addition of new paths for the derivative SoCs*
- *Path-specific measurement requirements*
- *Interleaved traffic patterns*
- *Changes in performance requirements during the course of project*

To overcome these issues, a generic plug-and-play performance monitor component was developed. It had to be protocol-independent, a plug-and-play entity and easily integrated into a SoC verification environment like a typical UVM verification component. Using this monitor, a significant improvement in performance verification closure was achieved.

Table of Contents

1. Genesis of Performance Monitor 3
2. Architecture of Performance Monitor 7
3. Configuration 11
4. Measurements Supported 15
5. Reporting 17
6. Use of Performance Monitor in a testbench 20
7. Summary 25

Table of Figures

Figure 1 Performance Monitor Hierarchy 7
Figure 2 Detailed Class Hierarchy of Performance Monitor Environment 9
Figure 3 Example of instance based configuration 12
Figure 4 Sample CSV file 13
Figure 5 Example of setting configuration through CSV 14
Figure 6 Example of simulation summary report 17
Figure 7 Example of a message displayed in summary 18
Figure 8 Example of bandwidth report 18
Figure 9 Example of transaction latency report 18
Figure 10 Example of a trace file 19
Figure 11 Example system having a single master and slave 20
Figure 12 Steps for Performance Monitor Usage 21
Figure 13 Performance environment instance in top environment of the testbench 22
Figure 14 CSV file for example system 23
Figure 15 Performance transaction building and driving 23
Figure 16 Connection between testbench environment and performance monitor 24
Figure 17 Performance monitor integration in environment 24

Table of Tables

Table 1 Performance transaction fields 9
Table 2 Configuration Control at Each Hierarchy 11
Table 3 Configuration parameters for example system 22

1. Genesis of Performance Monitor

With SoCs becoming more complex by the passing day, performance verification has become an integral part of SoC front-end verification closure. The parameters of performance verification, namely latency and bandwidth, have to be measured and verified against an expected value for all the paths of interest. With a multitude of IPs connected and talking to each other, there is a possibility of some of these paths not meeting the requirement for performance as they may be sharing a common medium for communication, for e.g.: an SoC interconnect or a network switch. In such cases, the IPs may eat into each other's bandwidth or affect the latency. Given such scenarios, the necessity of performance verification cannot be overlooked.

In one of our recent projects, the SoC had an OCP/AXI/APB Bus Interconnect, connected with 20+ bus masters (such as Dual Core Processors, DDR controller, PCIe controller) and 80+ bus slaves (Peripheral IPs, System RAM, DDR) and thereby had multiple paths of communication. This complexity of SoC made its performance verification a challenging task.

We divided the verification approach into following broad categories/steps:

a) *Traffic generation*
   Where the necessary traffic load was generated and driven on the path
b) *Event collection*
   Collection of start and end time for transactions, captured at required event
c) *Measurement through collected data*
   Having collected the data, perform calculations as per requirement
d) *Reporting of results*
   Displaying messages indicating the success or failure of meeting the requirements along with other related data

We picked the start time and the end time of transactions, collected the data bytes in the monitor callbacks for respective protocols, and in the absence of a uniform and standard method, performed the needed calculations. Since the start time and end time capturing conditions and successful performance requirement for different protocols were different, the calculations for different protocols could not be put in a single method. This meant a lot of coding – per protocol, per traffic pattern requirement. As an example, in our project, the Read and Write traffic latency requirements were different between a particular master and slave, which demanded separate calculations. In addition, the success or failure of meeting the requirement was printed as an info or error respectively in the log file. This necessitated the use of an extra script that will pick messages of interest and give a summary or else we had to look for them manually in the simulation log files. While we went about the task, we also had to keep an eye on the time at our disposal, which is usually critical in verification projects.

The approach worked fine and we were able to close the performance verification on time. However, further reduction in the effort during the performance verification cycle was required. The effort described above was reasonable if this was a one-time effort. If new paths were added, or if the requirements changed, chopping and changing was required at all the stages of the cycle, even if it was as little as changing the info messages. Since the code and calculation was specific to a project, next project demanded a complete repetition of the cycle. Driven by this demand, we identified the requirements of developing a general purpose, re-usable performance monitor and identified the following challenges that the monitor should take care of:

- The SoC IPs communicate using a host of protocols so the monitor should be protocol independent
- Each path can have specific measurement requirements. The requirements can include interleaved traffic patterns within single path. The architecture of the monitor has to take care of this important requirement.
- Performance requirements can be changed during the course of the project so the monitor should be configurable
- There can be a possibility of addition of new paths in derivative SoC apart from the existing multiple paths over which performance has to be measured.
- The reporting of results has to be detailed and easy to read and analyze so that no additional script development is required.

To overcome above challenges, a highly configurable Performance Monitor was developed with UVM Methodology base. This performance monitor is a protocol independent, plug-and-play entity that can be integrated easily into the SoC verification environment like any other verification component. The user has to integrate the monitor in verification environment, configure it as per requirement and the rest of the steps of performance verification will be taken care of by the monitor upon being provided the transaction details such as start time, end time and data bytes. These steps are explained with examples in the sections that follow. The different options available for configuration allow the user to tailor the measurement according to the requirement. These configuration options include the number of interfaces to be verified, traffic patterns over each interface, reporting style of results among others. In case of changed requirements, user has to reconfigure only the component. The reporting mechanism is uniform and detailed for each path. Since the user is not developing any extra component or any supporting script, it reduces development time, which in turn leads to significant reduction in the overall time consumed in performance verification. Following sections discuss the features of this monitor, its architecture, its usage in testbench with an example and a case study of the use of monitor in actual project.

1.1 Features of Performance Monitor

- The monitor gives user a control of the measurements one wants to do. The user can choose between the following measurement type or perform both simultaneously:
    - *Latency measurement*– Here latency measured can be end-to-end latency (latency for a request/response to reach from one point to another in the design) or round-trip latency of transaction (request to response latency). The monitor is able to provide the latency value for each transaction and an average value of latency over a number of transactions.
    - *Bandwidth measurement* – Bandwidth can be measured over user defined number of transactions (e.g. between 25th and 75th transaction) or between user defined events (e.g. when the transaction size varies in DMA transfer, the bandwidth can be measured between DMA start and DMA end), depending upon user's requirement.

- The results will be reported (in tabular form) in user's choice of units. These include:
    - *Latency* : ms, us, ns, ps
    - *Bandwidth* : KBps, MBps, GBps, Kbps, Mbps, Gbps

- A user-friendly method of configuration is provided. User puts the configurations in a .csv (comma separated value) file which is read by the monitor and gets configured accordingly(This method is in addition to the regular way of taking instance of the configuration classes and updating the configuration parameters).

- The user can configure expected values of latency and bandwidth, which the monitor will use to compare the measured values. If the measured values are not as expected, an error will be thrown from the monitor. The measurement path that is not meeting the requirements is also indicated, which helps in quick debugging and saves manual effort.

- Configuration compliance checks are provided.

- Some requirements may ask for 'Setup' and 'Hold' transactions facility. The Setup transactions are the transactions before the measurements start and Hold transactions are the ones after the measurement is completed. Since these transactions take care of the cold start effects at the beginning and trailing edge effects at the end of the flow, these number of transactions are neglected in calculation. A support for configuring the number of such transactions is provided. Also the window size (number of transactions over which the latency or bandwidth has to be measured) is user defined. User can choose same window of transactions or different windows for latency and bandwidth measurement. Measurement over multiple windows is possible.

- A support for alternate window, other than the configured window over which parameters are measured, is provided which helps in checking the consistency of results over the

simulation. For example, say the user measured the average latency between 40th and 80th transaction, but wants to know the average latency between 30th and 90th transaction for consistency purpose. This can be achieved by configuring the required parameters at the beginning of simulation. This number can also be provided as a percent of the total number of transactions (e.g. between 30% and 90% of total transactions). The result of this window is reported alongside the summary of primary window, which makes the comparison easier.

- The monitor has rich result reporting features, the verbosity of which is configurable. All the details related to the measurement are displayed in tabular form. The reporting configurations can help the user to point to the exact window in a traffic pattern within an interface that is not meeting the performance requirement thereby helping in the debugging process.

- The user can compare the performance results across entities (interface), if needed, and get the cumulated bandwidth for different traffic patterns within an interface. For e.g., between a master and a slave, requirement can be of 100MB/s of combined read and write traffic of which 70MB/s is read traffic and 30 MB/s is write traffic. All these requirements can be verified by configuring the performance monitor.

- In case a variation of results is expected, user can provide this value as ‘tolerance’ value. The performance monitor will define a new value for ‘expected’ parameter depending upon the tolerance value and shout error only if the new value is violated in measurement.

- Callbacks have been provided that give the performance data values for coverage purposes. In case the testcase writer wants to base the behavior of test upon the results of measurements, APIs have been provided that return the measured values and status of simulation.

- The user can also change the monitor configuration run-time in which case the monitor will use the updated configuration for measurement and reporting.

  All these features help in making the component flexible and re-usable across projects.

2. Architecture of Performance Monitor

The measurement over different interfaces and traffic patterns is taken care of by the hierarchical structure of the performance monitor. This architecture is discussed in this section.

2.1 Detailed Architecture

For measuring performance of multiple traffic requirements on same interface, performance monitor is designed with three levels of hierarchy as shown in figure below. Configuration options are available at each level. All class components are extended from "uvm_component" and all configurations are extended from "uvm_object" of UVM class library.

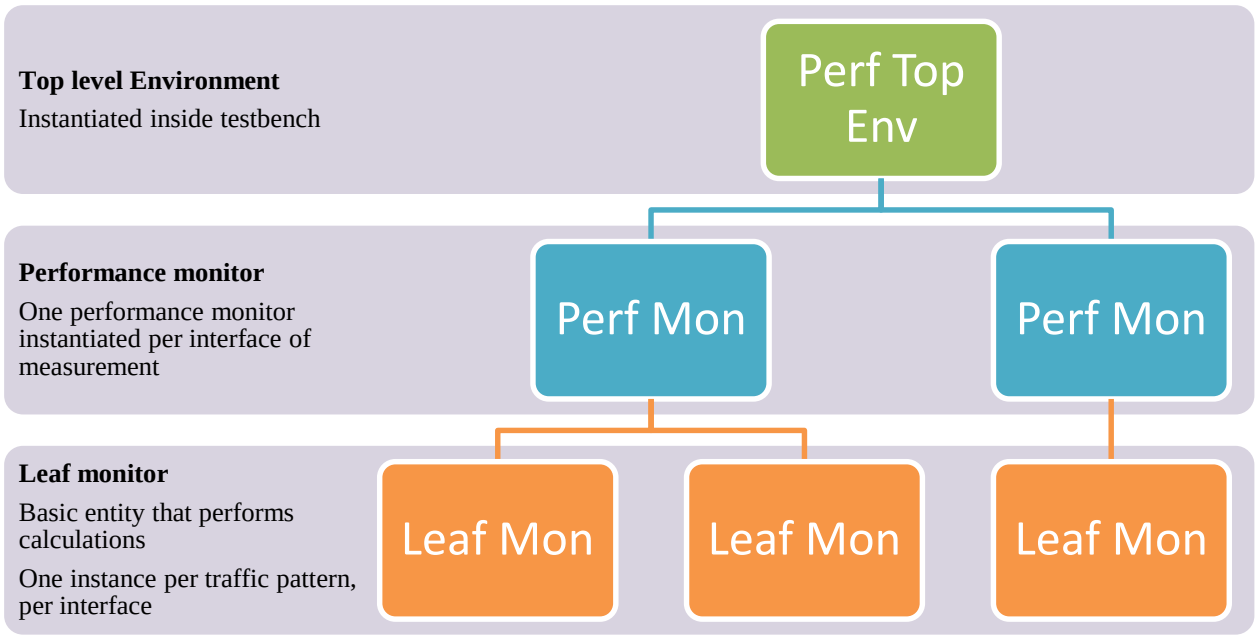


Figure 1 Performance Monitor Hierarchy

Top hierarchy class of performance environment is called as ***performance monitor environment (level -1)***. This class has its own configuration called ***performance environment configuration (level -1 cfg)***. This class will be instantiated by user in testbench to use this monitor. It contains the dynamic array of performance monitor and user can provide the number of performance monitor to be created through configuration of this component. Upon creation of performance environment class, it creates the instances of performance monitor based on configuration.

Next level in hierarchy is the ***performance monitor (level -2)***. This is a component, which can be used to calculate the bandwidth and latency at any entity (bus interface/serial interface/IP user interface or any customized requirement). Specifically, one performance monitor is needed for measuring the performance requirements on one interface (E.g. at

AHB/AXI/OCP system bus interface or RGMII interface of Ethernet or at USB interface, etc.). If it is required to measure parameters of more than one entity, then it is possible to create multiple instances of performance monitor through level-1 configuration. This component contains a UVM analysis port of performance transaction type, through which it will receive the required information (in the form of performance transaction) to calculate bandwidth and latency. Thus, for measurement, user will need to map the required information (such as start time, end time, number of data bytes) into performance transaction and write it through analysis port from inside the testbench.

If more than one type of traffic is flowing on single entity, then it may require measuring each type of traffic separately. For example,

1. Processors are using same bus interface for instruction load and store. However, traffic requirement of load and store are different, so it may require verifying bandwidth and latency for both type of traffic separately.
2. On USB bus, there is bi-direction data traffic, one is from Host to Device and the second is from Device to Host. In that case, it may require checking bandwidth for both direction traffic individually.

In order to meet this kind of requirement performance monitor contains a third level of hierarchy called the ***leaf monitor (level -3)***. A leaf monitor is needed per traffic pattern, per interface. The performance monitor contains dynamic array of leaf monitor and can be configured for required number of leaf monitors equal to number of different traffic to be measured using level-2 configuration. Minimum of one leaf monitor is required per performance monitor for measurement. The configuration options for leaf monitor (level -3 cfg) contain the controlling flag, as well as expected reference values for bandwidth and latency. This component also does reporting of calculated bandwidth and latency in log file, as well as dumps the information in trace file based on the configuration.

When the leaf monitor is created, a unique ID is assigned to each leaf monitor from configuration based on traffic type. A corresponding field has been provided in the performance transaction as well. So while performance transactions for all the traffic will be received through the same analysis port in performance monitor, the traffic will be differentiated based on ID field of each transaction and since there will be one leaf monitor for each traffic type with matching ID, the packet will be navigated to the corresponding leaf monitor for calculations.

Leaf monitor contains the logic of bandwidth and latency calculation performed on the performance transaction. As mentioned above, since the performance monitor can be used with a variety of interfaces, there is a need for a generic performance transaction type. The following table mentions the fields of performance transaction.

Field	Description
Id	Decides the leaf monitor that this transaction is intended for
req_lat_start_time	Stores the start time for latency calculation
req_lat_end_time	Stores the end time for latency calculation
bw_start_time	Stores start time for bandwidth calculation
bw_end_time	Stores end time for bandwidth calculation
data_bytes	Stores the total number of bytes being transferred in a transaction

Table 1 Performance transaction fields

User will update these fields when generating the performance transaction (inside a testbench component or hook-up from where start time and end time of transactions are available) upon the occurrence of required event and pass it to performance monitor through connecting port.

Detailed class hierarchy of performance environment and configuration class is shown in below figure.

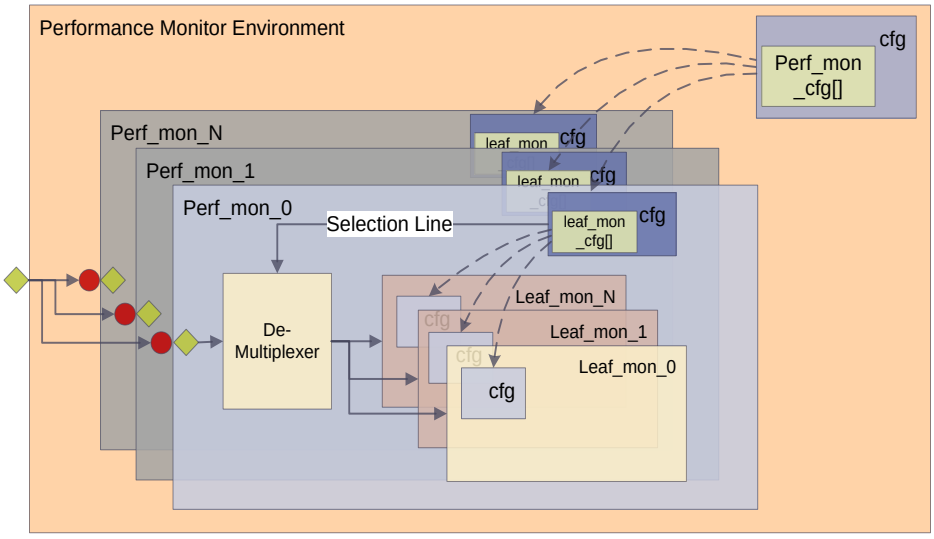


Figure 2 Detailed Class Hierarchy of Performance Monitor Environment

In summary, there is one performance monitor (level-2) from performance environment used per measuring interface and each performance monitor has one dedicated analysis port through which it will receive the performance transaction. Each performance monitor has leaf monitor (level -3), which calculates and reports measured bandwidth and latency. There can be

multiple leaf monitors, if it is required to measure different types of traffic simultaneously on same entity/interface.

Next section discusses about the configuration parameter methods to configure the performance monitor, based on which the operation of the performance monitor will be decided.

3. Configuration

Similar to the hierarchy of performance monitor components, there is a hierarchy of configuration classes, with each one dedicated to corresponding hierarchy component as discussed in previous section. The table below describes configuration parameters at each level. These parameters need to be set based on requirement for performance verification.

Level	Hierarchy Name	Configuration Control
L1	Performance Environment Configuration	• Number of performance monitor(s) • ID of each performance monitor • Uniformity check configuration
L2	Performance Monitor Configuration	• Number of Leaf Monitor • ID of each leaf monitor • Cumulative measurements configuration
L3	Leaf Monitor Configuration	• Measurement types • Expected values for each measurement • Measurement window information and window type • Reporting level and trace file configuration

Table 2 Configuration Control at Each Hierarchy

Following two easy-to-use methods are available to configure the performance monitor:

1) Configuration class based approach
2) Comma Separated Values (CSV) file based configuration approach

3.1 Configuration class instance approach

This is a common approach for configuration of any component. Instance of configuration class for each level is taken in the testbench. As performance monitor is developed in UVM methodology, each instance needs to be created in "*build_phase*" of user testbench or testcases. Therefore, in the build phase, user has to create an object of this configuration and then assign appropriate values to its members based on requirement of each level. Below example demonstrates the creation of performance monitor.

```
20  // Instance of Performance Enironment Configuration class(Level 1)
21  perf_top_env_cfg_c cfg;
22  //-----------------------------------------
23  // Instances of Performance Monitor Configuration class(Level 2)
24  perf_mon_cfg_c     mon_cfg[];
25  //-----------------------------------------
26  // Instances of Performance Monitor Configuration class(Level 3)
27  perf_leaf_mon_cfg_c leaf_cfg [perf_mon_cfg_c][];
28  //-----------------------------------------
29
30  virtual function build_phase();
31  begin
32    // Creating Top Level Configuration (Level - 1)
33    cfg = perf_top_env_cfg_c::type_id::create("cfg");
34
35    // Creating instance of Performance monitor (L-2)
36    mon_cfg     = new[num_of_perf]; // num_of_perf - Based on requirement
37
38    // Creating instance of Leaf monitor (L-3) for for each performance monitor
39    foreach(mon_cfg[i])
40    begin
41      mon_cfg[i] = perf_mon_cfg_c::type_id::create($sformatf("mon_cfg[%0d]",i));
42      // Creating leaf Monitor
43      // leaf_per_mon[i] - Based on requirement of
44      //                   no. of leaf monitor per performance monitor
45      leaf_cfg[mon_cfg[i]] = new[leaf_per_mon[i]];
46      // Configuring Measurement Types at each leaf monitor
47      for(int leaf_no = 0; leaf_no < (leaf_per_mon[i]); leaf_no++)
48      begin
49        leaf_cfg [mon_cfg[i]] [leaf_no] =
50             perf_leaf_mon_cfg_c::type_id::create($sformatf("mon_cfg_%0d_leaf_cfg[%0d]",i,leaf_no));
51        leaf_cfg[mon_cfg[i]][leaf_no].leaf_mon_en = 1;
52        leaf_cfg[mon_cfg[i]][leaf_no].avg_lat_calc_en = 1;
53        leaf_cfg[mon_cfg[i]][leaf_no].exp_avg_lat_per_window = 50;
54        // Set leaf Monitor configuration (L-3) to corresponding Performance monitor (L-2)
55        mon_cfg[i].add_perf_leaf_mon_cfg_f(leaf_no,leaf_cfg[mon_cfg[i]][leaf_no])
56      end
57      //-------------------------------------------------------------------
58      // Set Performance monitor configuration (L-2) to Perfromance env Configuration (L-1)
59      cfg.add_per_mon_cfg_f({master_cfg_name,$psprintf("_%0d",i)},mon_cfg[i]);
60    end
61    //-------------------------------------------------------------------------
62    // Creating Performance Monitor
63    top_env_inst = perf_env_c#(perf_trans_c)::type_id::create("top_env_inst",this);
64    //-------------------------------------------------------------------------
65    // Setting Top level configuration (L-1) to performance monitor
66    uvm_config_db#(perf_top_env_cfg_c)::set(this,"top_env_inst","perf_env_cfg_inst", cfg);
```

Figure 3 Example of instance based configuration

Line numbers 55 and 59 show how the APIs are used to pass the configurations across levels. The configurations of lower level component are stored in a dynamic array of the component immediately above it in the hierarchy. (L3-cfg stored in array in L2-cfg, and L2-cfg stored in array in L1-cfg). User passes the configuration instance and name of the component to this APIs.

3.2 CSV file based configuration approach

This is a user-friendly approach to configure the performance monitor. There is one pre-defined format of CSV file, which contains all required parameters for configuration of leaf monitor, performance monitor and top-level configurations. User just has to fill this CSV file and use it with the performance monitor in the testbench. This CSV file will be read by the monitor and configuration for all the levels will be set as defined in the file.

3.2.1 Example of setting configuration through CSV

Below figure shows the format of CSV file. User needs to configure this file for each level of configuration base on the requirement. There is a separate set of configuration parameters provided in the file for each level of hierarchy. Parameter for each level is recognized by LEVEL column. The performance environment configuration will be provided with the LEVEL column value **L1**. Correspondingly, the performance monitor configuration will be provided with a value **L2** and the leaf monitor configuration with the value **L3**. If there are more than one leaf monitor under one performance monitor, then multiple rows can be created for each leaf configuration.

The sample CSV below shows that two performance monitors: one for AXI and the other for OCP. For each interface, there is requirement of measuring bandwidth and average latency for *read* and *write* transaction. So there are two L2 level of configurations (one for OCP and one for AXI, provided with *PERF_MON_NAME*) and each is having two L3 level of configuration (for READ and WRITE, provided with *LEAF_MON_ID*). All such requirements can be created in single set of configuration and can be used for all the testcases.

Sr. No.	CONFIG ID	SEQUENCE NAME	LEVEL	NUM OF PERF MON	PERF MON NAME	NUM OF TRANS TYPE	TYPE NAME	LEAF MON ID	MEASUREMENT TYPE	EXPECTED LATENCY	LATENCY UNIT	EXPECTED BANDWIDTH	BANDWIDTH UNIT
1	0	ocp_axi_read_write	L1	2									
			L2		OCP	2							
			L3				READ	0	AVG_LATENCY + BANDWIDTH	48	ns	100	MBps
			L3				WRITE	1	AVG_LATENCY + BANDWIDTH	48	ns	100	MBps
			L2		AXI	2							
			L3				READ	0	AVG_LATENCY + BANDWIDTH	48	ns	100	MBps
			L3				WRITE	1	AVG_LATENCY + BANDWIDTH	48	ns	100	MBps

Figure 4 Sample CSV file

It is also possible that different testcases may have different configuration requirement or there may not be requirement of some components while measuring performance for dedicated path or interface. E.g. If there two independent paths for which we need to measure performance and we have two testcases. Therefore, it is better to create performance monitor dedicated to the path in particular testcase. In such case, multiple set of configurations can be created for each testcase or set of testcases within same CSV by using *CONFIG_ID* or *SEQUENCE_NAME* parameter.

After creating CSV file, an API is used to apply configuration from the CSV file, named *get_perf_config_for_seq_f*. In this API, first argument is the name of the CSV file and second is the name of testcase (if more than one set of configurations are used, otherwise by default it will take first one). Below snippet shows the example of setting configuration from “sample.csv” for “ocp_axi_read_write_seq” sequence.

```
perf_top_env_cfg_c cfg;
cfg = perf_top_env_cfg_c :: type_id :: create("cfg");
cfg.get_perf_config_for_seq_f("sample.csv", "ocp_axi_read_write_seq");
```

Figure 5 Example of setting configuration through CSV

Once the monitor is connected and configured as per requirement, the performance monitor is ready for measurements. The types of measurement that the performance monitor is able to do are discussed in the next section.

4. **Measurements Supported**

The performance monitor is capable of performing different types of measurements based on the configuration. All the calculations related to these measurements are done using the data received in the performance transaction. The types of measurements that the monitor is able to perform are:

1) *Per Transaction Latency Measurement:*

   This measurement is done when the latency on a particular interface is to be measured without traffic on the other interfaces or traffic requirement is of guaranteed service. The performance monitor will check for each transaction meeting expectations and report an error in case of failure for any of the transactions.

2) *Bandwidth and Average Latency Measurement over a number of transactions:*

   This measurement gives the value of bandwidth and latency over a user-defined window of transactions. The monitor will take into considerations the 'Setup' and 'Hold' number of transactions in this measurement and perform calculations over user-defined window.

   The average latency is measured when the traffic is moving on the other interfaces as well along with the interface of interest. In this case, the value of latency being measured on any interface can vary depending upon the traffic on other interfaces. Therefore, an average or RMS value of latency is taken in such cases. The average value of latency can be measured along with 'Per Transaction' latency measurement.

3) *Bandwidth and Average Latency Measurement between two events:*

   This measurement is done when data contained in each request is not constant and measurement window has to be defined by the total bytes to be transferred.

   For example, when the total number of bytes to be transferred from DMA engine is 4KB, the bandwidth to be measured is regardless of the total number of requests sent from DMA. Measurement should start as soon as the DMA transfer starts and end once 4K transfers complete. The user can assign a start value of time in performance transaction upon DMA start. Then each subsequent transactions should have the time value equal to "-1" until DMA stop occurs. Upon the DMA stop event, the end value of time should be assigned in the performance transaction. A value of "-1" in any of the time related field is not considered by the monitor for calculation. The performance monitor will collect all the data bytes provided to it in between this start- and end-time and perform the calculations accordingly.

4) *Latency and Bandwidth measurement over an alternate window:*

This measurement helps to get the parameter measurement over a different window than the one defined by the user in the simulation. This helps to know the consistency of results. In this measurement, the monitor collects all the data received during the simulation and at the end of the simulation. It re-calculates the desired parameter over the alternate window defined by the user. This measurement can also be performed independently of any other measurement.

5) *Cumulative Bandwidth measurement:*

If there is a requirement of different traffic patterns meeting a bandwidth requirement collectively, cumulative bandwidth measurement feature of monitor helps. When enabled, the performance monitor accumulates the bandwidth over all the traffic patterns and compares it against the configured expected value.

6) *Uniformity Comparison:*

If the measured value for any interface has to be compared against the other interface, uniformity comparison can be used. One example is the performance comparison over multiple instances of the same type of interface (like checking performance of multiple GPUs of same type).

5. Reporting

After finishing the measurements, the monitor displays all the results and related information. While deciding the reporting structure, the main objective was to aid the process of debugging. Keeping this in mind, a tabular form of reporting was decided. In addition, a UVM_ERROR is reported whenever the required conditions are not met, helping to catch the erroneous case quickly. The errors are reported irrespective of the reporting levels (discussed below).

The monitor does two types of reporting:

1) Log file reporting
2) Trace file reporting

The information printed in log files can be controlled by configuration of the monitor as per the details needed. The levels of reporting are as follows.

5.1 Log File Reporting

5.1.1 Default Reporting

A summary of results is always printed at the end of the simulation, in report_phase, by default. This summary has the following information:

1) Name of the leaf monitor. This helps to identify the path and traffic under measurement
2) Number of windows and size of the windows
3) Expected bandwidth/latency
4) Actual bandwidth/latency measured (an average over all the windows)
5) Minimum and maximum bandwidth/latency for a window over complete measurement
6) Number of windows that did not meet the requirement

An example of this default report is shown in the following figure:

SIMULATION SUMMARY FOR BANDWIDTH :: PERF_MON_master_0_LEAF_0_RD

Total Windows	Window Size	Total Trans	Valid Trans	Exp. BW(MBPS)	Tolerance (MBPS)	Avg. BW(MBPS)	Min. BW(MBPS)	Max. BW(MBPS)	Unmatched Windows
0034	0111	03942	03942	60.25	4.40	62.47	50.07	73.91	00009

Figure 6 Example of simulation summary report

The alternate window bandwidth and latency information, if enabled, will also be printed in default summary. This value also will be compared with the expected alternate window bandwidth and latency.

Above message will be displayed as an error in case the average values or the alternate window values do not meet the requirement.

A display message in the default report will inform the user about the measurements that have been disabled. Below figure gives an example of such a message.

```
================================================================================================
|| MEASUREMENT FOR AVG LATENCY IS DISABLED :: PERF_MON_master_0_LEAF_0_RD                     ||
||____________________________________________________________________________________________||
```

Figure 7 Example of a message displayed in summary

5.1.2 Reporting Level -1

In reporting level – 1, apart from the default reporting, information related to windows of measurement will be reported runtime at the end of each window. It contains following information.

1. Window number and transactions in that window
2. Start and End transfer count for each window
3. Min, max and average bandwidth and latency
4. Start and end time as well as data bytes
5. Error difference between the expected and the average value (which can be used to calculate RMS value of error). This value is reported only if the requirement not met (i.e. if bandwidth measured is less than expected or latency is greater than expected)

```
=============================================================================================================
|| Bandwidth for window no : 0001                                                                           ||
||=========|========|========|========|===========|===============|===============|============|============|============||
||  Window  |  Total  |  Start  |   End   |   Total   |   Start Time   |    End Time    |     Exp.    |     Act.    |      BW     ||
||    id    | Request |    id   |    id   |   Bytes   |      (ns)      |      (ns)      |  BW(MBPS)  |  BW(MBPS)  | Diff (MBPS) ||
|| -------- | ------- | ------- | ------- | --------- | -------------- | -------------- | ---------- | ---------- | ---------- ||
||   0001   |  00256  |  00001  |  00256  |  00000128 |          0.86  |       1714.27  |     60.00  |     74.70  |      0.00  ||
||=========|========|========|========|===========|===============|===============|============|============|============||
```

Figure 8 Example of bandwidth report

5.1.3 Reporting Level -2

In reporting level -2, in addition to default and level-1 reporting, information related to each transaction namely per transaction latency information, start time, end time will be reported runtime in log file itself.

```
=============================================================================================
|| Latency for transaction no : 00004                                                      ||
||==========|================|================|=============|=============|=============||
||  Request  |    Start Time   |     End Time    |     Exp.    |     Act.    |   Latency   ||
||    id     |       (ns)      |       (ns)      | Latency(ns) | Latency(ns) |   Diff(ns)  ||
|| -------- | -------------- | -------------- | ----------- | ----------- | ----------- ||
||   00004   |         10.33  |          20.00  |     10.00   |      9.67   |       0.00  ||
||==========|================|================|=============|=============|=============||
```

Figure 9 Example of transaction latency report

5.1.4 Reporting level -3

This reporting level is for debugging. In this reporting level, all information will be reported those reported in reporting level 2 and with debug messages in log file.

Debug messages will give information about currently calculating information like:

1. Full simulation calculation
2. Start id and end id of full simulation
3. Selected full simulation window type
4. Total number of transactions available in queue
5. Full simulation bandwidth/latency and with start time and end time

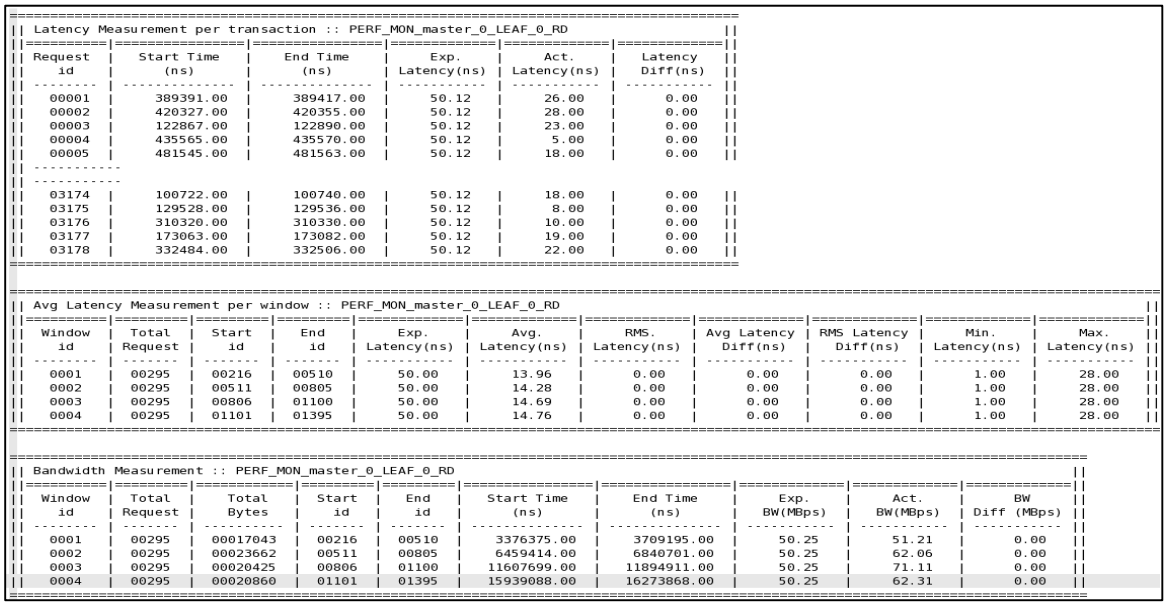
Since all the information related to measurement is available at one place, as an info or error, along with the name of the leaf monitor, the debugging process speeds up.

5.2Trace File Reporting

Reporting of the performance parameters is dumped in trace file if enabled. This trace file can be used to generate a graph of results by reading it in an Excel file. For each simulation, two types of trace can be done as per configuration parameters:

1. Leaf monitor trace file:Includes the latency information per transaction, average latency information per window and bandwidth information per window. It generates a trace file from each leaf monitor as per configuration set.
2. Default reporting trace file:Includes the default reporting for all leaf monitor.

An example of a trace file is given in figure below:



Latency Measurement per transaction :: PERF_MON_master_0_LEAF_0_RD

Request id	Start Time (ns)	End Time (ns)	Exp. Latency(ns)	Act. Latency(ns)	Latency Diff(ns)
00001	389391.00	389417.00	50.12	26.00	0.00
00002	420327.00	420355.00	50.12	28.00	0.00
00003	122867.00	122890.00	50.12	23.00	0.00
00004	435565.00	435570.00	50.12	5.00	0.00
00005	481545.00	481563.00	50.12	18.00	0.00
03174	100722.00	100740.00	50.12	18.00	0.00
03175	129528.00	129536.00	50.12	8.00	0.00
03176	310320.00	310330.00	50.12	10.00	0.00
03177	173063.00	173082.00	50.12	19.00	0.00
03178	332484.00	332506.00	50.12	22.00	0.00

Avg Latency Measurement per window :: PERF_MON_master_0_LEAF_0_RD

Window id	Total Request	Start id	End id	Exp. Latency(ns)	Avg. Latency(ns)	RMS. Latency(ns)	Avg Latency Diff(ns)	RMS Latency Diff(ns)	Min. Latency(ns)	Max. Latency(ns)
0001	00295	00216	00510	50.00	13.96	0.00	0.00	0.00	1.00	28.00
0002	00295	00511	00805	50.00	14.28	0.00	0.00	0.00	1.00	28.00
0003	00295	00806	01100	50.00	14.69	0.00	0.00	0.00	1.00	28.00
0004	00295	01101	01395	50.00	14.76	0.00	0.00	0.00	1.00	28.00

Bandwidth Measurement :: PERF_MON_master_0_LEAF_0_RD

Window id	Total Request	Total Bytes	Start id	End id	Start Time (ns)	End Time (ns)	Exp. BW(MBps)	Act. BW(MBps)	BW Diff (MBps)
0001	00295	00017043	00216	00510	3376375.00	3709195.00	50.25	51.21	0.00
0002	00295	00023662	00511	00805	6459414.00	6840701.00	50.25	62.06	0.00
0003	00295	00020425	00806	01100	11607699.00	11894911.00	50.25	71.11	0.00
0004	00295	00020860	01101	01395	15939088.00	16273868.00	50.25	62.31	0.00

Figure 10 Example of a trace file

6. Use of Performance Monitor in a testbench

This chapter demonstrates the use/integration of the performance monitor in a user's testbench. This is described using example of an OCP master connected to an OCP slave through interconnect.

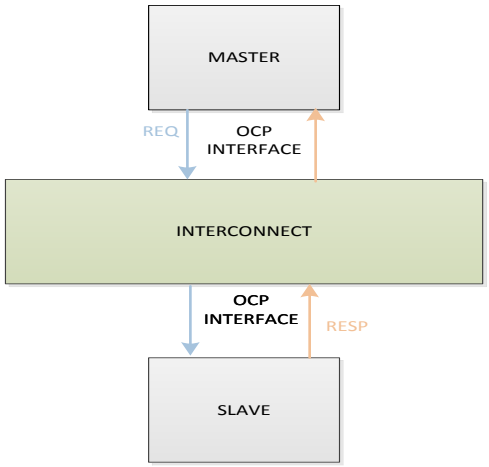


Figure 11 Example system having a single master and slave

Let us assume that the specification and requirement of this hypothetical system example is as follows:

- Three types of traffic flows between master and slave:
  1. Random write traffic
  2. Random read traffic
  3. Sequential read traffic
- The master should be able to drive 1.6 GB/s of combined traffic to the slave. Out of this combined traffic,
  1. 1.2 GB/s must be for sequential read traffic
  2. 300MB/s for random reads
  3. 100MB/s for random writes
- The request to response read latency from the master to the slave shall not exceed 130 ns.

The steps involved in use of the performance monitor are mentioned in the diagram below. These steps are explained using the system shown in figure 11.

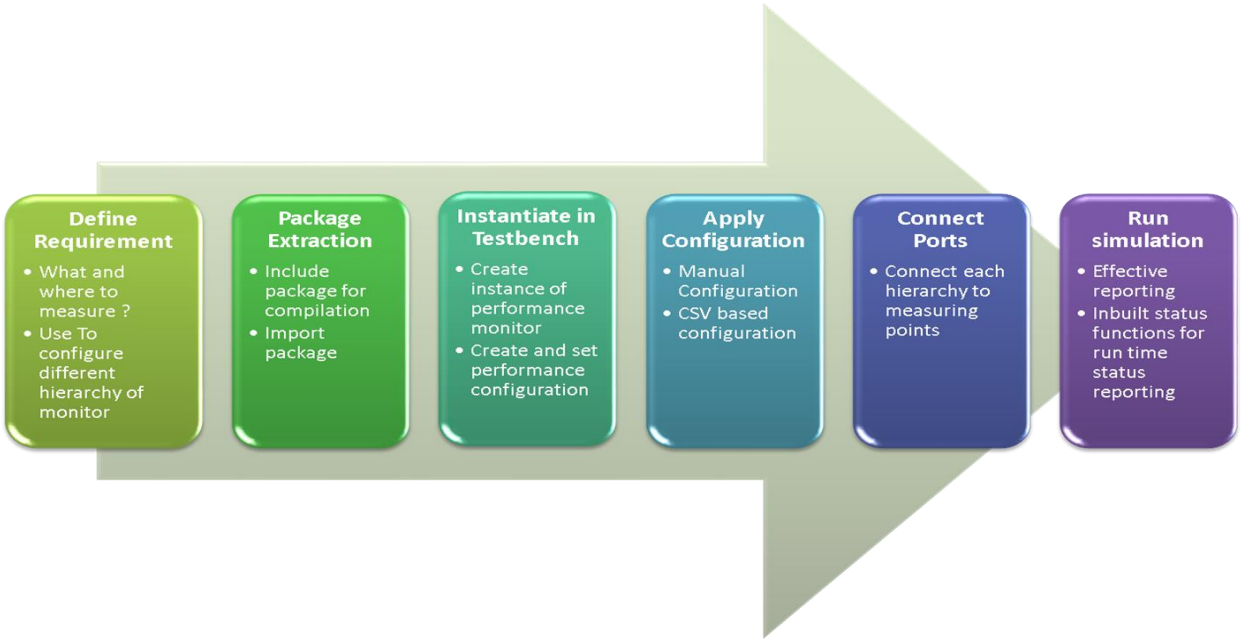


Figure 12 Steps for Performance Monitor Usage

1) *Requirement definition:*

At the outset, user needs to determine the performance parameters those are of interest for a particular design under verification. In addition, the number of interfaces over which performance needs to be measured must be known along with the traffic patterns. These definitions help in configuring the monitor according to the requirement.

In case of our example system, we filter out following information:

- Bandwidth and latency both are required to be measured.
- Since the requirement is of request-to-response latency, we will have to observe the interface between the master and interconnect, where both request and response can be observed. So accordingly performance monitor to be integrated.
- 3 types of traffic flows on the interface i.e. Write, Read and Sequential Read. According to the architecture of performance monitor, we will need 3 leaf monitors – one per traffic type.

2) *Importing Package:*

The Performance Monitor files are provided as a package that must be included and imported in testbench.

3) *Instantiation in Testbench:*

User must then create an instance of the top environment of performance monitor and its configuration in the top environment of the testbench.

```
class sample_top_env_c extends project_top_env_c;

  /*! instance of virtual sequencer */
  sample_vir_sqr_c vir_sqr_inst;

  /*! Instance of top configuration  */
  sample_top_cfg_c top_cfg_inst;

  /*! Extended OCP master monitor callback used to provide information related to
      performance measurement */
  sample_ocp_master_monitor_callback ocp_cb_inst[];

  string slave_s;

  perf_env_c#(perf_trans_c) top_env_inst;

  ...
  ...
function void sample_top_env_c::build_phase(uvm_phase phase);

  ...
  ...
  super.build_phase(phase);

  top_env_inst = perf_env_c#(perf_trans_c)::type_id::create("top_env_inst",this);
  ...
  ...
```

Figure 13 Performance environment instance in top environment of the testbench

4) *Configuration:*

Using the information gathered in step 1, the performance monitor must be configured at each level of hierarchy. Some of the important fields that need to be configured are as follows:

Parameter	Value
MEASUREMENT_TYPE	BANDWIDTH, AVG_LATENCY
NUM_PERF_MONITOR	1 (Between master and interconnect)
NUM_TRANS_TYPE	3 (Random write, read and sequential read)
TOTAL_EXP_BW	1600 (MBPS)
BW_UNIT	MBPS
TIME_UNIT	Ns
EXP_BW	100, 300, 1200

Table 3 Configuration parameters for example system

A snippet of a part of the file for this configuration is given below:

Sr. No.	LEVEL	NUM OF PERF MON	NUM OF TRANS TYPE	TYPE NAME	LEAF MON ID	MEASUREMENT TYPE	EXPECTED BANDWIDTH	TOTAL EXP BW	BANDWIDTH UNIT	BANDWIDTH WINDOW
1	L1	1								
	L2		3					1600	MBps	
	L3			RANDOM_WRITE	0	BANDWIDTH	100			256
	L3			RANDOM_READ	1	BANDWIDTH	300			256
	L3			SEQ_READ	2	AVG_LATENCY +BANDWIDTH	1200			256

Figure 14 CSV file for example system

5) *Connection to environment components and creating performance transaction*:

The performance monitor now needs to be connected to any component or at different testbench hook-ups such that the user must be able to get the data like start time, end time and data byte of the passing transaction through the component to which the monitor connects. In our example scenario, the performance monitor is connected to the OCP monitor callback. The performance transaction is built inside the callback and passed to the performance monitor through the connecting port as shown in the figure below.

```
11  class ocp_master_monitor_callback extends project_ocp_master_monitor_callback;
12
13    // Instances of other members of the class
14    ...
15
16    /*! \This analysis port is used to send performance transaction to performance
17        Monitor for latency and bandwidth measurment  */
18    uvm_analysis_port #(perf_trans_c) perf_trans_ap;
19
20    /*!
21     * \brief new constructor
22     */
23     function new(string name = "ocp_master_monitor_callback", master_type_e master_id, uvm_component parent);
24       //Create other members
25       ...
26
27       //Create the perf analysis port
28       perf_trans_ap = new($sformatf("%s_perf_req",name),ap);
29     endfunction
30
31    /*!
32     * \dataflow_observed_port_cov
33     *  This function is used to get ocp transaction which is completed on interface
34     *  and for forming performance transaction and send to performance monitor through analysis port.
35     */
36     virtual function void dataflow_observed_port_cov(svt_ocp_monitor o_component, svt_ocp_dataflow_transaction o_dataflow);
37     begin
38       perf_trans_c perf_trans_inst;
39       perf_trans_inst    = new();
40
41       //Perf Trans ID. Decided from the traffic type.
42       // For Random Write : ID = 0, for Random Read : ID = 1, for Sequential Read : ID =2
43       perf_trans_inst.id = o_dataflow.tagid ;
44
45       // Latency Parameters
46       perf_trans_inst.req_lat_start_time =  latency_start_time_f(o_dataflow);
47       perf_trans_inst.req_lat_end_time   =  latency_end_time_f(o_dataflow);
48
49       // Bandwidth Parameters
50       perf_trans_inst.bw_start_time      =  latency_start_time_f(o_dataflow);
51       perf_trans_inst.bw_end_time        =  o_dataflow.svt_end_realtime   ;
52       perf_trans_inst.data_bytes         =  no_data_bytes_f(o_dataflow);
53
54     endfunction : dataflow_observed_port_cov
55
56       `uvm_info("PERF_TRANS",$sformatf("TRANS is \n %s",perf_trans_inst.sprint()),UVM_LOW);
57       perf_trans_ap.write(perf_trans_inst);
```

Figure 15 Performance transaction building and driving

The analysis port instance taken in line 18 is connected to performance monitor in top environment of the testbench. An API is provided in the top environment of the performance monitor for this purpose. This is shown in the figure given below (Here 'master' is the name of the master component given in CSV):

```
begin
  `uvm_info( "connect_perf_mon_f", $sformatf("Connecting %s port to the callback of OCP",master), UVM_HIGH);
  top_env_inst.connect_port_to_per_mon_f(master, perf_ocp_cb_inst.perf_trans_ap);
end
```

Figure 16 Connection between testbench environment and performance monitor

In line 57, this transaction is written into the analysis port, after which it will be available to the performance monitor for calculations.

In case these fields are not available to user through hook-ups, support for adapter class has been provided. This adapter class is connected to the performance monitor. User can write the logic to extract these fields by extending the adapter class and assign them to the performance transaction fields.

Following diagram gives an idea of set-up after connecting performance monitor to testbench:

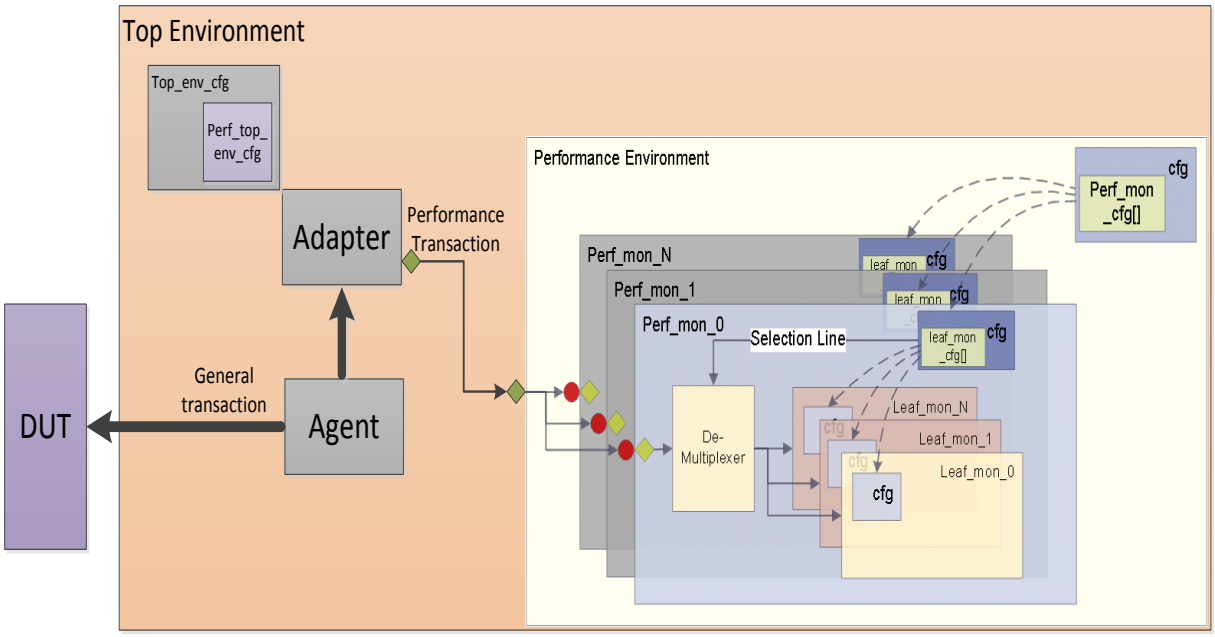


Figure 17 Performance monitor integration in environment

6) At this stage, the set-up for using performance monitor is done and user can run simulations to perform measurements.

7. Summary

We have used the mentioned performance monitor in three different projects for different clients so far. A comparison of the effort and time put in these projects vis-à-vis the effort put in performance verification of projects done without performance monitor vouch for the usefulness of the monitor.

In the SoC mentioned at the start of the paper, building the set-up for performance verification took as long as a month and another month was spent achieving the closure through testcase and debugging. A derivative of the same SoC was verified for performance using the performance monitor and the complete performance verification closure was achieved within 3 weeks duration, with lesser resources. This shows a clear saving in the person-hours put in for the closure.