Using a Generic Plug and Play Performance Monitor for SoC Verification

Dr. Ambar Sarkar*

Kaushal Modi Bhavin Patel

Janak Patel Ajay Tiwari

eInfochips

September 18, 2015

SNUG Austin

Agenda

Introduction

Challenges – Why Performance Monitor

Features

Architecture and Configuration

Measurements Supported

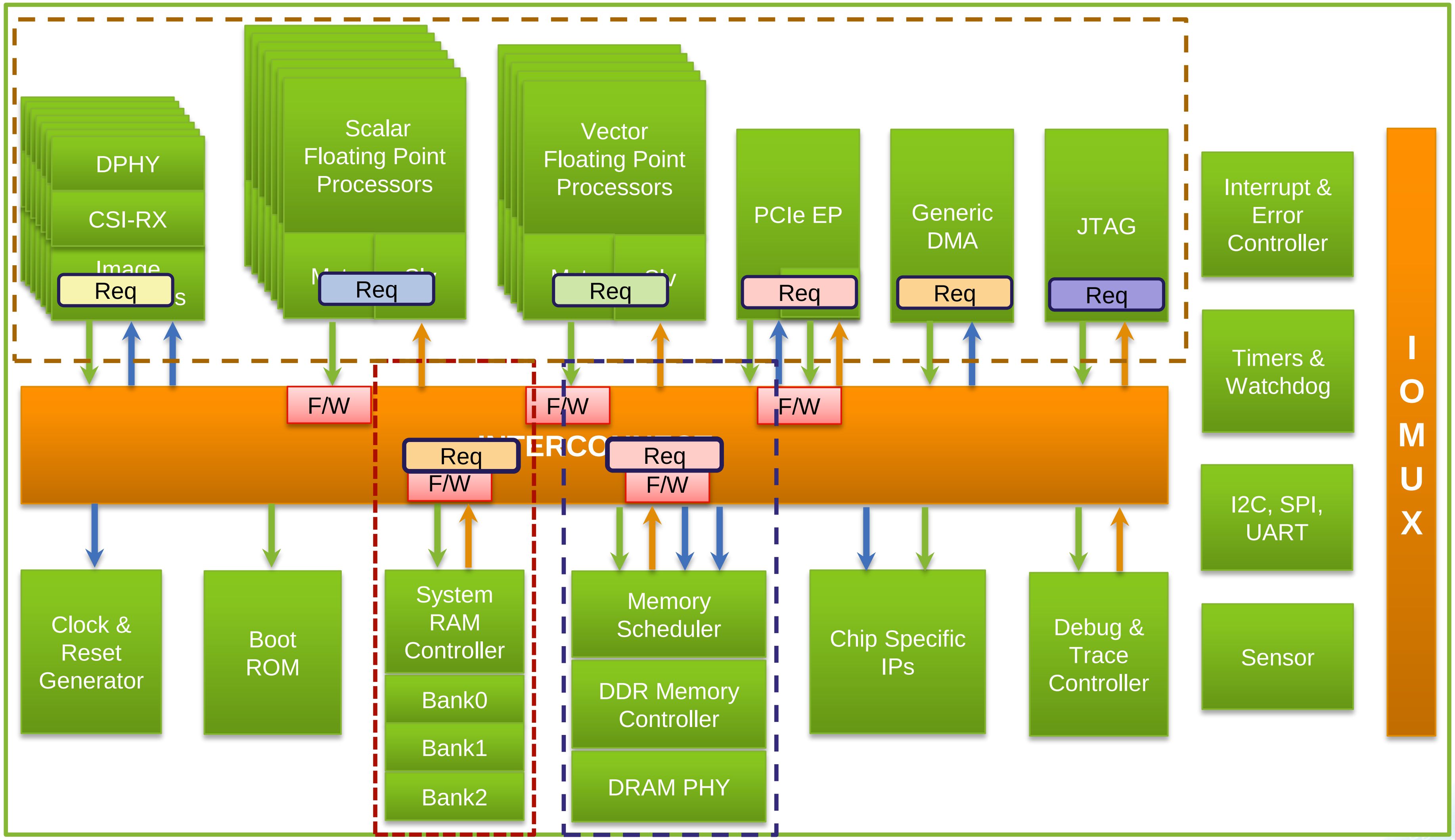
Performance Report

Steps for Monitor Usage

Introduction

- Two parameters of Performance Verification:
  - Latency : Time taken by data to reach from one point to another
  - Bandwidth : Amount of data transferred in a given time
- The parameters have to be measured and verified against an expected value for all paths of interest; the process known as performance verification
- Functional Verification alone cannot guarantee meeting of performance criteria
- With the complexity of SoCs increasing, importance of performance verification grows in verification cycle

Complex SoC: Diagram



Complex SoC: Perf Requirement

Paths	Traffic Type
Image Processor 0 -> DDR	128B WR 75% Sequential, 25% Random
Image Processor 1,2 -> DDR	128B WR 75% Sequential, 25% Random
Image Processor 3 to M -> DDR	128B WR 75% Sequential, 25% Random
Floating processor 0,1 -> DDR	[1] Sequential 128B RD [2] Random 32B RD [3] Random 32B WR
Floating processor 2,3 -> DDR	[1] Random 32B RD [2] Random 32B WR
Floating processor N -> System RAM	Sequential 128B RD
Other Floating processor -> DDR	[1] Random 32B RD [2] Random 32B WR
PCIe -> DDR	[1] Sequential 128B RD [2] Sequential 128B WR

Why Performance Monitor

Challenges in Performance Verification

- Multiple interfaces having different protocols
- Bunch of data paths of measurement
  - From single master to multiple slaves, multiple masters to multiple slaves, multiple masters to single slave
- Each path can have specific measurement requirements
  - Paths can have interleaved traffic pattern
  - Variable measurement window for different traffic patterns possible
  - Selective traffic measurement on particular interface
- Performance requirements may change during course of the project
- Possibility of addition of new paths in derivative SoC

Why Performance Monitor

Challenges in Performance Verification

- Different approaches used
  - Calculation in tests or callbacks
    - Testcase developer has to take care of calculations in tests
    - Testcases become too complex and lengthy
  - Reporting using post processing
    - Testcase developer has to print unique and appropriate messages
    - Requires development of extra script – to pick up the messages and give a summary of results
    - Overhead in terms of time and effort
    - In absence of a script, manual effort is required to check the success or failure of performance requirement – from log files
- No standard method – approach becomes dependent on individual

Why Performance Monitor
Solution

- A scope of reduction in time and effort put behind performance verification was realized
- A Performance Monitor was thought of – that could take care of the challenges
- Accordingly, this monitor had to be:
  - Protocol independent
  - Re-usable
  - Plug-and-play entity
  - Using result reporting that is easy to read, presentable and helpful in debugging

Features of Performance Monitor

Features

Highly Configurable
- Multiple paths and multiple traffic patterns supported
- Configurable units for latency (ns, us, ms) and bandwidth (KBps, MBps, GBps, Kbps, Mbps, Gbps)
- Configuration methods convenient, uncomplicated
- Configuration compliance check

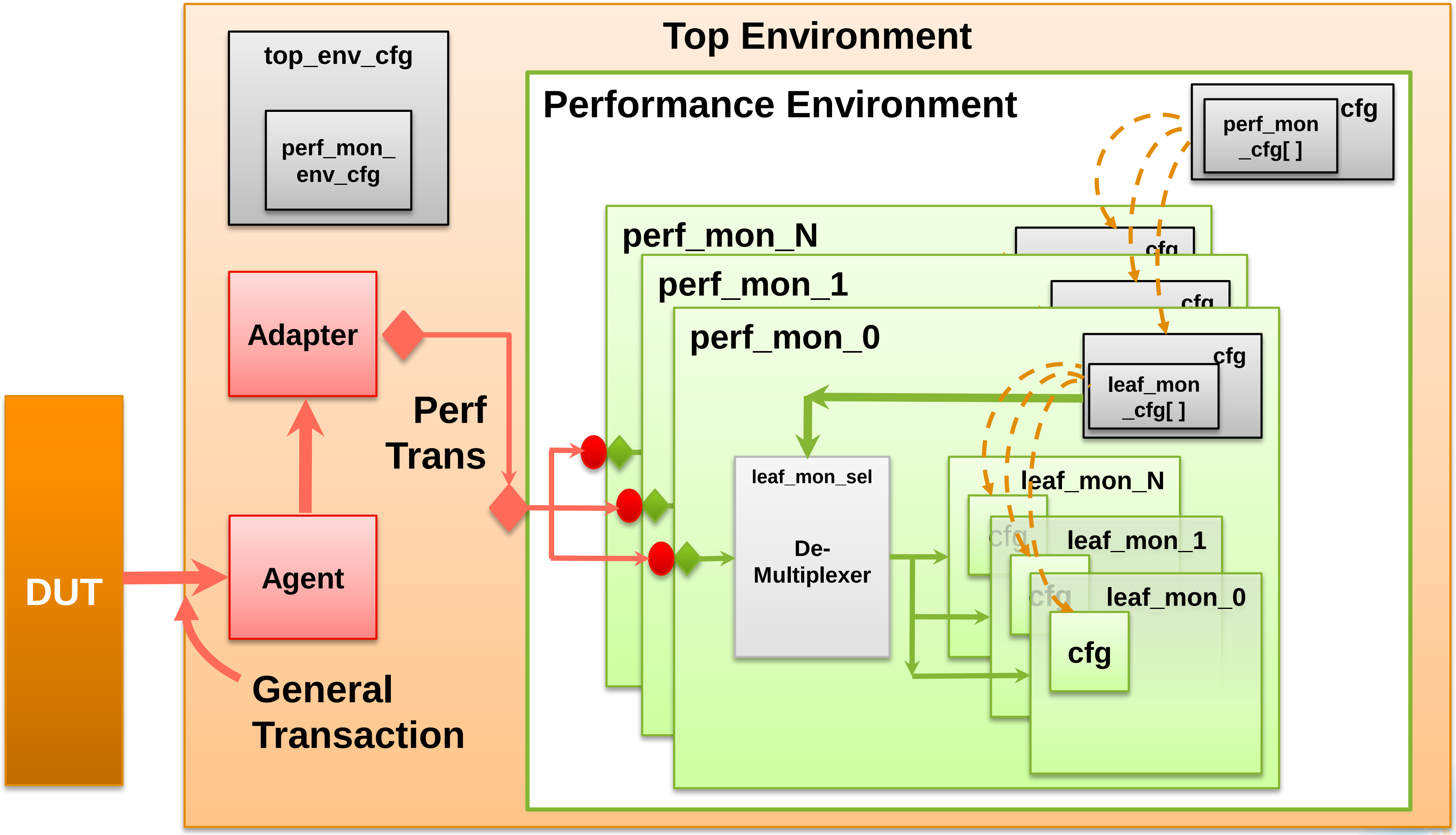
Types of Measurements
- Per transaction & Average latency
- Bandwidth
- Alternate bandwidth and latency window
- Cumulative bandwidth measurement for multiple traffic patterns
- Uniformity comparison between interfaces
- Root mean square (RMS) value calculation for latency, bandwidth and error values

Reporting
- Measurement report depending upon user defined verbose level
- Error reporting in case of incorrect result
- Run time reporting of measurements
- Trace file reporting for all the measurements

Miscellaneous
- Callback support to collect functional coverage
- API support to get performance statistics on-the-fly
- Measurement Enable/Disable Support

Architecture and Configuration of Performance Monitor

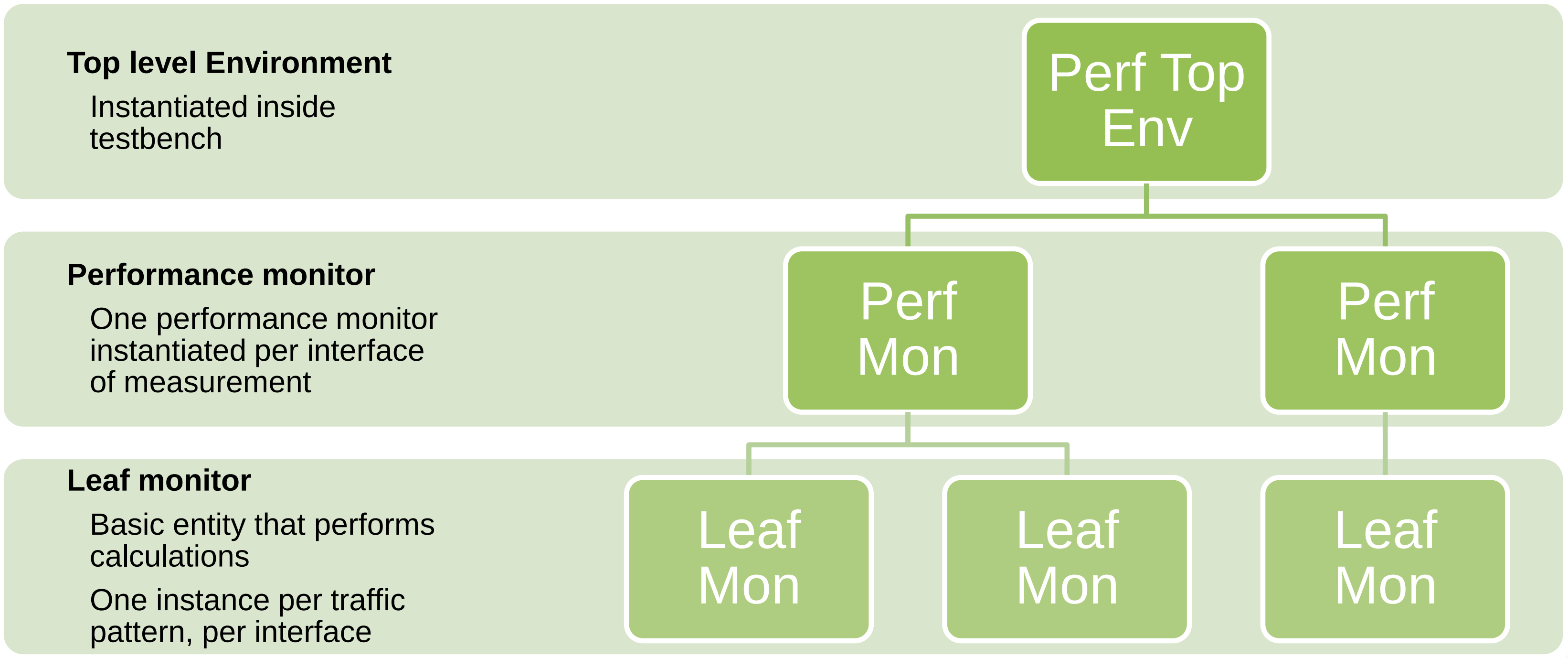
Architecture
Use Model



Architecture

Hierarchy of Monitor

- The measurement over different interfaces and traffic patterns is taken care of by the hierarchical structure of the performance monitor
- Each level has its own configuration parameters


Top level Environment
Instantiated inside testbench
Performance monitor
One performance monitor instantiated per interface of measurement
Leaf monitor
Basic entity that performs calculations
One instance per traffic pattern, per interface
Perf Top Env
Perf Mon
Perf Mon
Leaf Mon
Leaf Mon
Leaf Mon

Architecture
Performance Transaction

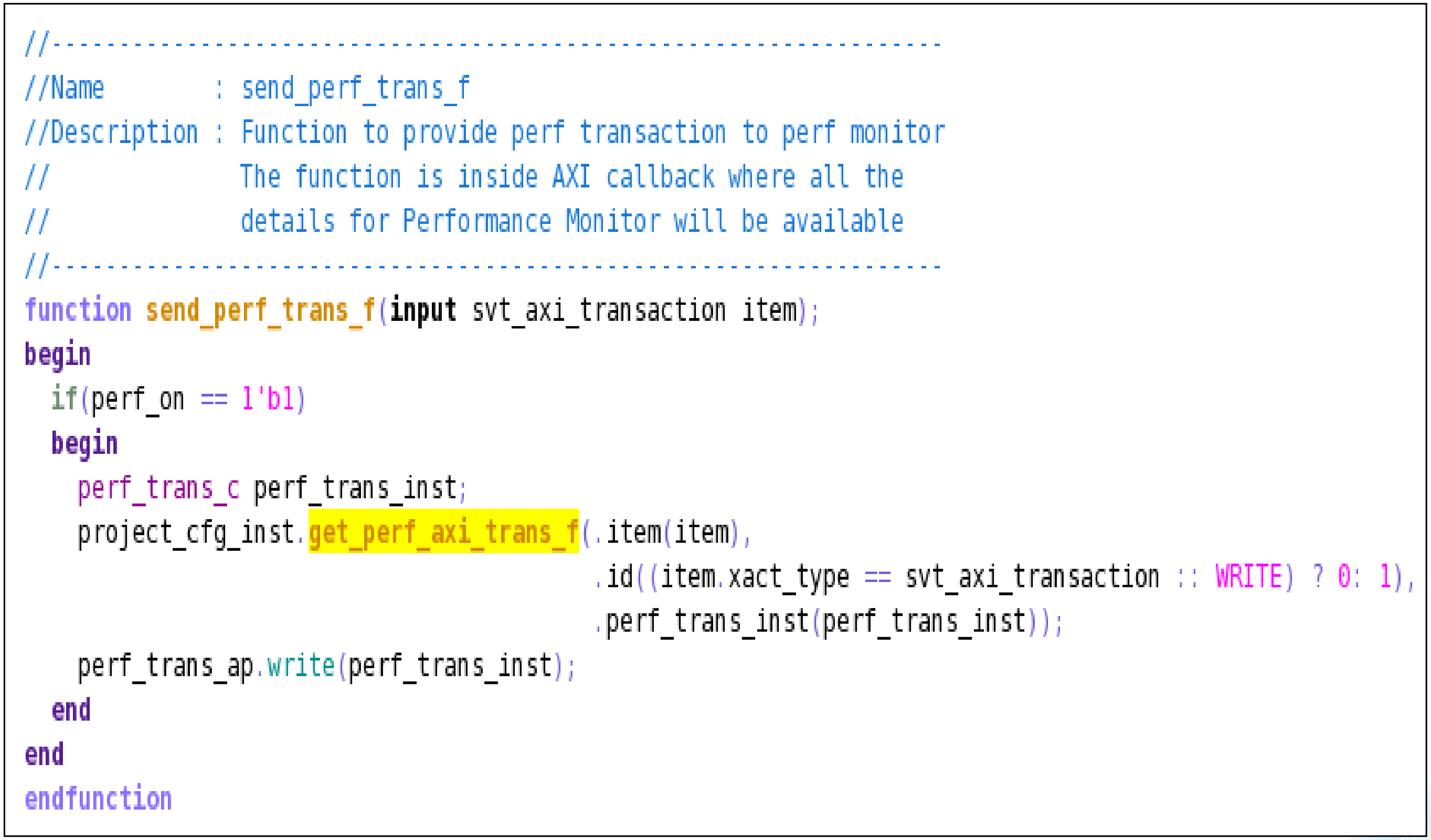
- Since the monitor can be used with different interfaces, a generic transaction type is needed – Performance Transaction. The fields of which are:

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

- User will update these fields upon the occurrence of required event
- This can be done inside any of the testbench components, where the required data will be available
- Passed to performance monitor through connecting port

Architecture

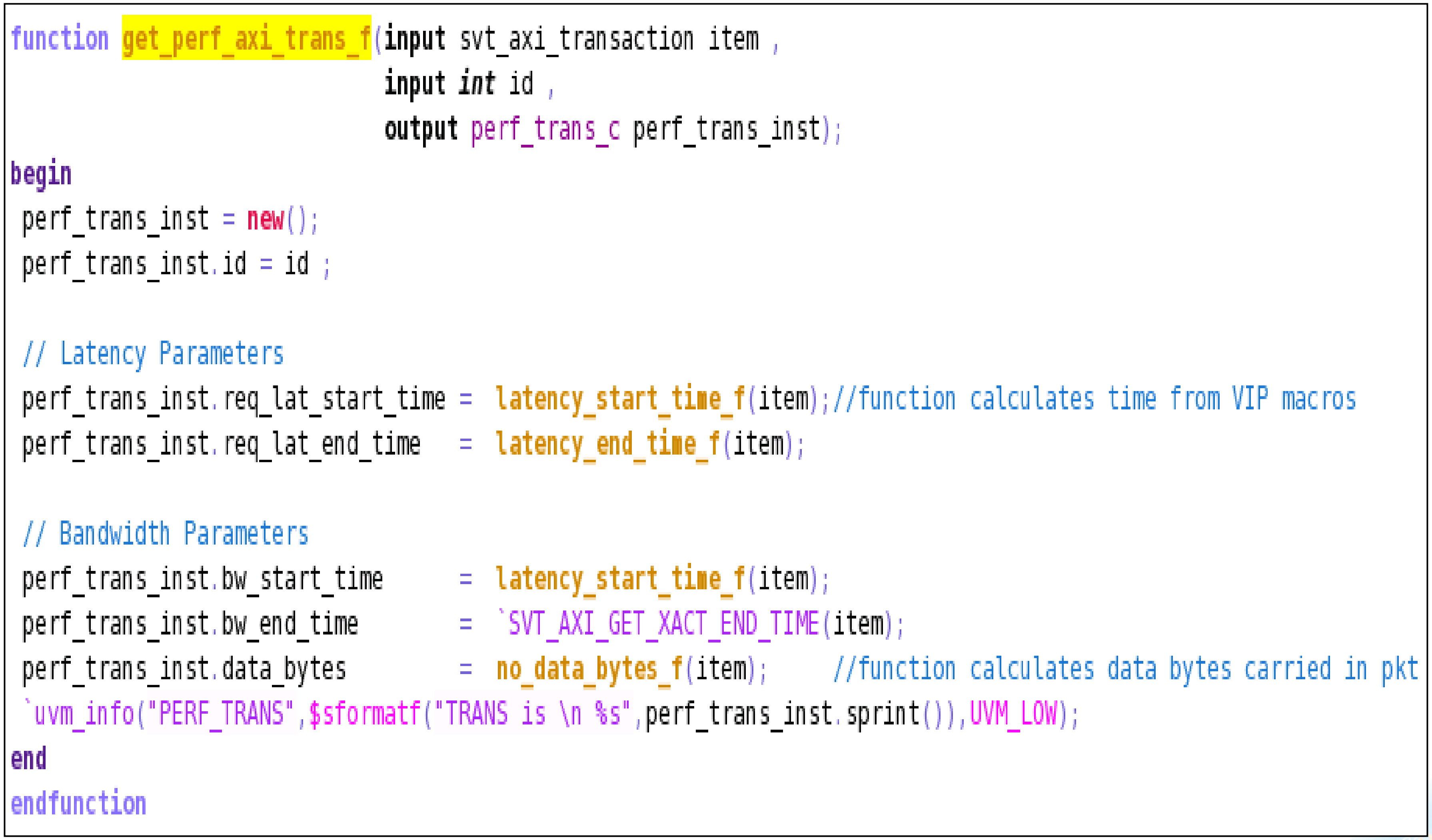
Example – Perf Trans Generation



```
//-----------------------------------------------------------------
//Name        : send_perf_trans_f
//Description : Function to provide perf transaction to perf monitor
//              The function is inside AXI callback where all the
//              details for Performance Monitor will be available
//-----------------------------------------------------------------
function send_perf_trans_f(input svt_axi_transaction item);
begin
  if(perf_on == 1'b1)
  begin
    perf_trans_c perf_trans_inst;
    project_cfg_inst.get_perf_axi_trans_f(.item(item),
                                          .id((item.xact_type == svt_axi_transaction :: WRITE) ? 0: 1),
                                          .perf_trans_inst(perf_trans_inst));
    perf_trans_ap.write(perf_trans_inst);
  end
end
endfunction
```

Architecture

Example – Perf Trans Generation



```
function get_perf_axi_trans_f(input svt_axi_transaction item ,
                              input int id ,
                              output perf_trans_c perf_trans_inst);
begin
 perf_trans_inst = new();
 perf_trans_inst.id = id ;

 // Latency Parameters
 perf_trans_inst.req_lat_start_time =  latency_start_time_f(item);//function calculates time from VIP macros
 perf_trans_inst.req_lat_end_time   =  latency_end_time_f(item);

 // Bandwidth Parameters
 perf_trans_inst.bw_start_time      =  latency_start_time_f(item);
 perf_trans_inst.bw_end_time        =  `SVT_AXI_GET_XACT_END_TIME(item);
 perf_trans_inst.data_bytes         =  no_data_bytes_f(item);     //function calculates data bytes carried in pkt
 `uvm_info("PERF_TRANS",$sformatf("TRANS is \n %s",perf_trans_inst.sprint()),UVM_LOW);
end
endfunction
```

Configuration

Methods of Configuration

- Configuration class based approach
  - Instance of the configuration class is taken inside testbench
- CSV file based approach
  - A pre-defined format of CSV file to configure 3 levels of hierarchy

SR. NO.	SEQUENCE NAME	LEVEL	NUM OF PERF MON	PERF MON NAME	NUM OF TRANS TYPE	LEAF MON ID	MEASUREMENT TYPE	EXPECTED LATENCY	EXPECTED PER_TRANS LATENCY	LATENCY UNIT	EXPECTED BANDWIDTH	BANDWIDTH UNIT
1	ocp_axi_read _write	L1	2									
		L2		OCP	2							
		L3				READ	AVG_LATENCY + BANDWIDTH	48	48	ns	100	MBps
		L3				WRITE	AVG_LATENCY + BANDWIDTH	48	48	ns	100	MBps
		L2		AXI	2							
		L3				READ	AVG_LATENCY + BANDWIDTH	48	48	ns	100	MBps
		L3				WRITE	AVG_LATENCY + BANDWIDTH	48	48	ns	100	MBps

Measurements Supported

Measurements Supported

- Per Transaction Latency measurement
  - Performed when traffic requirement is of guaranteed service
- Measurement over a number of transactions
  - Gives the value of bandwidth and latency over a user defined window of transactions – ‘setup’ and ‘hold’ transactions can be configured
- Measurement between two events
  - Used when measurement window has to be defined by the total bytes to be transferred, instead of total number of transactions.
- Measurement over an alternate window
  - To get the latency and bandwidth measurement over a window other than the configured window; helps to check the consistency of results

Measurements Supported

- Cumulative Bandwidth measurement
  - Helps if there is a requirement of different traffic patterns meeting a bandwidth requirement collectively
- Uniformity comparison
  - If the measured value for any interface has to be compared against the other interface, uniformity comparison can be used

Performance Result

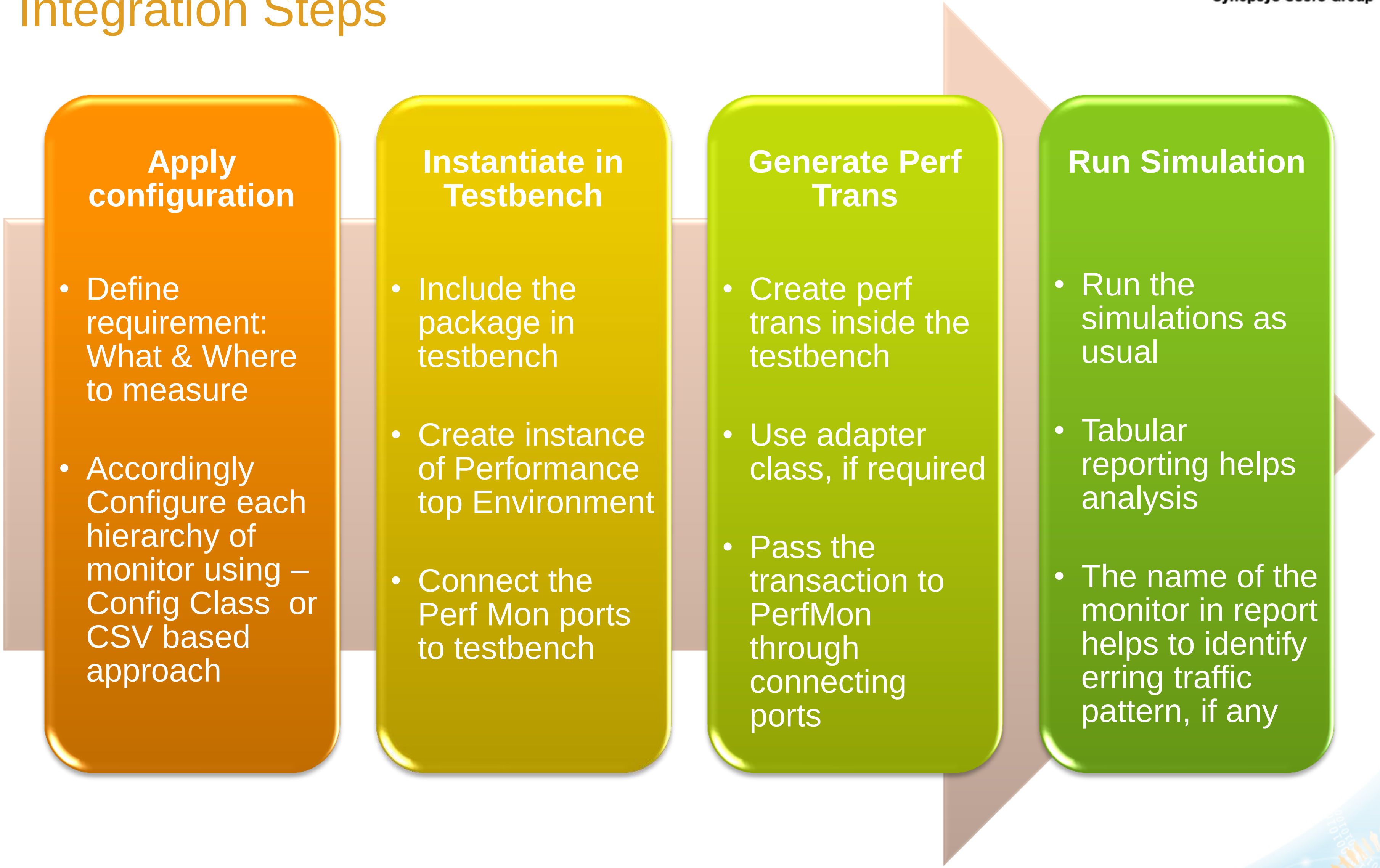
Result Reporting

- Two types of reporting:
  - Log file reporting
  - Trace file reporting
- Configurable report verbosity
  - Default Reporting : Summary of results reported in report_phase. This includes total windows, number of matching windows, average latency over all the windows among other details
  - Reporting Level 1 : Run-time reporting of results for each window, in addition to default reporting
  - Reporting Level 2 : Run-time reporting of results for each transaction, in addition to default and level 1 reporting
  - Debug Reporting : Debug messages for all the relevant information, apart from above mentioned details

Steps for Performance Monitor Usage

Steps for usage

Integration Steps



Apply configuration

- Define requirement: What & Where to measure
- Accordingly Configure each hierarchy of monitor using – Config Class or CSV based approach

Instantiate in Testbench

- Include the package in testbench
- Create instance of Performance top Environment
- Connect the Perf Mon ports to testbench

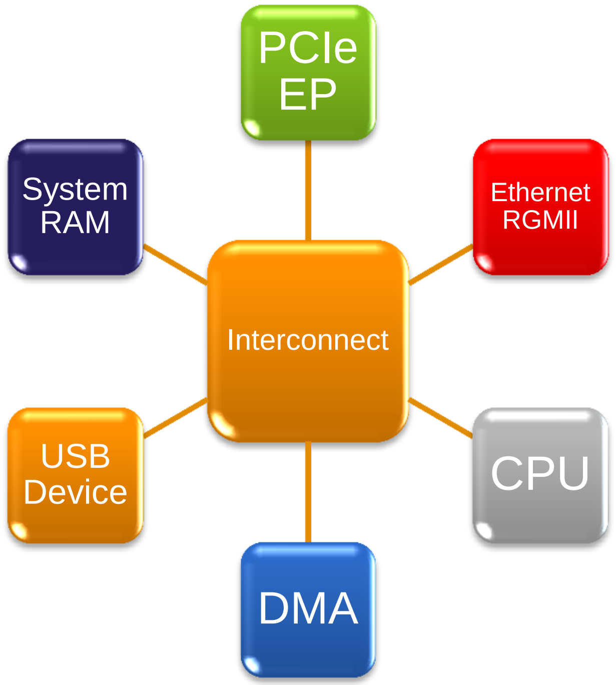
Generate Perf Trans

- Create perf trans inside the testbench
- Use adapter class, if required
- Pass the transaction to PerfMon through connecting ports

Run Simulation

- Run the simulations as usual
- Tabular reporting helps analysis
- The name of the monitor in report helps to identify erring traffic pattern, if any

SoC Performance Requirement



Scenario1: **PCIe <-> Ethernet**	**Start Event:** Start of Trans 1 **End Event:** End of Trans 512
Scenario2: **DMA enabled traffic**	**Start Event:** DMA GO **End Event:** DMA DONE

Performance Requirements

USB	Bulk In & Bulk Out
PCIe	Descriptor Mem Read
	Descriptor Mem Write
	TX Eth Packet Mem Write, Avg. Latency
	RX Eth Packet Mem Write - 980Mbps, Avg. Latency 3us
Ethernet	TX Packet from RAM
	TX Packet from PCIe, Avg. Latency
	RX Packet from RAM
	RX Packet from PCIe, Avg. Latency
CPU	PCIe Read/Write
	RAM Random Read/Write
	RAM Sequential Read/Write
DMA	TX Descriptor Read/Write
	RX Descriptor Read/Write
	RX Packet Read/Write

Summary

- The monitor is used in three different projects for different clients so far
- With ad-hoc methods, for a complex SoC with 20+ masters and 80+ slaves, building the set-up for performance verification took as long as a month. Apart from this, another month was spent achieving closure through testcase and debugging
- A derivative of same SoC was verified using Performance Monitor and complete performance verification closure was achieved in 3 weeks duration, with lesser resources
- This demonstrates a saving in the man-hours for the closure

Thank You