Standardizing VMM Performance Analyzer Implementations across VMM Testbenches

Michael Kappelhøj Andersen
Jacob Sander Andersen
Peter Jensen

SyoSil
Systems on Silicon

Himmelev Bygade 53, 4000 Roskilde, Denmark
{michael, jacob, peter}@syosil.com

www.syosil.com

ABSTRACT

The VMM Performance Analyzer application provides the user with a standard mechanism for capturing performance data, but it provides no common usage model for enabling it inside a VMM test bench, nor how to present the captured performance results.

This paper introduces a generic, uniform approach for implementing the VMM Performance Analyzer application in a VMM test bench through definition of a generic use model for enabling the application and presentation of performance results.

The use model defines a set of guide lines, which allows 100% automatic generation of textual reports and graphs for the performance results.

Table of Contents

Table of Figures

Table of Tables

1 Introduction

The VMM performance Analyzer is a utility for measuring RTL cycle performance in a VMM test bench. The performance metrics are used for validating the RTL performance against the specifications, and as foundation in a design optimization process.

This paper is an offspring of using the VMM Performance Analyzer on an internal example in order to be able to deploy it on industrial projects. This work discovered the need for a generic approach for implementing the VMM Performance Analyzer application across VMM test benches.

Since the performance results stored in the VMM Performance Analyzer database can not be read back and presented by the test bench, the need for a post simulation processing script for automatically presenting performance metrics became obvious.

2 The VMM Performance Analyzer

The VMM Performance Analyzer [1] supplies a set of classes to capture and store data used in performance analysis. Data can be captured from any shared resource, transaction or hardware structure depending on the capabilities of the verification environment and of the transactors it contains.

Captured data are stored in tenures. A tenure is defined by any activity with well defined starting and ending point. The basic tenure captures the data shown in Table 1

Tenure data	Description
tenureID	Automatically added unique id (integer)
initiatorID	Who initiated it, e.g. input port (integer)
targetID	Who was the target, e.g. output port (integer)
start	Tenure start time (bigint)
end	Tenure end time (bigint)
active	Active time, end time minus start time minus suspend time (bigint)
abort	Tenure status, completed or aborted (tinyint)
user schema	User defined columns of any type

Table 1 Tenure data

When a tenure is ended it is added to a database structure and is no longer accessible from the test bench. Currently, only SQLite [2] databases are supported. In the rest of the document it will be denoted as SQL. A simple implementation is shown in Figure 1.

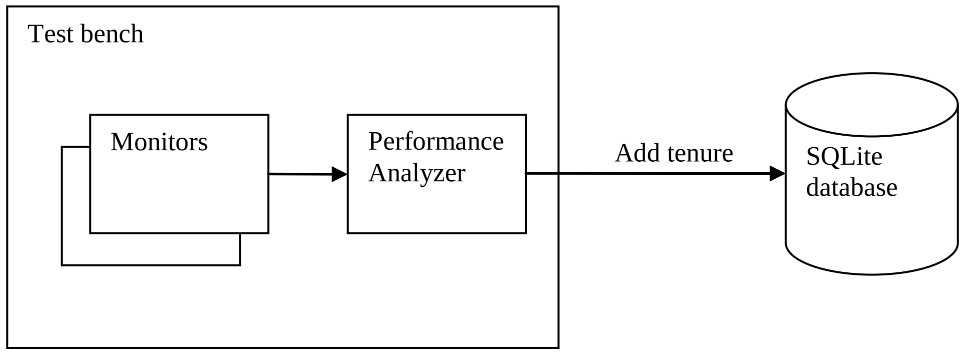


Figure 1 VMM Performance Analyzer - simple implementation

Please refer to [1] for a complete description of the VMM Performance Analyzer.

3 Extending The VMM Performance Analyzer Framework

The VMM Performance Analyzer provides a powerful way to collect performance data, but the raw SQL data needs to be processed in order to get a meaningful result. A way to do this could be to import SQL data to a spreadsheet. This approach is good; if the performance strategy is not yet defined or if different strategies are to be tested.

However, analyzing data in a spreadsheet can be very time consuming, and is not suitable for an automated flow. Performance data from nightly or weekly regressions needs to be processed automatically, as performance metrics usually is part of the sign-off criteria.

The aim here is to post process the performance data 100% automatically. This calls for a standardized way to report performance data from the SystemVerilog test cases, such that a post simulation processing script can generate performance reports and interesting charts.

4 Extended Framework Overview

The main goal of the extended framework is to generate a standardized performance report in a simple form, but with enough information for it to be self containing. The framework has been developed with emphasis on usability; it shall be easy to use and re-use in different situations with as little effort as possible.

The extended framework can be divided into three main areas:

- Generic standard SQL data structure
- General SystemVerilog API
- Perl post simulation processing script

The need for a generic standard SQL data structure is obvious, as this is the foundation for creating a general API and to be able to use a single post simulation script for all sorts of performance metrics. The suggested SQL data structure is described in section 6.1. The structure will cover most use cases, but can be extended or modified to specific use.

The SV API is developed to raise the level of abstraction by creating the specific SQL data structure with only a few API calls. Tables for holding performance metrics are created, and a description in form of title, headers and column names is defined for use in the report. The SV API is described in section 6.2.

Defining the actually performance metrics has to be done by SQL statements. Tenures are stored in the SQL database and can not be read back to the SystemVerilog test bench for processing. Calculation of performance metrics is done by executing SQL statements after simulation has ended and all tenures are captured.

Figure 2 shows how the extend framework interacts with the VMM Performance Analyzer.

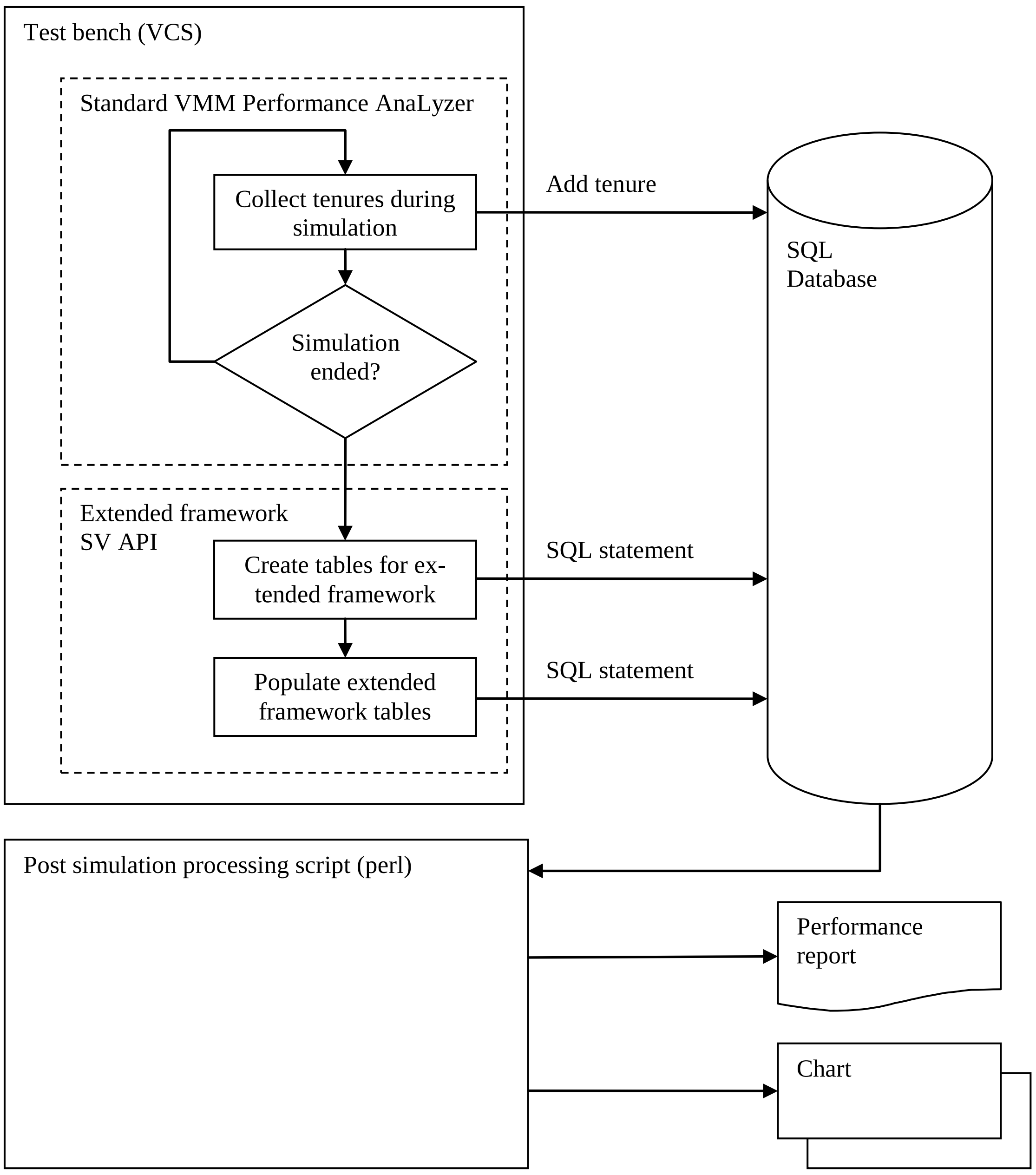

Test bench (VCS)
Standard VMM Performance AnaLyzer
Collect tenures during simulation
Simulation ended?
Add tenure
SQL Database
Extended framework SV API
Create tables for extended framework
SQL statement
Populate extended framework tables
SQL statement
Post simulation processing script (perl)
Performance report
Chart

Figure 2 Performance Analyzer Flow

How performance metrics are calculated is specific to the device tested. A few exceptions are simple metrics only using the default tenure data; namely minimum, maximum and average duration, and percentage of utilization. These are also calculated by the VMM Performance Analyzer. Examples of SQL statements are shown in 6.2.1.

The performance report structure is defined as shown below, and of course reflected in the SQL data structure.

Performance report title
Result data table header

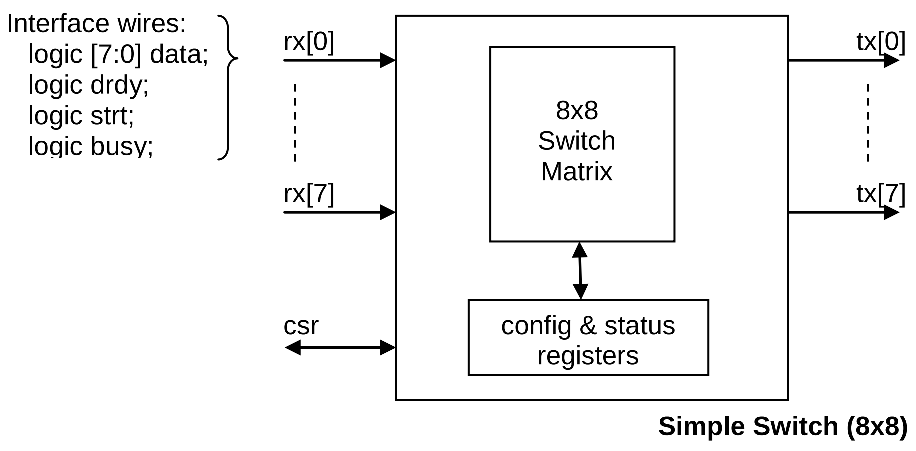
Column name 1	Column name 2	Column name 3	Column name n
data	data	data	data
data	data	data	data

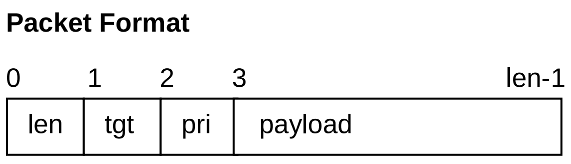
Any number of result data tables can be included.

Datasets for Gnuplot [3] is converted to graphics and will not be part of the textual performance report.

5 Example Application

In order to clarify the use and the implementation of the extended VMM Performance Analyzer framework, a small example has been created. The "Simple Switch" model is, as the name implies, a simple packet based switch with 8 input ports and 8 output ports, as shown below.





Field Name	Description
len	Length of packet (4-255)
tgt	Target port
pri	Packet priority
payload	Byte payload, 1-252 bytes

Figure 3 Simple Switch Diagram

The rx and tx switch ports receive/transmit byte streams, with notification of start bytes, and allows backpressure. Configuration and status registers allow setting port status (up/down), as well as gathering packet statistics per port at run-time. Packets sent to non-existing ports, as well as to inactive (down) ports, are dropped by the switch.

For the VMM test bench for the Simple Switch, a VMM data class has been defined as follows. Note that the len field is calculated by the `byte_pack()` method, once the byte stream is computed.

```
class cl_spp_data extends vmm_data;
   …
  rand logic [7:0] tgt;       // Target port
  rand logic [7:0] pri;       // Priority
  rand logic [7:0] payload[]; // Payload 1-252 bytes
  constraint co_payload_size {
    payload.size() >= 1;
    payload.size() <= 252;
  }
   …
endclass
```

In addition to the data class, bus functional models (rx/tx), a protocol monitor and other auxiliary transactors exist to make up a complete piece of Verification IP for the Simple Switch protocol.

5.1. Performance Analyzer Implementation

Looking at the data class for the Simple Switch as defined above, it is apparent that payload size will be interesting in a performance perspective. Therefore a `packet_size` column is added to the default tenure. This is defined when the VMM Performance Analyzer is created as follows:

```
// Create VMM perf analyser with no maximum initiator, target nor tenure
// User schema packet_size added to default teneure
switch_perf = new("Simple_Switch", db, 0, 0, 0, "packet_size INT");
```

The implementation around the VMM Performance Analyzer is depicted in Figure 3. The bus monitors will capture transactions at the Rx and Tx ports, and send them via channels to the Performance Monitor. When an inbound transaction is received, a tenure is created by calling `switch_perf.start_tenure()` with initiatorID, targetID and the transaction. The tenure is completed when a matching outbound transaction is received by calling `switch_perf.end_tenure()` and the payload size is added.

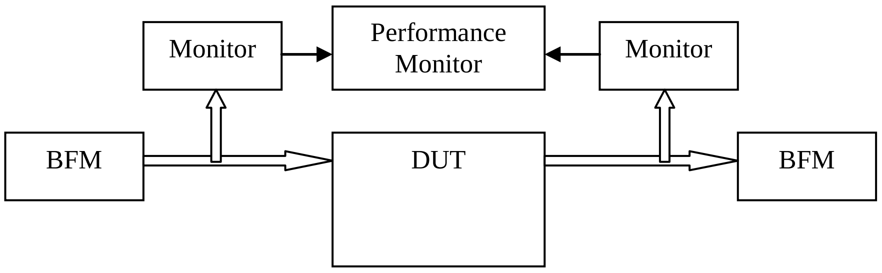


Figure 4 Performance Monitor Implementation

6 Extended Framework Implementation

6.1. SQL Data Structure

The SQL data structure for the extended framework is defined as a set of nested tables. A generic table structure is necessary in order create a script that can be used with any number of performance entries and across different test benches.

The main handle for the table structure is the `performance_report` table. The table can hold any number of entries pointing to either a `result_data` table or a `chart_data` table. The `_header` fields are used to present the result or plot with a description in the report. `merge` indicates if and how `result_data` is to be merged across multiple test runs.

Table::performance_report						
result_header	result_data	chart_header	chart_data	xlabel	ylabel	merge

The `performance_report_title` table simply holds the title of the performance report.

Table:: performance_report_title
report_title

A `result_data` table holds a user defined table structure that is specific for results generated by the test bench.

Table:: <result_data>
<Result specific columns>

A `<result_data>_header` table is used to associate a column name in a result_data table with a column header used when generating the performance report.

<result_data>_header	
column_name	header

A `<chart_data>` table contains any number plot data to be created in same chart.

Table:: <chart_data>	
plot_lable	dataset

The `<dataset>` table simply contains x/y data for a plot.

Table::<dataset>	
x_data	y_data

6.2. SystemVerilog API

The SystemVerilog API is provided for generating and partly populating the standard SQL data structure. This will enable the verification engineer to generate consistent performance reports across VMM test benches without detailed knowledge of SQL table structures.

The actual result data and plot dataset cannot be expressed generically, as this depends on what kind of performance data is defined for the specific test case.

API definition	Options

function new (
  vmm_log log,
  vmm_sql_db db,
  string vmm_perf_analyser_name,
  string report_title
);
 	
Description: Create the standard SQL table structure for performance reporting.
	• vmm_log log:
	vmm_log object
	• vmm_sql_db db:
	Database object
	• string vmm_perf_analyser_name:
	Name of the vmm_perf_analyser instance
	• string report_title:
	Performance report title

task create_result_data_table(
  string result_header,
  string result_data_table,
  string scheme[][],
  string merge = ""
);
 	
Description: Create result data table with user defined scheme.
	• string result_header:
	Result data table header
	• string result_data_table:
	Result data table name
	• string scheme[][]:
	Array holding the relation between a column in the result data table and a column name
	• string merge = "no":
	Enum indicate if and how result data are to be merged with data from other simulation runs

task create_chart(
  string chart_header,
  string chart_data_table,
  string xlabel,
  string ylabel
);
 	
Description: Create a new empty chart with the given chart_header (main title in gnuplot).
	• string chart_header:
	Chart name to be shown in chart
	• string chart_data_table:
	Chart data table name
	• string xlabel:
	Text describing the x axis
	• string ylabel:
	Text describing the y axis

task add_plot(
  string chart_data_table,
  string plot_label,
  string plot_dataset
);
	• string chart_data_table:
	Chart data table name
	• string plot_label:
	Plot label to be shown in chart
	• string plot_dataset:
	Plot dataset table name

Description: Add plot to existing chart.	
task add_vmm_performance_analyser_report_data(); **Description**: Add standard vmm_performance_analyser report data to SQL (minimum, maximum and average duration and percentage of utilization).	-

Table 2 SystemVerilog API

The performance metrics produced by `add_vmm_performance_analyser_report_data()` is actually the same as produced by the VMM Performance Analyzer `report()` function. `report()` is calculating the result run-time and does not uses the SQL data. As the function only can print to stdout (or text file) and variables are local, the results must be re-calculated by SQL statements if needed in the performance report.

The API methods are used on the example test bench in the following way to create the basic structure as follows. User or test bench defined performance metrics is not yet added.

```
function cl_vmm_perf_simple_switch::report();

  cl_vmm_syo_perf_report perf_report;

  // Save collected performance data
  this.switch_perf.save_db();

  // Create performance report structure as expected by post sim script
  perf_report = new(this.log, this.db, "Simple_Switch",
                    "Performance report for Simple Switch example");

  // Create result data table
  perf_report.create_result_data_table(.result_header("Accumulated packet size for output ports"),
                                       .result_data_table("result_data_output_pkt"),
                                       .scheme('{'{"output_port","Output port"},
                                                 '{"acc_pkt","Accumulated packet size (bytes)"},
                                                 '{"bandwidth","bits/s"}}));
  // Create table for coverage data
  perf_report.create_result_data_table(.result_header("Packet delay distribution for input ports"),
                                       .result_data_table("result_data_input_pkt_distribution"),
                                       .scheme('{'{"input_port","Input port"},
                                                 '{"small_delay","Small delay (1:100000)"},
                                                 '{"medium_delay","Medium delay (100001:200000)"},
                                                 '{"large_delay","Large delay (200001:2000000)"}}),
                                       .merge("sum"));

  // Create new chart
  perf_report.create_chart(.chart_header("Accumulated packet size over time for input ports"),
                           .chart_data_table("apst")),
                           .xlabel("Simulation time (ns)"),
                           .ylabel("Accumulated Packet size (bytes)"));

  // Add plot to chart
  perf_report.add_plot(.chart_data_table("apst"),
                       .plot_label("Port 0"),
                       .plot_dataset("port0"));

    // Add standard vmm_performance_analyser report data to SQL.
  perf_report.add_vmm_performance_analyser_report_data();

  this.db.commit();
endfunction
```

6.2.1. Generating Performance Metrics

In section 6.2 it is shown how to create a basic structure for holding performance metrics. To generate and insert desired performance metrics, basic SQL statement knowledge is mandatory. Tenures collected and stored in the database can currently not be read back to the test bench (VMM 1.2). The following examples show how tenure data can be processed and placed in the performance report SQL data structure.

All packets sent to a specific port are summarized and the utilized bandwidth for the port is calculated. In this example, output port 2 is chosen. INSERT INTO will create a new row in the `result_data_output_pkt` table. `Simple_switch` is the name of the table where all tenures are stored. `targetID` is the id of the packet destination port. `packet_size` is the name of the user defined column in the tenure.

```
this.db.statement("\
  INSERT INTO result_data_output_pkt (output_port, acc_pkt, bandwidth) \
  SELECT 2, SUM(packet_size), SUM(packet_size) * 8 * 1000000000 /(MAX(end)-MIN(start)) \
  FROM Simple_Switch \
  WHERE targetID = 2");
```

It can be interesting to see the packet size distribution at the input ports. Normally this would be implemented using functional coverage but here it is also used for a merge example. UPDATE will overwrite the `small_pkt` field in `result_data_input_pkt_distribution` where input port equals `i`.

```
this.db.statement($psprintf("UPDATE result_data_input_pkt_distribution \
                             SET small_pkt = (SELECT COUNT(start) \
                               FROM Simple_Switch \
                               WHERE initiatorID = %0d AND (packet_size) < 20) \
                             WHERE input_port = %0d", i, i));
```

Plot data can simply be extracted from a table and be inserted into the plot dataset. Here input packet size is accumulated in a number of timesteps.

```
 for (int i = 1; i < number_of_timesteps + 1; i++) begin
   this.db.statement($psprintf("INSERT INTO port%0d (x_data, y_data) \
                                SELECT %0d, sum(packet_size) \
                                FROM Simple_Switch \
                                WHERE initiatorID = %0d AND start < %0d",
                                port, timestep * i, port, timestep * i ));
end
```

6.3. Automated Report Generation

The defined SQL structure makes it possible to generate uniform textual report for any kind of performance data. When test cases are executed, the VMM performance analyzer SQLite database is written to file. The database contains both the VMM performance analyzer defined tables (e.g. collected tenures) and the tables generated by use of the extended framework.

A Perl script (`perf_report.pl`) is used to post process the database. The `performance_report` table is traversed to generate textual performance report and charts.

Below is an example of textual report and chart, Figure 5.

```
Performance report for Simple Switch example

Accumulated packet size for output ports
Output port     Accumulated packet size (bytes)     bits/s
0               -                                   -
1               -                                   -
2               9762                                4720046
3               5596                                2907201
4               8032                                3802355
5               -                                   -
6               6265                                2958537
7               8436                                3824766


Accumulated packet size for input ports
Input port     Accumulated packet size (bytes)     bits/s
0              -                                   -
1              -                                   -
2              8065                                3824812
3              8139                                3845863
4              7436                                3371379
5              -                                   -
6              7243                                3672501
7              7208                                3316653


Packet size distribution for input ports
Input port     Small pkt (<20 bytes)     Medium pkt (20:200 bytes)     Large pkt (>200 bytes)
0              0                         0                             0
1              0                         0                             0
2              6                         42                            15
3              3                         49                            13
4              2                         42                            15
5              0                         0                             0
6              3                         47                            7
7              2                         47                            10


Standard vmm_performance_analyser report, Active tenures
Min Tenure Duration      Avg Tenure Duration      Max Tenure Duration      Utilization
10000                    277873                   1064800                  432


Standard vmm_performance_analyser report, Aborted tenures
Min Tenure Duration      Avg Tenure Duration      Max Tenure Duration      Utilization
0                        0                        0                        0
```

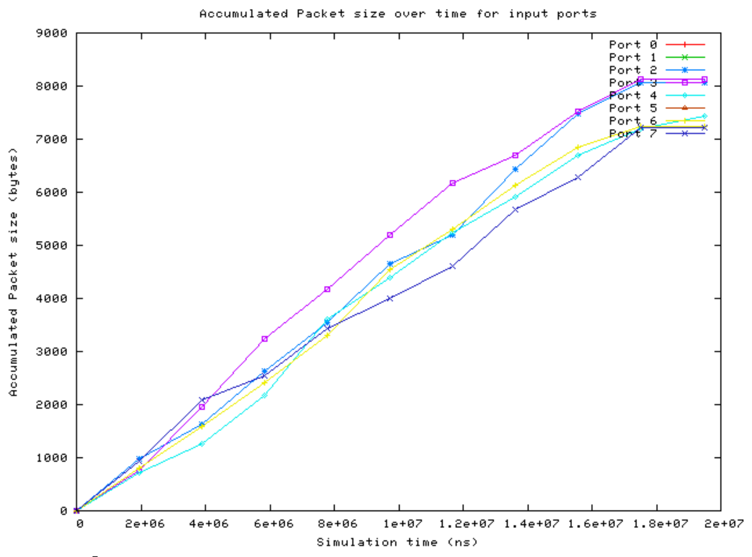


Figure 5 Gnuplot example

6.3.1. Merge Performance Metrics

When creating a result data table, it can be marked if the data can be merged with other test runs, and what scheme shall be used. Table `result_data_input_pkt_distribution` in section 6.2 is enabled to be summarized. In this simple scheme the merge script (`merge_perf.pl`) assumes that the first column holds category (here input data), and the succeeding columns holds data that are to be summarized.

Running the test with a different seed gives a new result for the table:

```
Packet size distribution for input ports
Input port     Small pkt (<20 bytes)     Medium pkt (20:200 bytes)     Large pkt (>200 bytes)
0              0                         0                             0
1              4                         31                            9
2              0                         48                            7
3              1                         42                            8
4              0                         0                             0
5              2                         31                            11
6              0                         0                             0
7              0                         0                             0
```

Running the merge scripts on the two SQL databases will create a third database with a `performance_report` table only containing the merged `result_data_input_pkt_distribution` table.

The performance report script can then be used on the new database resulting in:

```
Performance report for Simple Switch example

MERGED - Packet size distribution for input ports
Input port     Small pkt (<20 bytes)     Medium pkt (20:200 bytes)     Large pkt (>200 bytes)
0              0                         0                             0
1              4                         31                            9
2              6                         90                            22
3              4                         91                            21
4              2                         42                            15
5              2                         31                            11
6              3                         47                            7
7              2                         47                            10
```

7 Recommendations and Future Work

Deploying the VMM Performance Monitor and the extended framework to a VMM test bench is done in 5 steps. These are common for any kind of designs where performance need to be measured:

- Define target performance metrics for design
- Define tenure user schema for collection relevant data
- Find appropriate place in test bench (e.g. transactor, monitor, callback) to collect tenures
- Define extended framework table structure by using the SV API
- Write SQL statements to populate result and chart tables

Section 6.3.1 describes a simple sum of performance metrics from multiple test cases. Such a simple merge scheme will far from always be sufficient. Here it will be necessary to write specific merge script in your preferred language.

Running tests that generates a large number of SQL entries will have a negative impact on the memory consumption. This problem can be addressed by saving the database regularly during simulation. This will assumingly free up memory. The VMM Performance Analyzer provides the `save_db()` task for this purpose.

7.1. In the Future

The extended framework provides a basic structure to generate a simple performance report and charts. This can easily be extended to more general or specific use. The most apparent are:

- The ability of generating different chart types e.g. histogram. Currently only linepoints are supported
- Insert pass/fail criteria that automatically can be checked by the post processing script

Currently, the VCS / SQLite API provided for the VMM Performance Analyzer has no mechanism for retrieving the results from a database query. If/when this becomes possible the textual performance report can be generated directly by the test bench, instead of using post processing script. The need for the extended framework SQL data structure and SV API will remain, but the post processing script can be reduced to generate charts.

8 Conclusions

This paper has shown and facilitated a way to use the VMM Performance Analyzer in a generic uniform manner. With use of the proposed standardized SQL table structure, SystemVerilog API and post simulation processing Perl script it is possible to auto generate a comprehensive performance report with only little knowledge of SQL.

This approach enables easy integration and use of the VMM Performance Analyzer framework in both small and large verification projects and increases the overall verification quality.

The SystemVerilog API and the post simulation processing script are available for download. Please contact the authors for details.

9 References

[1] VMM Performance Analyzer User Guide. VMM Performance Analyzer Version 1.1. December 2009 http://vmmcentral.com/pdfs/vmm_perf_guide.pdf
[2] Description of SQLite can be found at http://www.sqlite.org
[3] Description of Gnuplot can be found at http://www.gnuplot.info

10 Abbreviations

API	Application Programming Interface
BFM	Bus Functional Model
DUT	Device Under Test
RTL	Register Transfer Level
SQL	Structured Query Language
SV	SystemVerilog
VMM	Verification Methodology Manual