Traffic Profiling and Performance Instrumentation For On-Chip Interconnects

Mark Peryer
Mentor Graphics (UK) Ltd
Rivergate, London Road, Newbury,
Berkshire, RG14 2QB, U.K.

Bruce Mathewson
ARM Ltd.
110 Fulbourn Road, Cambridge,
Cambridgeshire, CB1 9NJ, U.K.

***Abstract*—On-chip bus interconnect fabrics have become critical sub-systems in SoC platforms. Not only do they need to be functionally correct, but they also need to deliver the performance demanded by user applications in end products such as mobile platforms. The validation process for an interconnect in a simulation or emulation environment requires the generation of stimulus corresponding to a realistic use case. The typical brute force approach to this involves running applications software on processor cores that interact with the SoC RTL and checking that the overall design performance is satisfactory. An alternative approach is to validate the bus fabric stand-alone before integration with the rest of the SoC design using a verification environment that uses VIP to model the behavior of design IP. In order to create realistic traffic scenarios, the behavior of the different IP bus masters is described using traffic profiles. The performance of the interconnect is measured using instrumentation which monitors the bus activity. This paper describes a proposal for the specification of bus master traffic profiles and system level traffic scenarios, together with the definition of performance metrics that need to be instrumented to ensure that an interconnect is meeting its performance targets.**

Keywords—SoC Interconnect; Validation; Performance; Bandwidth; Latency; traffic profiles; traffic scenarios; stimulus abstraction;instrumentation

I. INTRODUCTION AND OVERVIEW

Nearly all Systems on a Chip (SoCs) are implemented using an internal architecture where at least one CPU core is connected to a number of hardware resources via an internal interconnect based on a standard on-chip bus protocol such as AMBA. Most design IP is designed with one or more on-chip bus interface sockets so that it can easily be integrated into a SoC by connecting it to one or more counter-part sockets on the SoCs interconnect fabric. The success of this approach has meant that designs that started off as relatively simple specialized microprocessors, with a CPU core and a few peripherals, have now evolved into complex flexible computing platforms. At the same time the software that runs on the SoC has evolved from closed firmware supporting limited device functionality to multiple operating systems capable of running third party applications. The fact that the SoC has effectively become an open computing platform means that it is becoming increasingly important to validate that the platform will be able to support multiple use models. Each of these scenarios will use different combinations of the on-chip hardware resources and each will place performance demands on the interconnect at the core of the SoC. If the interconnect does not deliver on bandwidth in these different situations, then the user experience will suffer and ultimately will cause the end product to fail in the market place.

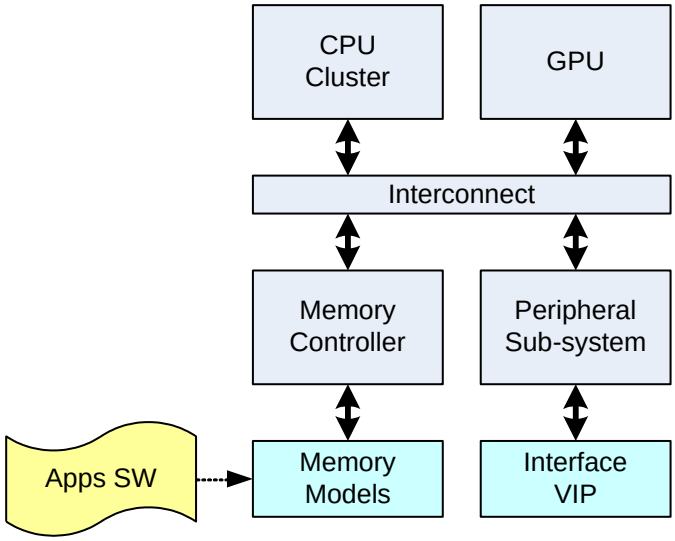

Figure 1 - Representation of a SoC Verification Environment

A. Validation process

Interconnect performance is something that is considered at the architectural level and spreadsheets are commonly used to arrive at a first order approximation to latency and bandwidth requirements. ESL tools, modeling the interconnect in SystemC, are also used to analyze system architectures and performance requirements. However, creating a model of a sophisticated interconnect with enough timing accuracy to get realistic results is a complex task and it is usually easier to work with the actual interconnect RTL.

Typically, the interconnect fabric is generated according to the specification and performance budgets estimated by the system architect. Then it is integrated with the rest of the design IP that makes up the SoC. The functionality of the fabric can be verified stand alone or as part of the wider device. The performance of the interconnect is usually checked when the full SoC, or sub-system, is in place and verified by running some use case scenarios using the RTL and at least part of the application code. This is represented by the diagram in Fig.1 where the application software is loaded into memory and then executed on the CPU cluster causing a series of transfers to take place across the interconnect. Setting up one of these use case scenarios relies on having all or most of the SoC in place and having access to software that is mature enough to support

the test case. Therefore, running the scenario is something that happens fairly late in the frontend implementation cycle, arguably too late to be effective. Running the scenario with the SoC RTL and the application software is very resource intensive and may take considerable time to execute, exploring variations around a particular scenario will take even longer.

Experience shows that doing an early stand-alone functional check of the interconnect structure can quickly isolate specification and implementation problems which would be difficult to find within the context of an integrated SoC design.

B. *An abstract validation strategy*

An alternative strategy to validating the performance of the on-chip interconnect is to create an environment where the fabric is verified stand-alone with bus agents in the place of the various bus masters and slaves and using abstracted stimulus. This approach is illustrated by Fig.2. Each bus master agent is driven by an abstract traffic generator with characteristics representing the device it replaces and the interactions between the stimulus streams running on the bus masters are modeled as scenarios. This approach can be used to check the performance of the interconnect before it is integrated, making it much easier to isolate problems and fix them with a shorter turn-round time. Using traffic generation with bus interface VIP reduces the verification resources required and means that performance validation can be carried out much earlier in the design cycle, without the need to have the full SoC RTL and application software available.

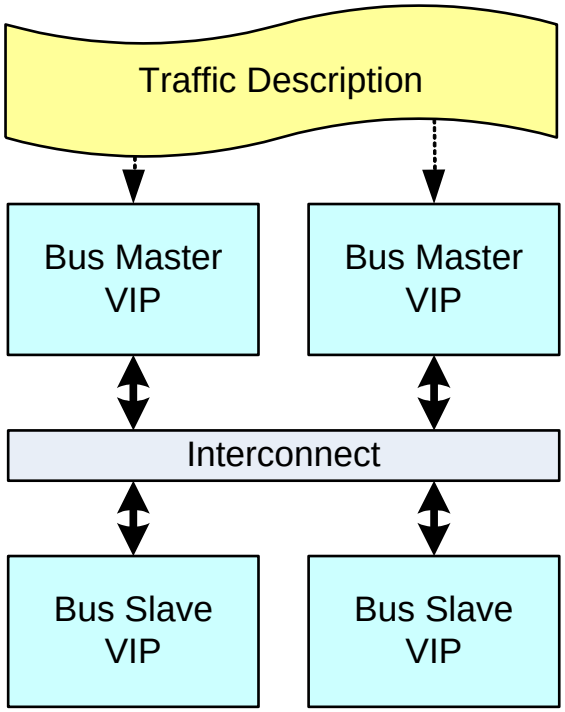

Figure 2 - Interconnect performance validation environment

C. *A bus traffic profile definition*

Putting together such a verification environment is relatively straight-forward since most of the necessary verification components such as bus agents are already available. However, finding a way to describe the stimulus so that it represents realistic bus traffic scenarios is not something that is well defined and understood. Mentor and ARM have therefore been exploring a way of defining bus traffic profiles so that they can be characterized for different classes of on-chip bus master devices and can be used to measure the performance of bus interconnects under different scenarios. In order to confirm that performance goals have been met, we have also defined some performance metrics which can be gathered by adding instrumentation to the verification environment. The proposed definition has been tested with a number of design IP providers in order to check that it is able to adequately describe the type of traffic their class of IP is likely to generate.

II. THE ARM-MENTOR BUS TRAFFIC PROFILE PROPOSAL

As with many other system level activities, SoC on-chip bus traffic can be represented by a series of abstraction layers. In the case of describing bus traffic, three layers can be used. The bottom layer is the physical layer, represented by the specifics of an on-chip bus protocol such as AXI4. The different transfers that can take place over the physical layer, and their bandwidth, are determined by the characteristics of the masters and the slaves in the system. The next layer up from the physical layer is the traffic profile which defines the characteristics of the traffic generated by a bus master. The top layer is the traffic scenario which describes the way in which the traffic from different bus masters is coordinated to simulate use cases.

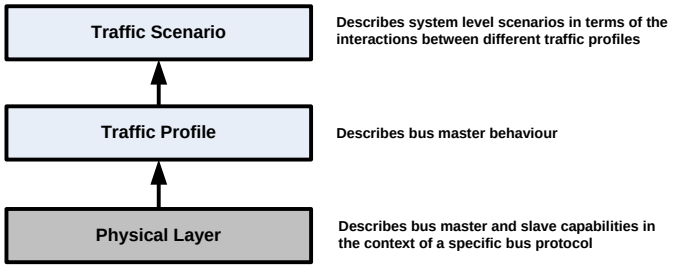

Figure 3 - Proposed traffic description layers

In any given design implementation, the physical layer performance will be fixed by the characteristics of the bus masters, the bus slaves and the bus interconnect. The traffic profile layer will be determined by the capabilities and characteristics of a particular bus master. If the bus master IP is configurable, then the traffic profile representing it may also be configurable, allowing a degree of architectural exploration to tune performance. The traffic scenario layer needs to be programmable since it describes how multiple traffic profiles interact and run on different VIP resources to represent different use cases.

A. *The physical layer*

At the physical layer, bus traffic is shaped primarily by the capabilities of bus masters and slaves. For instance, when a bus master transfers data with a slave the actual throughput between the master and the slave is determined by the maximum size of the individual transfers that both the master and the slave can handle. A master with a wide data path will be slowed down whilst making a transfer to a slave with a thinner data path, unless the interconnect is able to compensate for the slave by buffering transfers. Conversely, a master with a thinner data path than the slave it is communicating with could cause a slow-down in overall system performance by preventing other masters from accessing the particular slave.

1) *Bus Master Capabilities*

A bus master will have certain performance capabilities based on: the width of its bus fields; the maximum size of transfer it can handle; its reactivity to slave responses; the number of outstanding transfer requests it can issue and the sub-set of the bus protocol it supports. For bus interconnects that support multiple protocols, the bridge between one bus protocol and another may also introduce performance degradation.

Table 1- Physical Layer Capabilities Summary

Characteristic	Description
Bus Type	Defines which bus protocol the bus master or slave supports
Field Width	Affects performance with the number of data bits available to service a transfer.
ID Width	Affects the number of outstanding transactions that in turn affects performance across the interconnect
Burst Length	How many burst beats the bus master or slave can support
Limitations	Sub-set of the bus protocol the bus master or slave does not support, potential to affect performance.
Primary Timing	Timing parameters that directly affect performance
Secondary Timing	Timing parameters this indirectly affect performance with set defaults
Percent Error	Proportion of error responses returned by a slave

2) *Slave Capabilities*

Slaves affect the bandwidth of the system by their response to master requests. In addition to bus width limitations a slave will have latency, or a finite delay in responding to requests. A slow slave will have an adverse effect on the overall system performance, whilst a fast slave may not have any significant affect because it responds to a master faster than the master can generate new requests. In certain circumstances, a slave may also generate errors which will reduce the number of valid transfers which will impact the overall system performance.

For the purposes of interconnect performance validation a bus slave can be modeled as a 'perfect' slave, giving immediate response and maximum data bandwidth. This allows the raw interconnect performance to be evaluated.

The alternative to using a 'perfect' slave model is to use one that provides realistic response latencies so that overall system performance can be measured. This requires various bus protocol specific response delay parameters to be set up for a slave, usually specified as being within a range of values.

B. *Traffic Profile Layer*

The purpose of the traffic profile layer is to allow the traffic generated by a particular type of design IP bus master to be modeled in an abstract way. The stimulus for the traffic profile is then generated and executed on a bus master VIP instead of using the design IP RTL.

A bus master initiates data transfer requests across the bus interconnect to target slaves. At a first level of approximation, the size, frequency and locality of those transfers determines how much data is transferred between a master and its slaves in a given time period. However, there may be sub-characteristics of these transfers which mean that the overall traffic is better described in terms of a series of smaller transfers rather than a single burst of activity. In turn, those smaller transfers might be described in terms of another smaller set of transfers and so on. Underpinning the traffic profile layer is the physical layer. No matter how many layers of sub-profiles may exist, the bottom traffic profile layer interacts with the physical layer to implement the transfers.

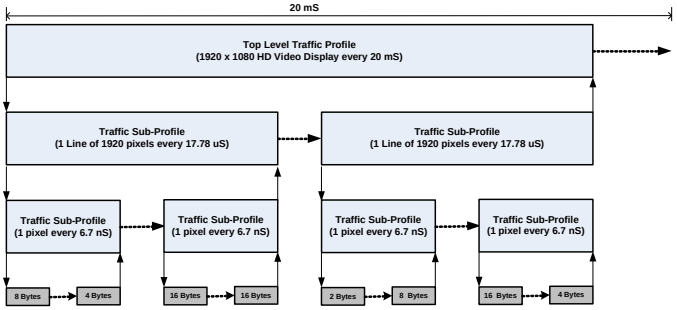

Figure 4 - Traffic Profile definition for a HD video frame

For example, the traffic for a 1,920×1,080 HD video display master might consist of a transfer profile that repeats itself every 20 mS to keep the frame buffer updated. However, the frame transfer could be broken down into blocks representing each of the 1080 lines which are fetched within 17.78 uS. In turn, the line transfer could be broken down further into individual fetches for each of the lines 1920 pixels which would have to complete within 6.7 nS. Finally, each pixel access is converted into one or more physical layer bus accesses. This traffic profile is illustrated by the diagram in Fig.4.

In order to describe a traffic profile we propose the use of a profile descriptor which can be used to describe all traffic profile layers. The parameters for the traffic profile descriptor are summarized in table 2.

The name parameter of the traffic profile descriptor is a string which is used to uniquely identify it. Sub-profiles are identified by their hierarchical path from the top level descriptor (<top>.<next_level>.<next …>).

The address range is used to limit the traffic generator to operate within an address range from a start or base address that is supported by the interconnect. The start address will vary from SoC to SoC and is therefore an overall stimulus generation control knob.

Table 2 - Traffic Profile Descriptor Parameters

Traffic profile descriptors	
Parameter	***Purpose***
Name	Label to identify the descriptor
Address_Range	Max address = base + Address_Range
Direction	Whether the traffic is read, write or mixed
Size	Number of bytes to be transferred
Stride	The offset in bytes from the current start and the next start address. If size == stride, then the next block continues from the last address of the previous transfer
Period	Time in which the transfer will repeat
Sub-profile(s)	Identifies any sub-profiles
Bus master capabilities	The physical layer capability description of the design IP

The direction parameter determines whether the traffic profile generates reads or writes or a mixture of the two. The mixed option is only valid for a top level profile and its use implies that there are read and write sub-profiles defined.

The size of each transfer in the traffic profile is defined in bytes since this is the lower common denominator in bus transfers. The size of any sub-profiles should be less than the size of a profile. During traffic generation, a sub-profile will be repeated until the number of bytes defined in its parent profile size parameter have been transferred.

The stride determines the start address of the next profile block transfer. If the stride value equals the size value, then the traffic profile will move through contiguous addresses, but if the size is less than the stride then there will a jump in the address value from the end of one block transfer and the start of the next.

The period defines the time that the transfer described in the profile descriptor should take. The period also defines when the next traffic profile will start. An alarm should be invoked if the traffic profile does not complete in time.

The sub-profile field is optional and, when present, contains a list of any sub-profiles to the current profile. The lowest level traffic sub-profile descriptors do not have any sub-profiles defined. During traffic generation, the lowest level of the traffic profile is translated into quanta of physical layer transfers.

A traffic profile can be described in terms of as many layers of sub-profiles as necessary, but in practical terms the limit is likely to be around four. If multiple sub-profiles are defined at a particular layer, then the generation process will make a random choice between them. For instance, if a top level traffic profile has its direction parameter set to mixed, then it would have at least two sub-profiles that describe transfers for the read and write directions. When the bus traffic is generated for the top level profile, then read and write transfers will be chosen at random until the size parameter of the top level transfer has been met.

Finally, as the traffic profile describes the behavior of a design IP, it needs to contain information on the physical layer that describes the bus protocol used and its bus master capabilities.

C. Traffic Scenarios

A traffic scenario describes the behavior of a system in terms of interactions between traffic profiles running on different bus masters. Since traffic profiles represent the behavior of target design IP, the physical layer of each traffic profile to be used in the scenario should be matched with a VIP in the verification environment that supports the target bus protocol.

A traffic scenario is primarily intended to model real-life use cases that would occur in the system and therefore a scenario needs to describe a number of different relationships between the generated traffic profiles. The relationships that can be specified are:

- Synchronisation – i.e. one traffic profile, or sub-profile can wait for one or more other traffic profiles or sub-profiles to complete before starting.
- Concurrency – Arranging for multiple traffic profiles to execute in parallel on different masters
- Repeats – How many times a traffic profile executes or whether it loops forever
- Ordering – Defining the order in which different traffic profiles execute on a given master

In order to describe traffic scenarios we have adopted a style of graph notation, whereby a traffic profile is represented by a hierarchical node containing a graph node for each sub-profile description. Since the traffic profiles execute on bus master models, rather than the actual RTL, each traffic profile needs to be associated with a bus master in the system, therefore the graph is overlaid on a set of rails which represent the progress of time for each master. The flow of execution is from left to right and is described by arcs between traffic profile nodes contained within a start and finish node. A dependency relationship is described by a series connection between nodes – i.e. a node has one predecessor and one successor. A concurrent relationship is described by a fork – i.e. a node has one predecessor but multiple successors, the different concurrent threads will ultimately terminate at the finish node. A triggering relationship is described by a ‘dotted’ arc which is capable of crossing concurrent threads in order to allow execution of one thread to be stalled until an execution node in another thread has completed. The completion of any traffic profile or sub-profile can be used to trigger the start of any other traffic profile or sub-profile.

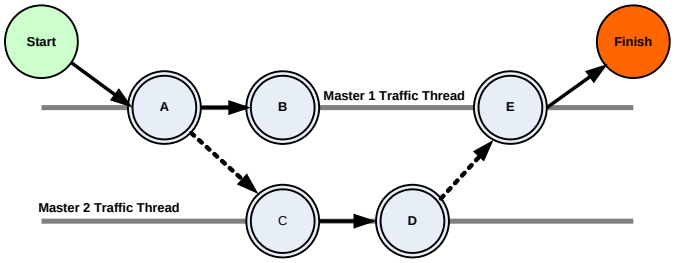

Figure 5 - Traffic Scenario Graph with dependencies and parallelism

The diagram in Fig.5 illustrates how these relationships are represented. The double circles are traffic profiles (which may or may not have sub-profiles) which execute on two different bus masters. At the start of the scenario, traffic profile A executes on bus master 1.When it finishes, traffic profile B starts on bus master 1 in parallel with traffic profile C on bus master 2. When traffic profile C completes, traffic profile D starts on bus master 2. When profile D completes, traffic profile E starts on bus master 1. When E completes, the overall scenario finishes.

More than one traffic profile might execute on a single bus master, this would correspond to the AXI protocols capability of being able to handle multiple outstanding transaction requests. The graph representation for this scenario would be to show two or more hierarchical profiles straddling the relevant bus master thread line.

In addition to the relationships between the traffic profiles, there are also a number of scenario control "knobs" which are used to define parameters which help fit the selected traffic profiles to the system characteristics. For instance, the start address of each of the scenarios has to be set to match the address map of the system as implemented in the interconnect. The primary physical level delays come with a defined value, but these can be tuned if the design IP is has parameterization options in this area.

1) Sliding Time Window

Running a traffic scenario gives some confidence that a bus inter-connect will be able to cope to with the performance requirements of a defined sequence of traffic events. One of the main objectives of this kind of analysis is to find interactions between bus masters that cause performance issues. Running multiple traffic profiles in parallel will have a natural tendency for them to drift in time relative to each other since they are likely to have different periods. However, in order pre-empt situations where masters might clash with bandwidth requirements each traffic profile thread that runs on each master can be subjected to a sliding time window. This allows the activities of different bus masters to move around in time, relative to each other causing interactions that may show up performance corner cases where the interconnect becomes overloaded, or worse still, locks up. The sliding window option preserves the graph arc relationships, but inserts delays within random time windows between the execution of each successor node within the overall time constraints.

D. Performance Measurement Criteria

The purpose of generating bus traffic is to determine whether the performance of an interconnect is acceptable. It therefore follows that some performance metrics need to be established to be able to objectively describe the behavior of the fabric.

Bandwidth and latency are the two main types of performance metric that can be measured in an interconnect system. Both of these can be determined by instrumenting the verification environment with monitors that capture phase level transactional information at the different points around the fabric. Each transaction contains a start and end time, together with other protocol level information relating to the size and type of the transfer.

1) Bandwidth Performance Criteria

Bus master bandwidth can be simplistically defined as the number of bytes transferred per master interface clock. The problem with this definition is that the master activity may be intermittent, which means that the on-demand bandwidth requirement, when the master is active, needs to be high, but over time this will be averaged out to a low bandwidth as the inactive periods are taken into account. This leads to a refinement of the original definition into several sub-definitions of bandwidth as measured from the perspective of a bus master or a bus slave:

- **Instantaneous bandwidth** – measured over a rolling time window of 100 interface clock periods
- **Peak instantaneous bandwidth** – The highest instantaneous bandwidth measured
- **Time windowed average bandwidth** – measured over a specified time window of a selectable number of interface clocks
- **Average bandwidth** – Overall average measured over the whole simulation

2) Latency Criteria

Latency is of interest in analyzing interconnect performance because the occurrence of high latencies usually points to a design flaw. Different bus protocols have their own phasing transitions, but in general terms latency metrics boil down to:

- Address/command phase latency – the time for the master bus request to be accepted by the slave
- Address to data latency – time from a transfer request being accepted to the start of the data transfer
- Data transfer latency – the overall time that a data transfer takes to complete – i.e. from the start of the address phase to the end of the data transfer phase

These generalized latency definitions map onto the AXI on-chip bus protocol as shown in fig 6. In the case of the AXI protocol, there are separate read and write channels and so there are two sets of latency metrics. The latencies shown hold true for the AXI3, AXI4, ACELite and ACE bus protocols.

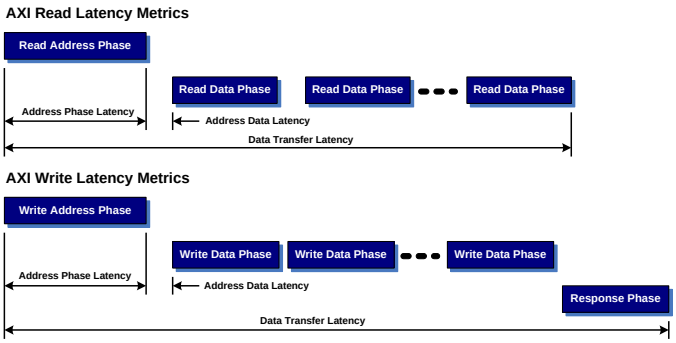

Figure 6 - AMBA AXI Bus Protocol Latencies

3) Channel Occupancy

Another useful metric, closely related to bandwidth and latency, is that of channel occupancy. This describes the amount of time that a particular channel was active. If the bandwidth requirements of a system are not being met, then often this metric gives an indication of whether the data channel is being swamped or whether a lack of responsiveness from the request channel is the cause of the problem.

4) System Level Performance Analysis

An on-chip bus interconnect is a sub-system and its performance can be analyzed from several points of view. A system level analysis of the traffic flowing between each master and its target slave ports, together with the associated latency information can help identify the source of performance issues.

For instance, with a reasonable amount of transactional analysis, it is possible to analyze the following for traffic between each master and slave:

- The number of read and write transfers
- The min, max and average latencies
- The min, max and average bandwidths
- The proportion of overall master traffic flowing to each slave and the proportion of slave traffic by master

III. Implementation

ARM and Mentor are exploring the practical implementation of the traffic profile description proposal with various mutual customers.

The first stage in the process has been to test the profile description against the anticipated system level behavior of various classes of design IP as reflected in bus level transfers and to refine the description as necessary.

The second stage has been to develop a practical verification environment implemented as a UVM testbench with bus protocol VIPs.

Fig. 7 shows a conceptual block diagram of the environment where each port of the interconnect RTL is connected to a VIP. The bus master VIP stimulus is generated from a traffic scenario, implemented as a virtual sequence which launches traffic profile sequences which are configured from a data structure. The physical layer of the stimulus is adapted from a generic bus transfer to the specific transaction type of the target VIP using an adaption layer. The bus slaves are modeled using slave sequences which mimic memory behavior with configurable inter-phase latencies.

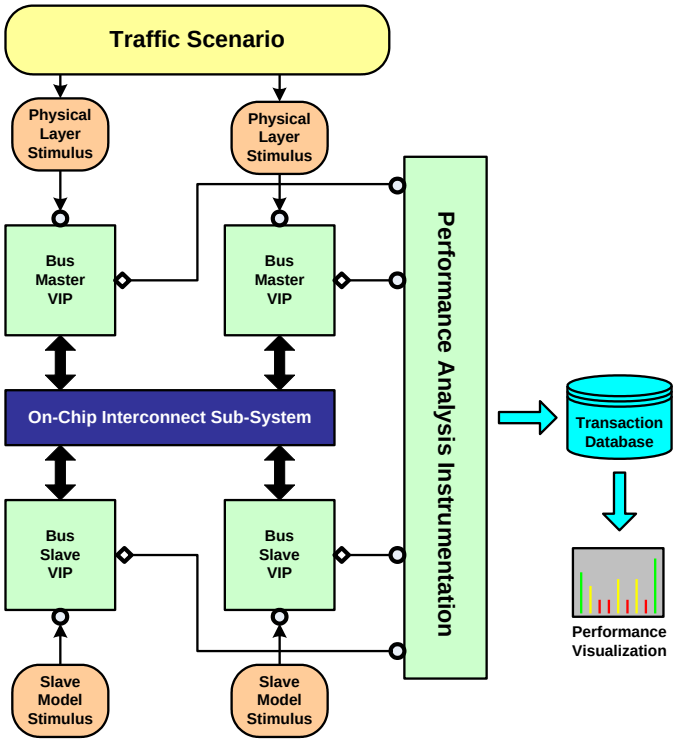

Figure 7 - UVM implementation of the traffic generation and performance instrumentation

The performance instrumentation is implemented with a performance analyzer component that collects transactions from all the VIP in the environment and analyses the relationships between them. The modified transactions are then stored in a database for off-line visualization.

IV. Conclusions

As the on-chip design IP component count increases in SoCs, the on-chip bus interconnect becomes one of the most critical sub-systems in the design. The generation of the interconnect is a largely automated process, but its validation is left until after integration and relies all too often on having the resources of the full SoC available to run application software. A more effective strategy is to validate the interconnect standalone, using a verification environment which uses bus protocol VIP to send realistic bus traffic across the interconnect. This paper has described a proposal for describing the behavior of bus masters in terms of traffic scenarios and traffic profiles. Performance measurement criteria which can be captured using testbench instrumentation have also been defined.

References

[1] ARM Ltd., AMBA Specification – Rev 2.0, ARM IHI 0011A, 1999

[2] ARM Ltd., AMBA AXI and ACE Protocol Specification, ARM IHI 0022D, 2011