Framework for Exploring Interconnect Level Cache Coherency Using Systemc-TLM

Parvinder Pal Singh, Senior R&D Engineer, Synopsys, India,
email id: ppsingh@synopsys.com

***Abstract*—With the proliferation of caches, the challenge is to provide consistent views of shared data and to minimize the external memory bandwidth requirement, which is increasingly becoming the main system performance bottleneck. This data-sharing, communications and data movement combined with the presence of multiple caches amongst heterogeneous processing units drives the demand for a flexible cache coherency protocol that can work not just amongst like CPUs (as has been done many times in the past on various processor architectures), but can also work between dissimilar CPUs as used in big.LITTLE technology, GPUs and the hardware accelerators present in today's SoCs.[1] For achieving the above mentioned objectives, many protocols supporting the coherency exist in today's world, most common of which are ACE, CHI [2]. These protocols open a vast space for the design exploration, the kind of different configurations, implementation that can be possible using above protocols is huge and design space around cache itself is exploding.**

This paper proposes a generic coherent interconnect solution that can be extended to any protocol with little efforts and provide out of the box configuration like support of any number of coherent/non coherent masters, downstream coherency port, different cache line size, speculative fetch, directory and broadcast mechanism etc. which help user to analyze and optimize its system. Later specific IP implementation like ARM® CoreLink™ CCI-550, Arteris N-core™, NetSpeed Gemini etc. can be used for validating the system.

In this paper we will discuss what are the underlying challenges while doing exploration and optimization of coherent design and later discuss how that problem can be solved with the help of the generic interconnect. We will end the discussion with the case study we have created to support the solution.

Keywords—Coherent Interconnect; System Level Analysis, coherency protocol, snooping mechanism, cache coherent, snooping, Architecture Exploration.

I. INTRODUCTION

Caches provide a mechanism to overcome the memory latency issue for the data that is being frequently used by the processor. But now a-days when we have multiple CPUs sharing the same data, it becomes necessary to provide them with coherent view of the memory. This also means that if two or more CPU/Cluster sharing some data in their local cache then the view of the data should be consistent at any point of time. In previous days we were using software based coherency mechanism which are complex and inefficient. To overcome this, various coherency protocol have been developed which allow different cache to be coordinated with each other and provide the same memory view to all the CPUs. Maintenance of coherent data across different caches is the responsibility of the interconnect (coherent interconnect). With the increasing number and increasing complexity of coherence protocols, complexity of coherent interconnect is also increasing and provides a vast space of design exploration
around the interconnect model only. "Figure 1" shows some of the design space exploration challenges around interconnect:

- **Selection of coherency protocol**: Selecting a protocol is a tradeoff between complexity and performance. Selecting a coherence protocol that fulfills the power and performance requirement is important. Most common are MSI, MESI, MOESI [3] etc. Choosing suitable protocol depends on many factor, if you want to minimize the logic to maintain the coherency you will go with MSI protocol, but the problem is it will increase the snoop/memory traffic. For scenarios where optimization is not major concern compared to snoop traffic, you will go with MOESI.
- **Interconnect Rule:** Include interconnect properties like cache line size, arbitration scheme, speculative fetch etc. During exploration above listed property also plays an important rule. For example, speculative fetch increases the downward traffic but at the same time lower the latency in case of cache miss. So for your system you may want to analyze that for a given kind of configuration + traffic, which approach will serve you best. Similarly, you want to have different coherent master with different cache line size

connected to interconnect or want to explore data from different cache line of your cache, for which you need a interconnect model that can be configured easily for different cache line sizes.

- **Snooping mechanism:** Broadly two kind of mechanisms are used, directory based or broadcast. In broadcast snoop systems the coherency traffic is proportional to:
  N*(N-1) = N^2–N where N is the number of coherent masters
  Since for each master the broadcast goes to all other masters except itself, so coherency traffic for 1 master is proportional to N-1, and since each master can do this then you multiple by N originators to get N^2–N. Clearly purely broadcast-based snooping systems have limits to their scalability.
  In directory based system there is single 'directory' which contains a list of where every cached line within the system is held. A master initiating the transaction first consult the directory to find where the data is cached and then directs coherency traffic to only those master containing cached copies. For worst case scenario traffic scales at order N^2 which is rare (when all master have cached copy) but possible based on the application running on CPU. So it become necessary to find what is suitable method.
- **Interface Protocol:** Sometime architects do not want to limit themselves to a single coherency protocol, so they want to try out different protocol like ACE/CHI or some other predefined protocol to see which gave them best performance. Different protocols (AXI/CHI) have different interface requirement.
- **Shareability:** A shareability domain is a set of master components that enables a master component to determine which other master components to include when issuing coherency. For coherent transactions, a master component uses the shareability domain to determine which other master components might have a copy of the addressed location in their local cache. The interconnect component uses this information to determine, for any given transaction, which other master components must be snooped to complete the transaction. From the master's view it's an important decision to decide which master should be kept in shareable domain to avoid unnecessary snooping traffic. For the same to decide it's necessary for a master to have system level view.

Apart from the design space exploration challenges discussed above, another designer issue is the lack of system level analysis view to present the data for various exploration use cases. This limits the visibility in the system and potential performance hot spots.
Many IP vendors have come up with different solutions for coherent interconnect design some of them are, CCI400[4], CCI500[5], CCI508[6] of ARM and NCore[7] of Arteries etc. But these have been restricted on configurability and design space that can be explored like, CCI400 support only 2 coherent master interface, all the solutions currently available lacks to provide system level analysis to the user.

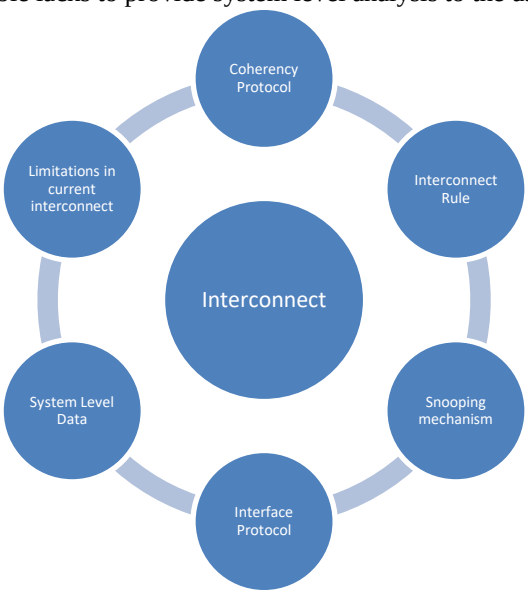

Figure 1 Design Space exploration challenges for interconnect

So a generic coherent design is needed for system implementation and that should also provide various customization on the interconnect to explore the design space. Apart from this solution should be able to present the system level view with the help of the analysis.

II. PROPOSED SOLUTION

We have designed a generic methodology for developing the cache coherent interconnect, using which any cache coherent interconnect can be configured or uplifted very easily for entirely new protocol without rewriting the entire interconnect model. We have currently enabled AMBA ACE protocol and AMBA CHI using the same methodology.
Capabilities of generic cache coherent interconnect for exploration can be categorized as:

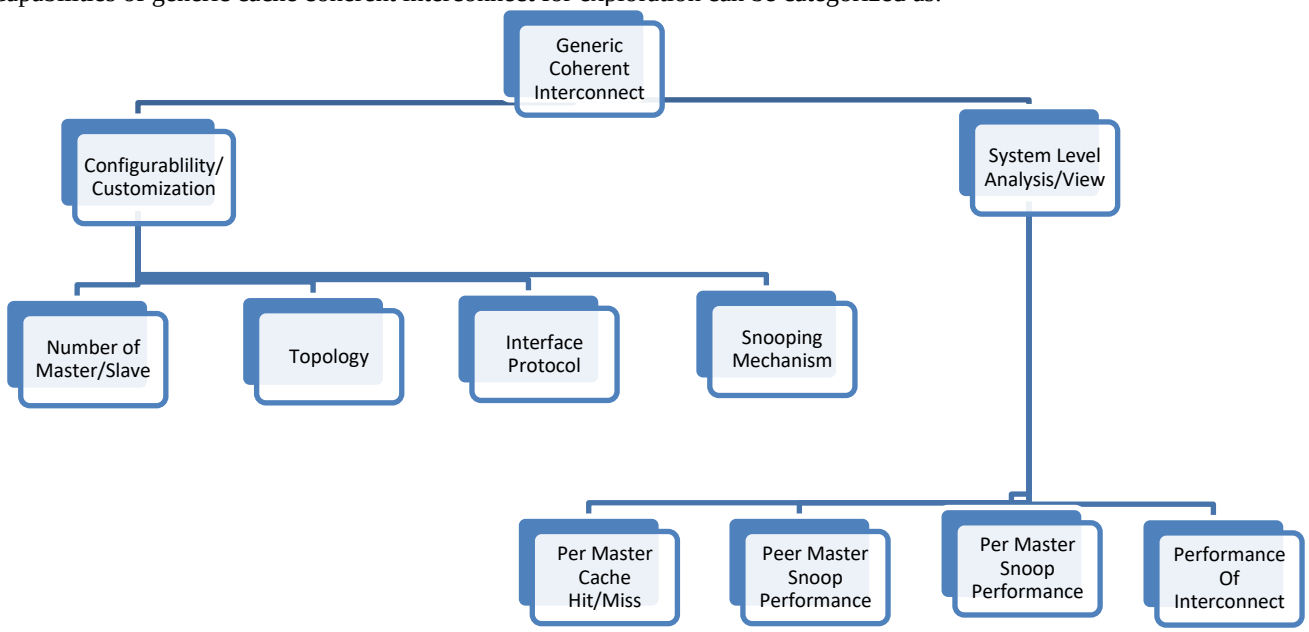

Figure 2:Capabilities of generic coherent interconnect

As seen from above diagram, capabilities of the generic module can be categorized broadly into two categories

- Customization and Configurability
- System Level Analysis/View

Detailed description of each feature are:

Customization and Configurability

Different configuration and customization are:

- **Configurable number of masters/slaves** Any number of coherent/non coherent masters and slaves can be connected. The major drawback with other available interconnect models is that they provide limited number of interfaces to connect to interconnect. Our model does not have any limit on this, you can connect as many as coherent/semi-coherent and non-coherent initiators and targets.
- **Capability to create complex interconnect topology:** You can configure interconnect to work for different cache line size, data width, speculative fetch etc. which allows you to try different configurations of your interconnect and come up with best possible configuration for given kind of application.
- **Extendibility to explore coherency protocol:** Current functionality can be extended with the use of callback mechanism provided. Which allows you to control snooping mechanism, directory structure, coherency command you send etc. Which helps you to determine, best suitable protocol out of MSI, MOESI etc., best suitable cache eviction policy if directory is full. And control over snooping.

- **Ability to configure the snooping mechanism:** User can select either broadcast or directory based protocol. As discussed above based on your application this parameter plays an important role to determine best configuration.
- **Ability to choose the interface protocol:** Generic interconnect provide support for different protocol like ACE/CHI. The inner functionality is made protocol agnostic. User can provide its own protocol handler and can use the inner functionality. Hooks are also provided to modify/re-write some or all parts of inner functionality.

System Level Analysis/View:

System level data plays an important role in determination of various factor for e.g. sharability, cache line size, snooping mechanism etc. All the features discussed above will help you only when you have system level analysis(data) available with you. Generic coherent interconnect model provides you with the graphical representation of various system level view that helps architect to fine tune his design and decide the best possible configuration. Some of the analysis view and how they help in determining configuration are discussed below:

- **Per Master Cache Hit/Miss:** System level view of hit and miss for each master plays an important role in determination of various factor. Considering the software running on the system as constant factor other parameter which can reduce the Miss ration are Optimal cache line size, Higher Associativity**,** Victim Cache, Coherency Protocol selected (MSI, MOESI, MESI).
- **Peer Master Snoop Performance:** This analysis data provides the view that which peer master is providing the data. This information is helpful in the selection of sharability domain and snooping protocol.
- **Per Master Snoop Performance:** This parameter is an indication for snoop performance of each cache (master cache). Using this parameter, it can be judged that how frequent invalidation of cache line is being done on master and what other master are causing this. With the help of this proper work load balancing of an application can be achieved
- **Performance of Interconnect:** This can be regarded as the most critical parameter that should be topic of interest for any system architect. System level analysis for interconnect can help in determination of various factor. For example, if bandwidth is not a concern for the user, he can select speculative fetch and minimize the memory access time. What should be the snooping mechanism and how much should be the size of the directory all these parameters are critical for your system which can only be concluded with the help of system level views available.

System Level Design is considered the appropriate way to deal with the ever increasing complexity and heterogeneity of SoC architectures [9]. The highest possible abstraction level for design space exploration and application mapping is static performance analysis [10]. Our methodology and analysis strategy mainly assist in architectural exploration through simulation of dynamic workload, which is more close and realistic as compared to static analysis done by architects usually through spreadsheets.

Below “Figure 3” shows the high level view of generic cache controller block:

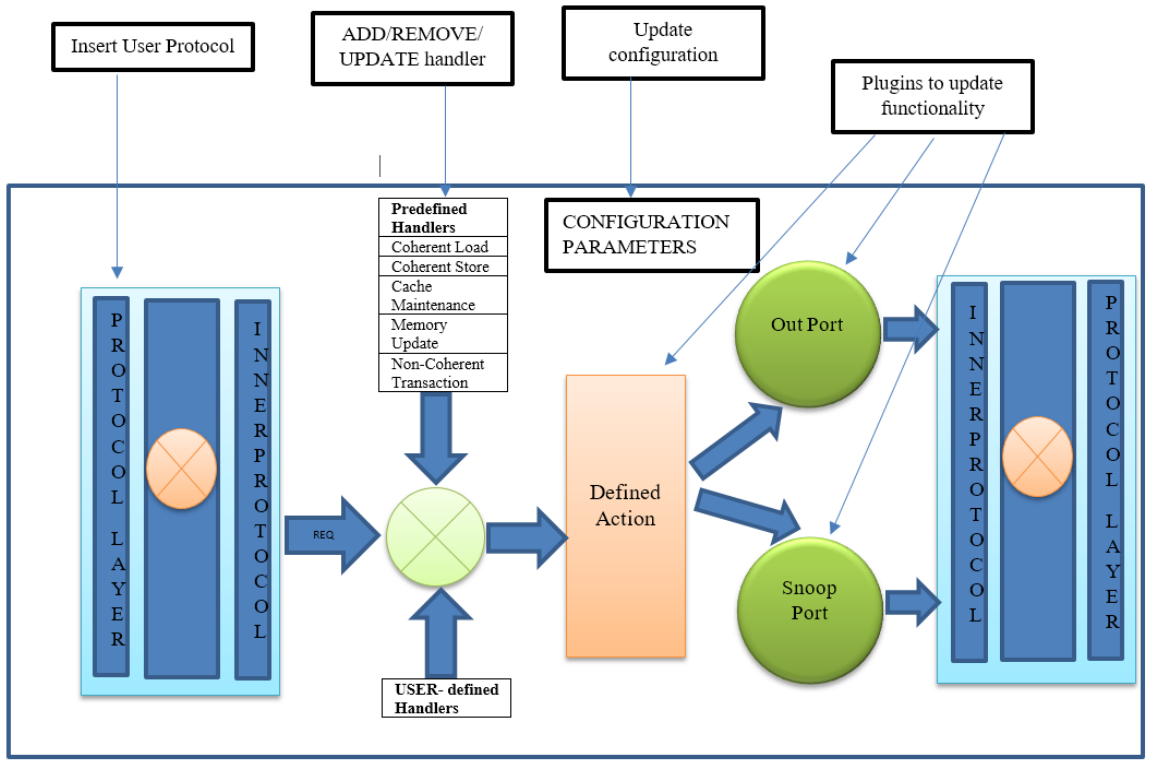

Figure 3: Block Diagram of Generic Cache controller block

As described in above figure 3, generic cache controller block consists of

- Protocol Agnostic Layer
- Set of handler to define action
- Defined action for each handler
- Functionality for Snoop and memory (out) port
- Configuration parameters

Protocol agnostic layer provides mechanism to separate out inner functionality from the selected protocol. By defining your protocol engine user can use the complete functionality without much efforts.

Interconnect model comes with set of predefined handlers for different type of transaction which can be appended, modified, or removed depending upon the use case. User can also plug-in a new set of handler to the model.

Based on the handler registered, when a new request is received corresponding action is called.

Before simulation start user can configure the interconnect model using different configuration settings. Also Plug-in functionality provided at different stages of the model to help user to insert desire functionality.

III. How SystemC TLM2.0 helped

The configurable generic solution described above is possible with the help of the SystemC-TLM2.0 standard which is developed for memory mapped protocol. Following features of the SystemC-TLM2.0 helped to achieve the goal described above.

- Standard provides the capability to extend the protocol using extension mechanism. Any memory mapped protocol can be extended with the help of this.
- Communication protocol can be enhanced to the new complex protocol like ACE using dual socket.
- Additional information can be added to the payload to get the system level analysis. As said system level analysis is the key feature to know the performance of your interconnect with given traffic under different configurations.
- Standard enables to get accuracy close to cycle accurate level without compromising the speed. Model at RTL level provides accuracy but at the same time speed is very slow to do this kind of exploration study. Also developing RTL model, itself is tedious. So with the help of language it's easy to uplift model based on requirement and get close to accuracy as close as cycle accurate.

IV. CASE STUDY

To back our claim, we did a simple case study to get the best possible configuration of interconnect model. For the same we have used a mobile platform. Traffic for the platform is taken from the previous generation (Nth) model using synopsys TGG (task graph generator) tool. The traffic then played using the traffic generator module (VPU), which reads the data generated by TGG.
Figure 4 represent the platform which we have used to validate our study.

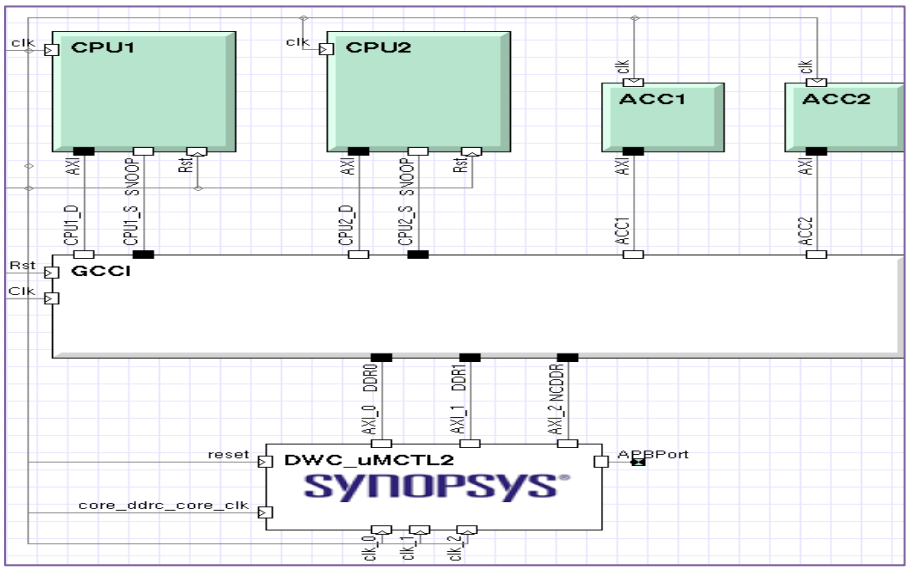

Figure 4: Simple platform to explore interconnect configuration

So our primary interest in this case study was to minimize the snoop traffic to other master. Other parameter which we did exploration case study are:

- Snooping mechanism
- Directory Size
- Different coherency protocols (MSI, MESI, MOESI)

Let's discuss results from each of the above mentioned items one by one.

Snooping Mechanism (Directory/Broadcast)

Now we will study which snooping mechanism holds good for our platform so for the same we have used above mentioned platform and ran it with different configurations (broadcast/directory). Result of the study are shown in below figure 5.

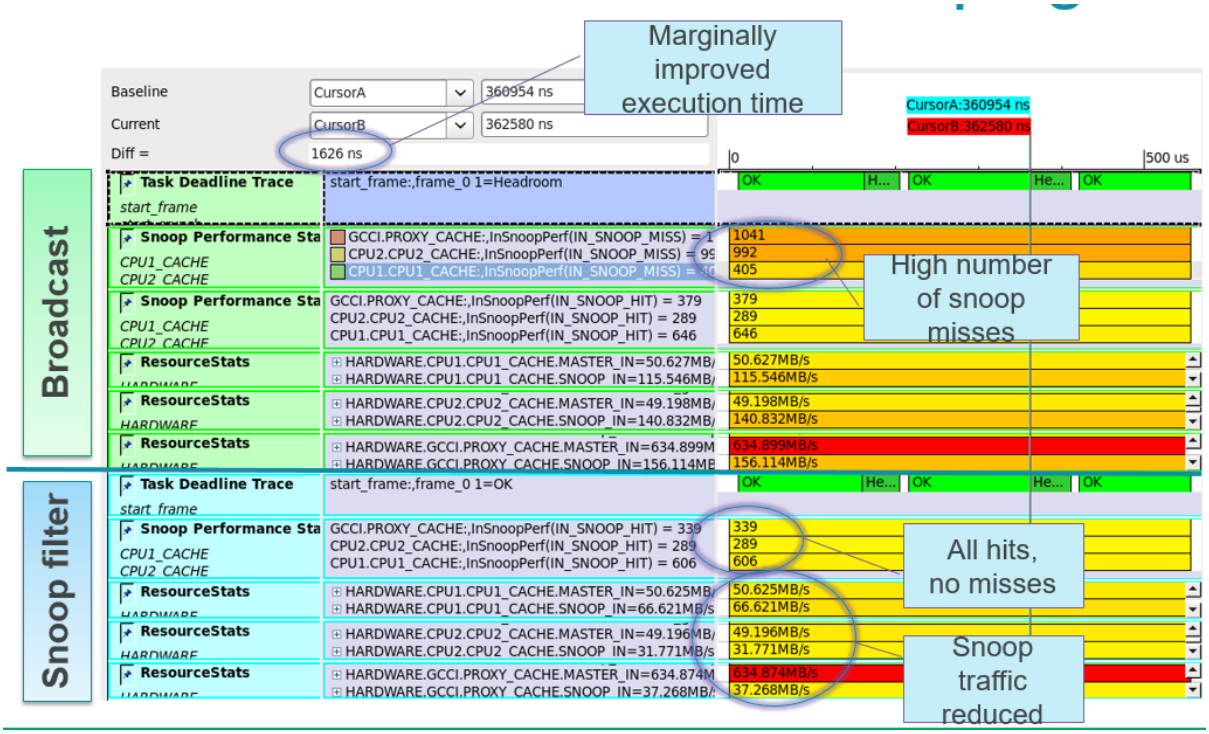

Figure 5:Broadcast vs Directory

As can be seen above, there is slight improvement in simulation time when we use directory based mechanism. Also directory helped us in reduction of snoop traffic as shown above. It's a clear win case for directory based mechanism in this case.

Different Directory size:

Once we have decided the suitable snooping mechanism (directory based), next task for us was to explore the optimal directory size. For which we again ran the same traffic with different directories. For simplicity we will describe results with 3 configurations here, which are:

- Directory of size 10,000 (Figure 6)
- Directory of size 20,000 (Figure 7)
- Directory with size 5000 (Figure 8)

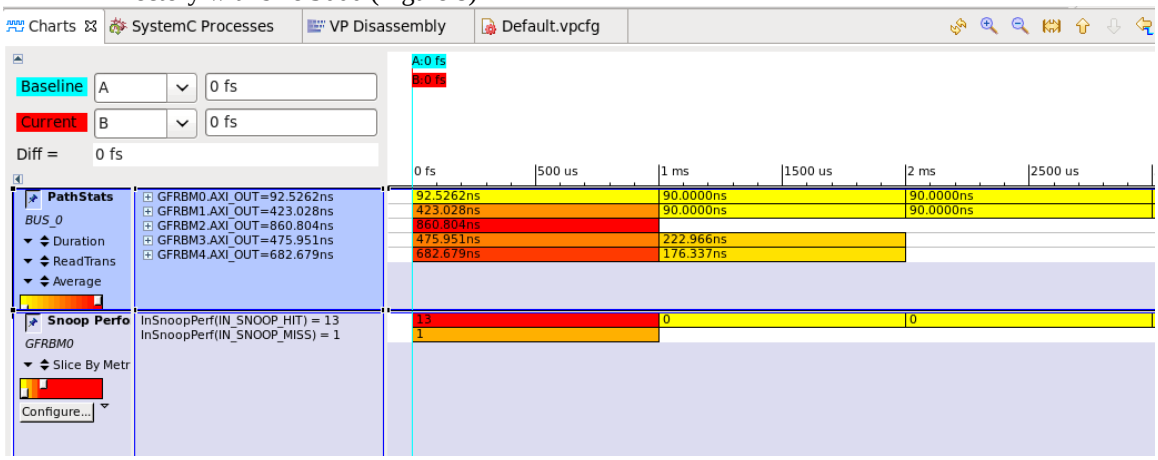

Figure 6:Analysis results with directory size 10,000

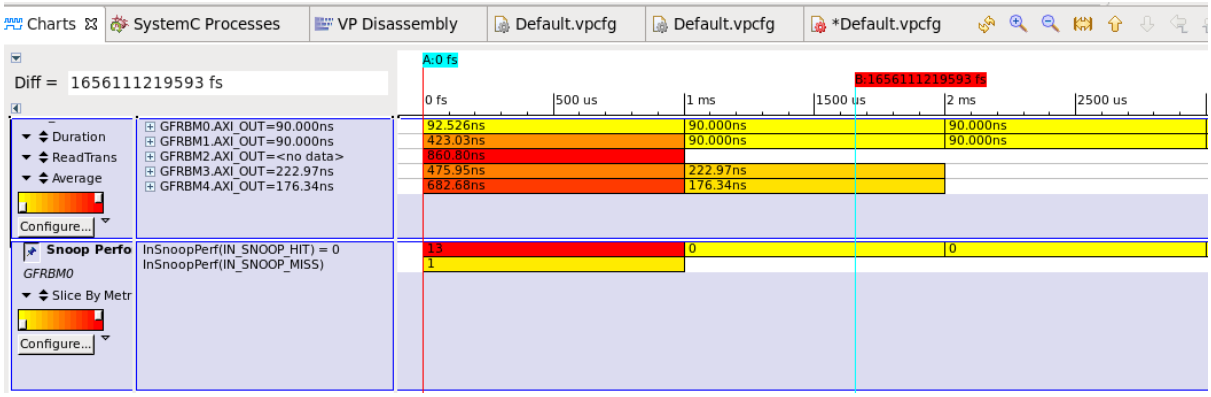

Figure 7: Analysis results with directory size 20,000

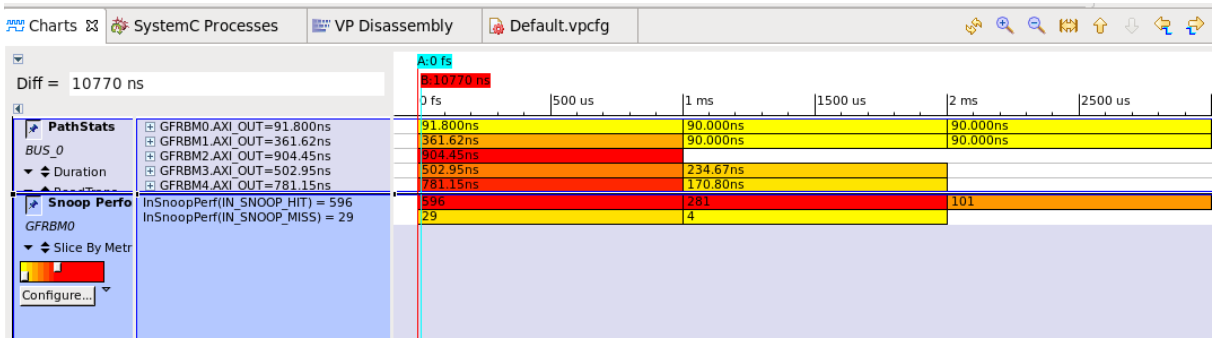

Figure 8:Analysis results with directory size 5000

As seen in figure 6 and figure 7 (for directory size 10,000 and 20,000), snoop traffic count and duration remains same for both the scenarios which clearly indicate that with increasing directory size to 20,000 we are not getting any benefits in terms of reduction in snoop traffic.
Now comparing figure 6 and figure 8 (for directory size 10,000 and 5,000), we found that snoop traffic is increased dramatically and the reason for this is frequent invalidation that snoop controller must do to invalidate any existing entry which caused the snoop to miss in the cache of masters and increased the snoop traffic. However, there is improvement in the average time but reason for that is that overall traffic is increased in the system which reduced the average duration that can see clearly in figure 9 shown below:

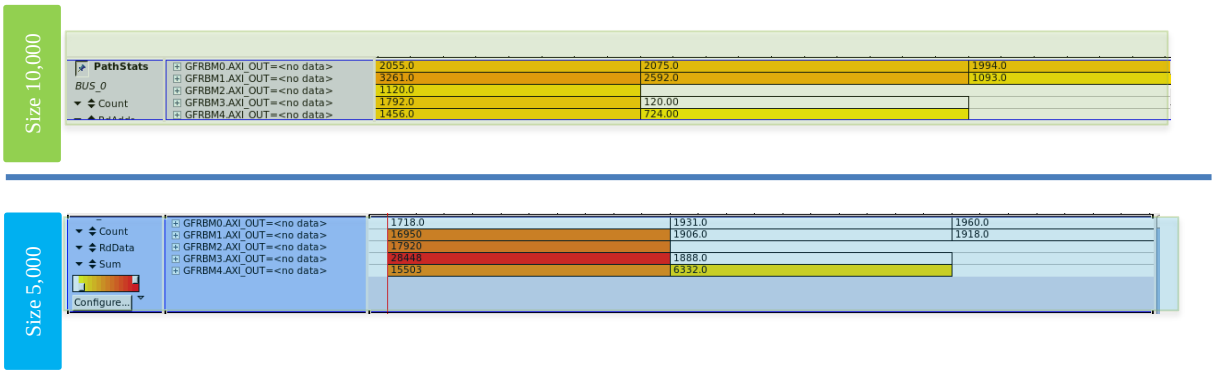

Figure 9: Transaction count difference between directory size 10,000 and 5000

Different Coherency Protocols (MSI, MESI, MOESI)

We ran same traffic in the system for above mentioned 3 protocols to see which protocol gives best result. We have studied it based on total simulation time consumed/traffic generated to memory controller block and snoop performance parameters like incoming request, main memory access, number of hit and write back.

When comparing overall simulation time, we found that MOESI took less time for completion. However, MSI took comparatively higher time and resulted higher number of memory transactions.
Results can be seen in figure 10 shown below:

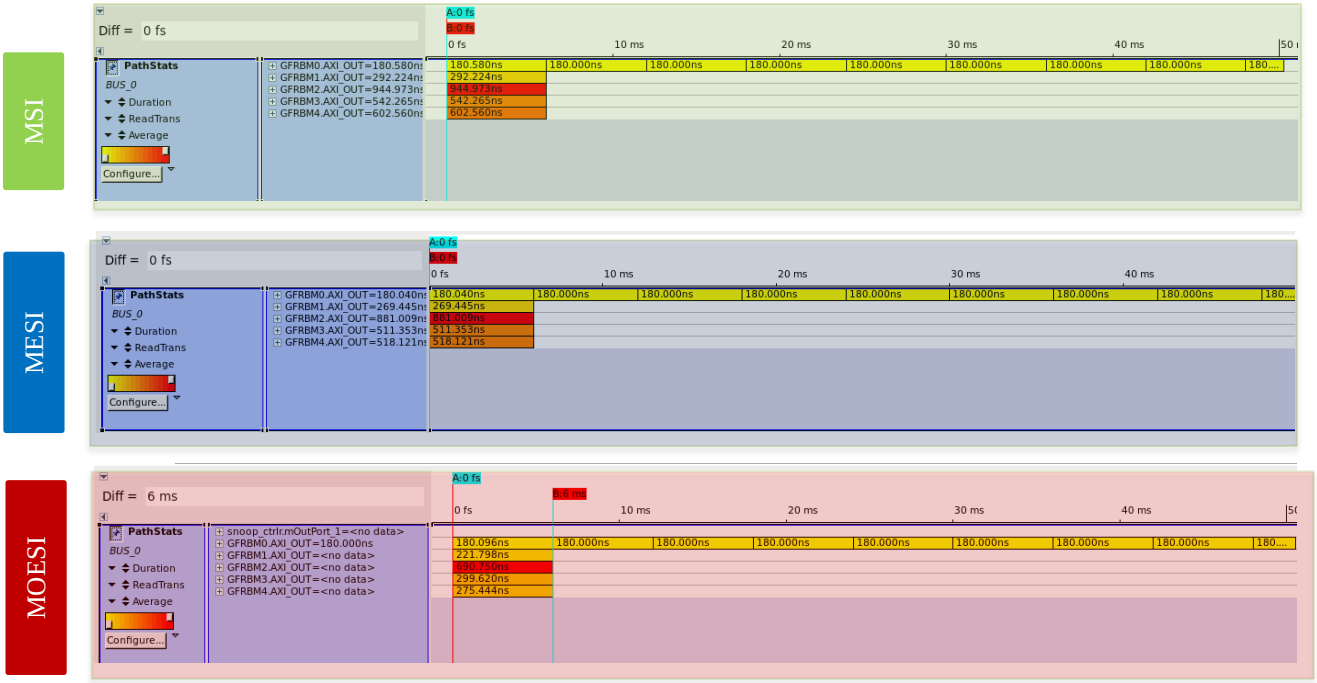

Figure 10: Simulation timing for MOESI, MESI and MSI

Apart from this snoop performance traces also must be evaluated before making any final call. When we have analyzed the snoop performance traces we realized that for our traffic MESI protocol seems to be best performing as compared to the MOESI and MSI protocol as it generates less write back, snoop miss etc. Figure 11 shows the snoop performance stats.

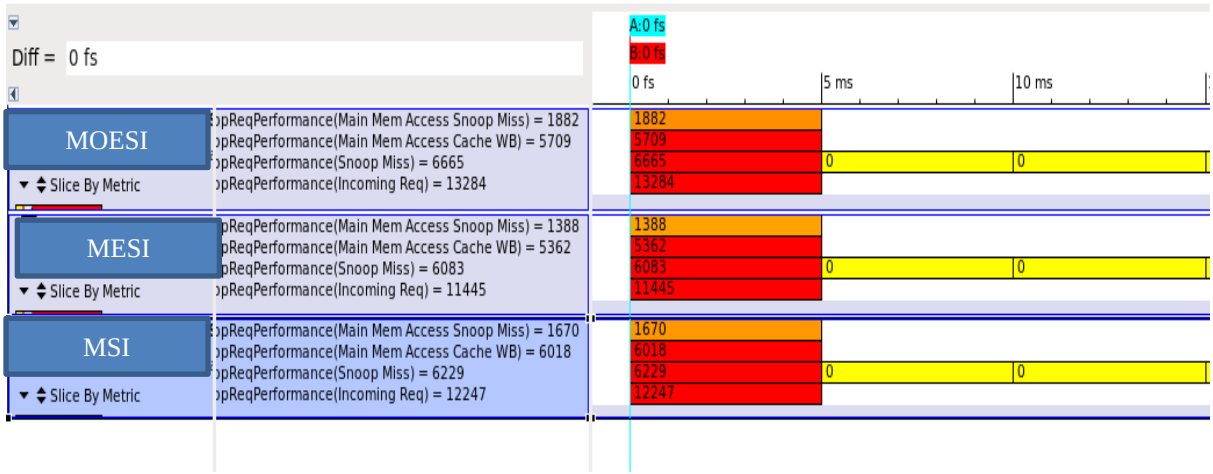

Figure 11: Snoop Performance Stats

As can be seen that MESI have an edge over the other two protocol when compared to snoop performance. Also our aim for the study is to maximize snoop performance for which MESI gave best results.

V. CONCLUSIONS

As stated previously hardware coherency provide a vast space for exploration which is currently not possible from the existing solutions because of lack of the configurability and system level analysis.
With the help of SystemC/TLM2.0 mechanism we tried to solve these problems related to configurability and providing system level view. There is lot of configurations to explore, with the help of proper configurable modules and analysis traces it can be achieved.
Also not only configuration which matter but the speed of the system also matters. We have completed our study in even less than half a day, which for any typical RTL model may take months of effort.

REFERENCES

1. https://www.arm.com/files/pdf/CacheCoherencyWhitepaper_6June2011.pdf
2. http://www.arm.com/products/system-ip/amba-specifications.php
3. http://sc.tamu.edu/systems/eos/nehalem.pdf
4. http://www.arm.com/products/system-ip/interconnect/corelink-cci-family/corelink-cci-400.php
5. http://www.arm.com/products/system-ip/interconnect/corelink-cci-family/corelink-cci-500.php
6. https://www.arm.com/products/system-ip/interconnect/corelink-ccn-family.php
7. http://www.design-reuse.com/news/39827/arteris-heterogeneous-multicore-cache-coherency.html
8. IEEE (Std 1666-2011) Standard for Standard SystemC® Language Reference Manual
9. K. Keutzer, S. Malik, A.R. Newton, J.M. Rabaey, A.Sangiovanni-Vincentelli. System-level design: Orthogonalization of concerns and platform-based design. IEEE Transactions on Computer-Aided Desig of Integrated Circuits and Systems, 19(12):1523–1543, December 2000
10. https://www.researchgate.net/publication/225210888_A_Power-Efficient_Methodology_for_Mapping_Applications_on_Multi-Processor_System-on-Chip_Architectures