Novel Approach for Cache Memory Simulation at Approximately-Timed TLM Abstraction Level for SMP System

Nishit Gupta, Sunil Alag
R&D in Electronics Group, Department of Electronics & Information Technology
Ministry of Communications and Information Technology, Government of India
{nishit.gupta, salag}@deity.gov.in

***Abstract*— To meet the ever increasing high speed computing requirements of complex System on Chips (SoC), there has been an increasing trend towards adopting Symmetric Multiprocessing (SMP) Systems encapsulating multiple processing cores each having multi-level cache memory for faster accesses. At an early design phase, estimating the requirement of - cache memory size and levels, prefetching strategy, snooping mechanism and coherency protocol to be adopted may save a lot of RTL simulation time and SoC area. Also, optimized cache parameters facilitate better SoC performance in terms of Bandwidth, latency, FIFO depth, arbitration policies etc. of various IP cores.**

In this work, keeping above in view, a novel approach is proposed to simulate coherent (Multi-Level) Cache Memory based on MESI protocol for Multi-Core Symmetric Multiprocessing (SMP) System deploying the benefits of Timed TLM simulations at an early design phase. The proposed Cache Memory system is provided with the memory reference traces extracted from earlier SoC simulation. Based on the user requirement, a memory hierarchy is dynamically generated which produces various cache memory access statistics which can be appropriately used for optimizing Cache Memory parameters.

Keywords—MESI; Symmetric Multiprocessing; Snooping; Cache Coherent; SystemC

I. INTRODUCTION

The impact of Microprocessor and its applications is significant in various lures of fields. It is a well accepted fact that every electronic device of today has embedded with a microprocessor in one form or the other. The advent of Multi-Core processors in the last decade, deploying the advantages of Symmetric Multiprocessing (SMP) system, has drastically changed its traditional computing mechanism. Such SMP systems have centralized memory which is shared among various processor cores operating under a single operating system. Each core, besides this shared memory, is usually associated with high-speed private memory commonly known as cache memory to reduce the system bus traffic and for faster accesses. Cache Memory is traditionally categorized at various levels that describe its accessibility and closeness to the processor core, i.e. Level 1 Cache is extremely fast but relatively smaller, expensive & closer to processor core as it is privately held by core while Level 2 Cache is relatively slower, larger, cheaper and generally shared among various processor cores. The efficiency of Cache Memory is determined by various Cache Access Time parameters (i.e. Cache-Hit & Cache-Miss ratio & time etc). In general, Average Cache Memory Access Time (AMAT) = Hit Time + (Miss Rate X Miss Penalty). Since the performance of Cache Memory is directly proportional to AMAT, to improve Cache performance, various cache parameters may be optimized including Hit Time, Miss Time, Cache Size etc.

In this work, a novel approach (based on cache simulator) is being proposed which aims at providing a tool modeled at Approximately-Timed Transaction Level abstraction level facilitating investigations of various cache parameters i.e. cache size, eviction policy, write-back, write-through, etc for SMP based Multi-Core processors. This tool facilitates the user to vary some or all of the cache parameters, and investigate how cache-misses, at the L1 & L2 levels change, and hence to derive optimized bandwidth of the data & instruction flows at the various levels of the memory hierarchy.

The proposed approach relies on simulation trace-files extracted from earlier SoC simulations representing the flow of instruction fetches and data reads & data write as sampled at core/ L1 interfaces. This approach deploys the benefits of Timed TLM simulation while implementing a clock (sc_clock), which is used to model the access delays to the L2 cache to refill an L1 cache line, and the L2/ L3 (i.e. L3 is the main memory) delay. The proposed tool also implements the cache coherency protocol viz. "MESI", which stands for "Modified, Exclusive, Shared, Invalid". At the end of simulation, various statistics on Cache Memory accesses are produced by this tool which can be used to optimize Cache Memory parameters at an early design phase.

II. RELATED WORK

While the concept of optimizing Cache Memory using HDL & sequential programming languages is not new [2,3,6,14], to the best of our knowledge not much effort has been put in the direction of exploring these techniques at an early design phase using approximately timed TLM abstraction level with a notion of clock for Multi-Core processor architecture wherein each core sharing various level of Cache Memory [1,7,13,15,17]. Also automation and standardization of the process remains a concern. Department of Computer Science, University of Wisconsin has, to an

978-1-5090-1489-7/16/$31.00 ©2016 IEEE

extent, addressed this issue by introducing Dinero IV Trace-Driven Uniprocessor Cache Simulator [10], however, which being a C-model lacks in the automation of using timed traces for hardware simulation. Also, it couldn't address the Multi-Core processor SoC scenarios [4, 6, 7, 11].

III. Model of Computation

The proposed approach adopts a Model of Computation (MOC), implemented in SystemC to achieve parallelism needed for Multi-Core behavior, which separates communication from computation through Timed TLM modeling as demonstrated in [5,8,9,12,16]. System synchronization is modelled in proposed cache simulator by specific means such as events, interrupts & signals. Compared to RTL simulation, it involves lesser context switches due to fewer parallel executions of processes resulting in high simulation speed. Clock provided by SystemC is being used to keep track of cache Hit & Miss Time & various other memory access parameters. The proposed Cache simulator interacts with other IPs of system at System Synchronizations points as explained in Figure 1. The advantage of adopting such MOC is that it enjoys the benefit of faster simulation due to TLM abstraction level without losing accuracy provided to it by timing delay through SystemC kernel.

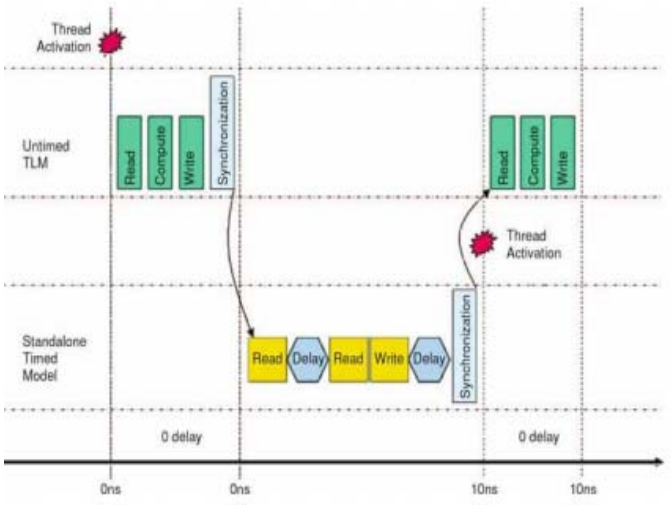

Fig. 1. Inter-Execuion of Un-timed & Timed Models at Transaction Level

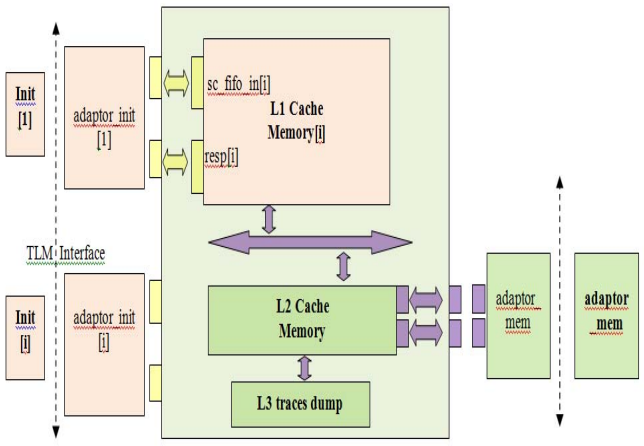

Fig. 2. Cache Memory Simultor Architecture

```
                Details of the 4 nibble (16 bits) attribute :
/* ============================================================ */
/*                                                              */
/* Bit definitions for "raw_attribute" field of MemAtt_t:       */
/* N.B: Unsigned short int is supposed to have 16 bits !        */
/* 15  14  13  12  11  10  09  08  07  06  05  04  03  02  01  00 */
/*  |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   | */
/*  0   0   0   0   0   0  M1  M0  OS OP1 OP0  OC  IS IP1 IP0  IC */
/*          Normal (ARM) __ 0   0   |   |   |   |   |   |   |   | */
/*          Device (ARM) __ 0   1   |   |   |   |   |   |   |   | */
/*Strongly Ordered (ARM) __ 1   0   |   |   |   |   |   |   |   | */
/*        Reserved _______ 1   1   |   |   |   |   |   |   |   | */
/*                                 |   |   |   |   |   |   |   | */
/*  Outer Shareable _______________ 1   |   |   |   |   |   |   | */
/*  Outer Non-Shareable ___________ 0   |   |   |   |   |   |   | */
/*  Outer Write Cache policy (WB+WA)___ 0   0   |   |   |   |   | */
/*  Outer Write Cache policy (WB+NWA)__ 0   1   |   |   |   |   | */
/*  Outer Write Cache policy (WT+WA)___ 1   0   |   |   |   |   | */
/*  Outer Write Cache policy (WT+NWA)__ 1   1   |   |   |   |   | */
/*  Outer Cacheable _____________________________ 1   |   |   |   | */
/*  Outer Non-Cacheable _________________________ 0   |   |   |   | */
/*                                                |   |   |   |   | */
/*  Inner Shareable _____________________________ 1   |   |   |   | */
/*  Inner Non-Shareable _________________________ 0   |   |   |   | */
/*  Inner Write Cache policy (WB+WA)_____________ 0   0   |   |   | */
/*  Inner Write Cache policy (WB+NWA)____________ 0   1   |   |   | */
/*  Inner Write Cache policy (WT+WA)_____________ 1   0   |   |   | */
/*  Inner Write Cache policy (WT+NWA)____________ 1   1   |   |   | */
/*  Inner Cacheable _____________________________________ 1   |   | */
/*  Inner Non-Cacheable _________________________________ 0   |   | */
/*                                                              */
```

Fig. 3. Transaction Attribute for configuring Cache Memory Simulator

IV. Cache Memory Simulator Architecture

Figure 2 depicts the broad architecture of proposed Cache Memory Simulator and its various interfaces. Each initiator through its adaptor connects to L1 Cache Memory. All L1 Cache Memories through snooping mechanism based on MESI protocol connects to common L2 Cache Memory. All references to main memory through adaptor are dumped to produces references traces.

A. Cache Lookup Mechanism

Proposed Cache transaction attribute & Lookup mechanism is explained through a timing diagram in Figure 3 & Figure 4 respectively. The function d4ref(…) is called from within each cache when a cache look-up is performed. When there is a cache miss, a recursive call is made toward downstream cache through d4_dopending(…) function. This recursive call is only enabled for L2 cache towards memory (L3). All caches have a pending stack "c->pending", where all cache-miss accesses are stored. From there, and in case of L2, theses accesses will be propagated to memory. d4_dopending(…) will recursively call d4ref(c->downstream, <element of c-> pending>) element by element until c->pending stack is empty. The chronology of events concerning managing a transaction from L1 Cache Controller to memory is explained below:

- A request is made by L1 $ Ctrler core 0.
- This request is stored into L1 $ Ctrler core 1 Outstanding Request Table (ORT).
- If the attribute is "shared", cache look-up is made into L1 $ Ctrler core 1 because of the cache coherency protocol. This operation will take time: L1SNOOP_DELAY in case of hit; L1SNOOP_DELAY_MISS in case of miss.
- In the same time, L2 does its "quick" cache look-up and determine if it will be a hit or a miss
- In case of L1 $ core 1 hit, - called "snoop-hit" - operation is aborted for L2 $.
- In case of L1 $ core 1 miss (or if it was not "shared"), operation is still alive within L2 $ ORT. Moreover, it has the time attribute orttab[i].respdelay = l2dlhitor orttab[i].

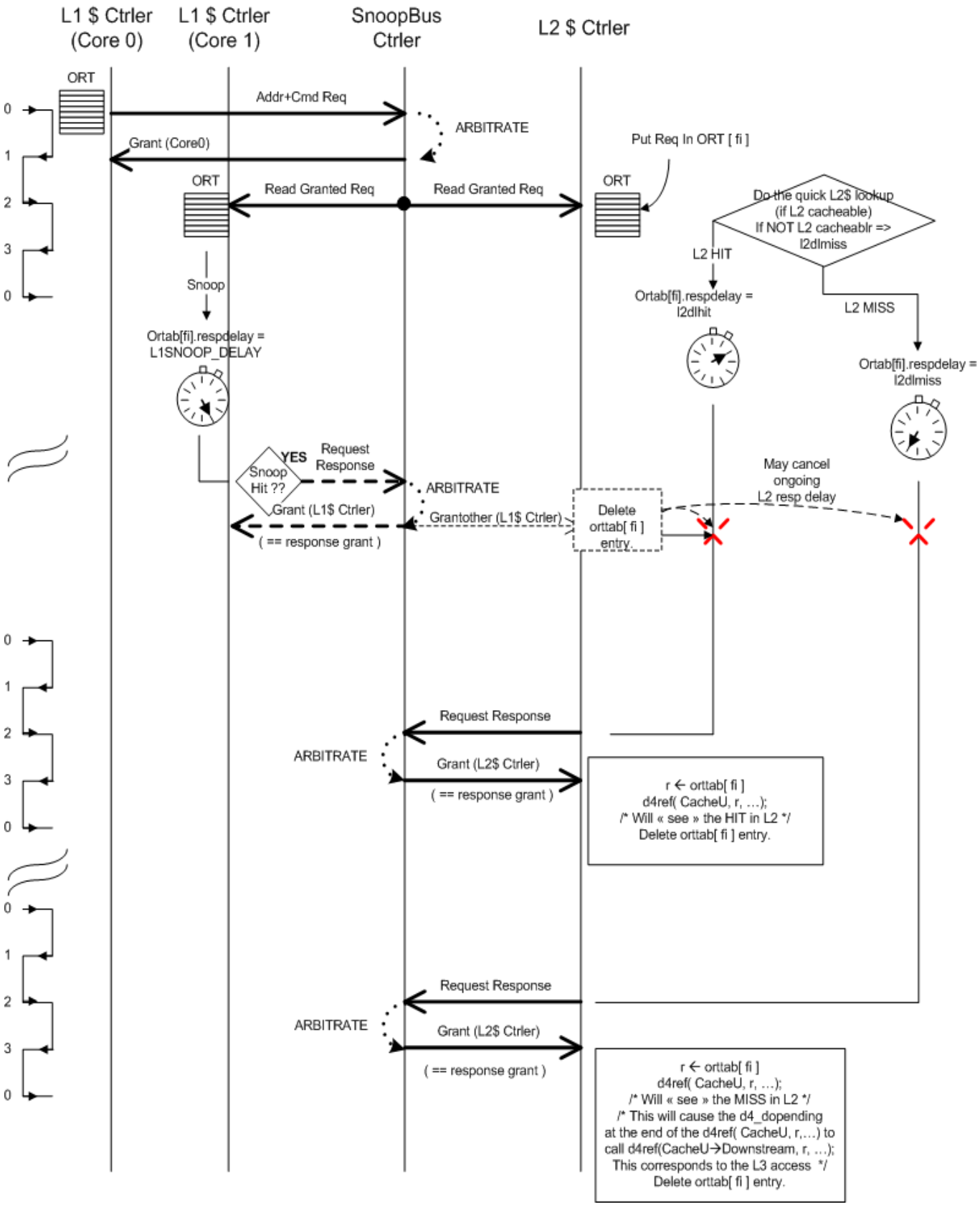

Fig. 4. Timing Diagram at Cycle Level of Cache Memory Lookup Mechanism

.respdelay = l2dlmiss.

- L2 $ Ctrler may directly responds after l2dlhit in case of hit. Then operation is deleted from its ORT.
- Or L2 $ Ctrler responds after l2dlmiss: this delay represents the time to access L3, as the access is made by a recursive call to d4ref(…). More explicitly: into L2 $, d4ref(CacheU, d4memref r,…) calls at the end of its execution and in case of miss , d4_dopending(…).
- d4_dopending calls d4ref(CacheU->Downstream, d4memref r, …). Then operation is deleted from L2 $ ORT.

V. Executing Cache Memory Simulator

The basic idea is to simulate a cache memory hierarchy connected as one or more trees, with the processors at the leaves and a main memory at each root. The various configuration parameters (architecture, policy, statistics) of each cache memory can be set separately. The configuration to be simulated is built up during initialization. After initialization, each reference is fed by a single simple function call to the appropriate top-level cache. Following command line may be invoked to launch proposed Cache Simulator:

executable_name <cache cfg file> <hardware counter file>, where cache configuration file has following characteristics:

```
2       8 50       8 1
L1_caches -l1-isize  32k       -l1-ibsize  64 -l1-isbsize  64 -l1-
iassoc  4 -l1-irepl  f -l1-ifetch  d -l1-ipfdist  0       -informat  Z
-l1-dsize  32k       -l1-dbsize  64 -l1-dsbsize  64 -l1-dassoc  4 -
l1-drepl  f -l1-dfetch  d -l1-dpfdist  0       -l1-dwalloc  a -l1-
dwback  a -informat  Z -stat-interval 150

L2_caches -l2-usize  524288  -l2-ubsize  64 -l2-usbsize  64 -l2-
uassoc  8       -l2-urepl  r -l2-ufetch  d -l2-upfdist  0       -l2-
uwalloc  a -l2-uwback  a -informat  Z

trace0.log.gz trace1.log.gz
trace2.log.gz trace3.log.gz
524288  L2-U cache "size "
```

Fig. 5. Cache Memory Simulator Configuration file

Cache Memory Simulator Cfg file (as shown in Figure 5) is made of 3 + N + 1 lines. The value of N depends on the number of cores in the SMP cache system to be simulated (1, 2, 4 ...).

- **first line**: n1 n2 n3 n4 n5 n6 //Up to 6 numbers (spaced by tabs), wherein, n1 = np: the number of processors in the SMP system (at least this parameter must always be on the 1st line at the left most position), n2 = dl2hit: the delay in cycles used by Cache Simulator to access the L2$ from the core in case of L2$-hit. If missing, L2DELAY_HIT will be used as a default delay, n3 = dl2miss: the delay in cycles used by Cache Simulator to access the L3 memory from the core in case of L2$-miss. If the parameter is missing, L2DELAY_MISS will be used as a default delay, n4 = Outstanding request table size in the L1 cache controllers, n5 = bmode: "block mode" should be set to 1, n6 = exclm: Sets the exclusive ARM mode: a data is either in L1$ or in L2$ but not both.
- **second line**: L1 cache parameters are provided.
- **third line:** L2 cache parameters are provided.
- **N following lines** is the list of trace file(s) names to be used. Obviously there are as many lines as cores instantiated in the SMP simulation. Thus we must ensure N = n1 (the left most number on the 1st line of the cache configuration file). Each of the N lines (one per core), can provide trace file names in 2 ways:

1. 2 file names per line of gzipped trace files:

   <L1_Instr_trace_core#i> <L1_Data_trace_core#i>

   Example for core #0: trace0.log.gz trace1.log.gz

   Example for core #1: trace2.log.gz trace3.log.gz

2. A single file of merged instruction and data traces, in plain text (not compressed). The format for this merged instruction/ data trace must be as following:

   type 1 pa 8 size ? attr 4 ts 8 //The numbers indicate the size of the field, i.e:

   1 27c132ac 4 0x0019 0x0013, where:

   - **type** = 0: read data, =1:write data, =2:instruction fetch, **pa** = physical address in hex format (8 nibbles; example: 27c132ac), **size** = the number of Bytes of the transfer in hex format (? = any number of nibbles), example: 2, 4, 8; values like 0x20 (32 Bytes), 0x40 (64 bytes) are allowed, **attr** = the MMU (Memory Management Unit) attribute; 4 nibbles in hex format, **ts** = the time-stamp in 8 nibbles hex format.

An example of simulation dump of data and instruction Cache indicating the various dumped parameters including Demand misses, Demand Miss Rate, Demand Fetches statistics is given in Table I & Table II. The simulation results thus produced by proposed Cache Simulator can be used to optimize Cache Memory Architecture in non-intrusive manner.

l1-icache

Metrics	Total	Instruction	Data	Read	Write
Demand Fetches	757321	757341	0	0	0
Fraction of total	1.0000	1.0000	0.0	0.0	0.0
Demand Misses	13743	13743	0	0	0
Demand miss rate	0.0183	0.0181	0.0	0.0	0.0
Multi-block refs	0				
Bytes From Memory	439776				
Demand Fetches	0.5807				
Bytes To Memory	0				
Demand Writes	0.0000				
Total Bytes r/w Mem	439776				
Demand Fetches	0.5807				

TABLE I. SIMULATION RESULTS: L1 I-CACHE MEMORY ACCESS STATS

l1-dcache

Metrics	Total	Instruction	Data	Read	Write
Demand Fetches	242661	0	242661	159631	83030
Fraction of total	1.0000	0.0	1.0000	0.6588	0.3424
Demand Misses	4248	0	4246	2097	2153
Demand miss rate	0.0175	0.0	0.0175	0.0133	0.0279
Multi-block refs	0				
Bytes From Memory	67968				
Demand Fetches	0.2803				
Bytes To Memory	40788				
Demand Writes	0.4912				
Total Bytes r/w Mem	108756				
Demand Fetches	0.4484				

TABLE II. SIMULATION RESULTS: L1 D-CACHE MEMORY ACCESS STATS

A. IP FIFO Size Correction after Cache Memory optimization

The waveforms in Figure 6 indicate the effect on IP having small FIFO sizes with high system latencies. The highlighted point shows the failure point (FIFO state going to empty) due to its high bandwidth requirement. After some tuning and FIFO size increase the FIFO level is brought to acceptable levels as shown in Figure 7.

After multiple simulation runs & with tuned cache memory parameters, the FIFO levels of all IPs are tuned to an optimized level where there is no overflow or underflow (Figure 8).

VI. FURTHER WORK

Current work was targeted only towards simulating Cache Memory for Symmetric Multiprocessing (SMP) system operating multiple cores. Simulation was carried out only till Level 2 Cache Memory. The proposed architecture, being scalable, can be extendible for Multi-level Cache Memory, however, the format for input configurable files expects parameters only for Level1 & Level2 Cache Memory which needs to be extended for Multi-level Cache parameters. The effect of current work needs to be measured for SoC architecture analysis (i.e. Bandwidth, data paths, FIFO depths,

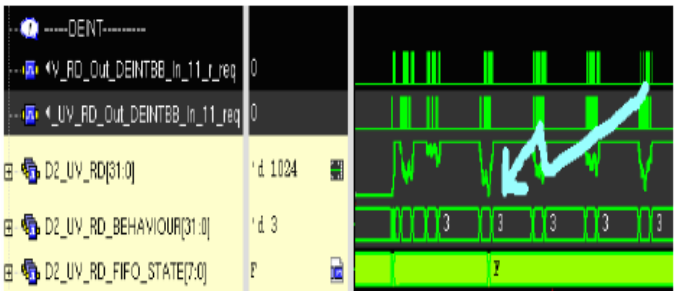

Fig. 6. Video IP simulation run –Initial paramteres

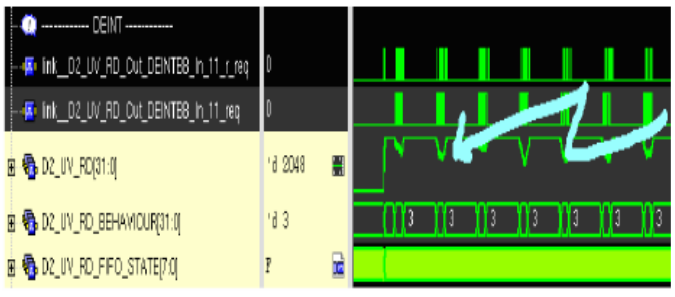

Fig. 7. Video IP simulation run- After FIFO size correction

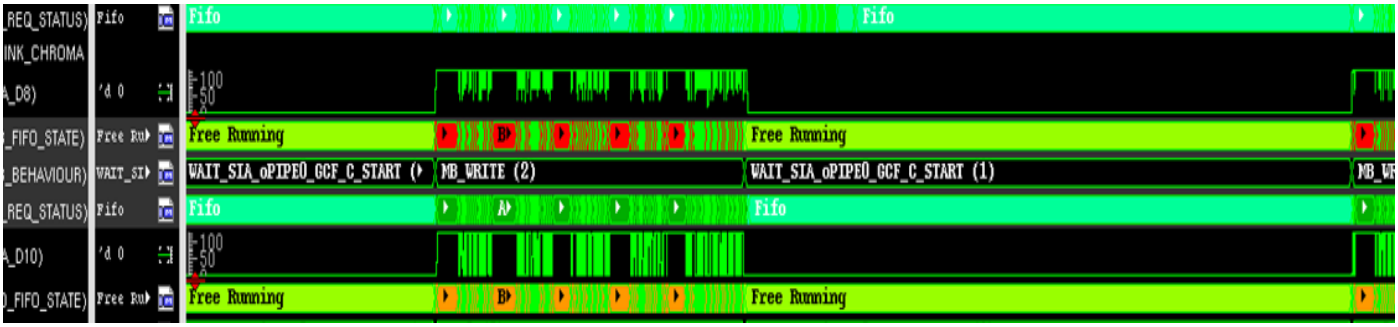

Fig. 8. SoC simulation run – after FIFO size correction

occupation, throughput, latency etc.) along with the provision of support of other Cache Coherency protocols other than MESI.

VII. Conclusion

In this paper, the importance and mechanism of simulating Cache Memory at an early design phase using timed TLM model was elaborated. Model of Computation, being TLM, had an advantage of faster simulation; however, the memory access time parameters were calculated using the introduction of timed notion in TLM Cache Memory Simulator.

Cache Memory architecture, if finalized at an early design phase, can not only save huge simulation time at RTL level but also provides further inputs to fine tune complete SoC architecture to meet the desired performance.

VIII. References

[1] N. Gupta and S. Alag, "Unified Approach for Performance Evaluation and Debug of System on Chip at Early Design Phase", In *Proc. Eighth International Conference on Contemporary Computing (IC3).* Noida, India, Aug 2015, pp. 410- 415.

[2] M. Ismail, T. Altaf and S. Mirza, "MCSMC: A new parallel Multi Level Cache Simulator for Multi Core Processors", In *Proc. of Electronics, Communications and Photonics Conference.* Fira, Saudi Arabia, April 2013, pp. 1- 6.

[3] N. Harrath, B. Monsuez and K. Barkaoui, "Verifying SystemC with predicate abstraction: A component based approach" in *Proc. of 14th International Conerence on Information Reuse and Integration (IRI).* San Franscisco, California, U.S.,Aug 2013, pp. 536-545.

[4] J. L. Hennessy and D. A. Patterson, *Computer architecture: A Quantitative Approach*, 5th ed., Elsevier, Inc., 2012, ISBN:978-0-12-383872-8.

[5] Accellera Systems Initiative, *SystemC*. Available at https://www.systemc.org, 2012.

[6] H. Khatoon, S. Mirza and T. Altaf, "Operating System Aware Cache Optimization Techniques for Multi Core Processors", in *Proc. of Frontier of Information Technology.* Islamabad, Pakistan, Dec. 2011, pp. 99- 105.

[7] D. Hackenberg, D. Molka, and W. E. Nagel, "Comparing cache architectures and coherency protocols on x86-64 multicore smp systems," in *Proc. of the 42nd Annual IEEE/ACM International Symposium on microarchitecture(MICRO)*. New York, USA, Dec. 2009, pp. 413-422.

[8] S. Pasricha, "Transaction level modeling of SoC with SystemC 2.0", In *Proc. of Synopsys Users Group Conference (SNUG)*. Bangalore, India, May 2002.

[9] A. Clouard, G. Mastrorocco, F. Carbognani and F. Ghenassia, "Towards Bridging the Precision Gap between SoC Transactional and Cycle Accurate Levels", in *Proc. of Design, Automation and Test in Europe Conference (DATE'02)*, 2002

[10] J. Edler and D. Hill, "Dinero Trace-Driven Uniprocessor Cache Simulator", Univ. of Wisconsin-Madison, U.S. Available: http://pages.cs.wisc.edu/~markhill/DineroIV/

[11] F. Ghenassia, "*Transaction-Level Modeling with SystemC: TLM Concepts and Applications for Embedded Systems*", 1st ed. Netherlands, Europe:Springer, 2006

[12] F. Bacchini et al., "Building a common ESL design and verification methodology - is it just a dream?", in *Proc. of 43rd ACM/IEE Design Automation Conference.* San Franscisco, California, U.S., 2006, pp. 370-371.

[13] D.C. Black and J. Donovan, *SystemC: From theGround Up*, 2nd. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2009

[14] W. Klingauf. "Systematic Transaction Level Modeling of Embedded Systems with SystemC", in *Proc. of Design, Automation and Test*. Europe, March 2005, pp. 566-567

[15] M. H. Jang Kang, M. Lee, K. Chae, K. Lee and K. Shim, "High-Level System Modeling and Architecture Exploration with SystemC on a Network SoC: S3C2510 Case Study", in *Proc. of Design, Automation and Test in Europe Conference and Exhibition*., Europe, 2004, pp. 538-543.

[16] S. O. Ogawa Noyer et al., "A practical approach for bus architecture optimization at transaction level", in *Proc. of Design, Automation and Test in Europe Conference and Exhibition,* Europe, 2003, pp. 176-181.

[17] G. De Micheli "Computer-Aided Hardware/Software Co-design", *IEEE Micro*, Vol 14, No. 4. August 1994, pp. 10-16.