Conquering Memory Bandwidth Challenges in High-Performance SoCs

ABSTRACT

High end System-on-Chip (SoC) architectures consist of tens of processing engines. In SoCs targeted at high performance computing such as HDTV chips, the communication bandwidth is dominated by accesses to the off-chip DRAM. Limited available bandwidth at the DRAM memory continues to be one of the most important constraints driving the design of these SoCs. Newer DRAM memories such as DDR-3 impose a minimum access granularity that can lead to memory wastage when configured to attain the bandwidth requirements of the application. In this paper, we introduce and highlight multi-channel DRAM memories in high-performance SoCs as a solution for the memory bandwidth problem, and present our communications architecture optimized for the same. We will address deadlock scenarios associated with multi-channel memories and provide mechanisms to avoid them. We compare our architecture with an industrial interconnect by experimenting with representative high definition digital TV applications. Our results indicate an average of 15% throughput improvement over the commercial interconnect solution.

1. Introduction

An SoC consists of several processing engines integrated onto the same chip. Most SoCs use one or more off-chip DRAM memories through which all major communication takes place. Figure 1 depicts a typical SoC architecture for HDTV systems[1]. For clarity, only the major blocks are included. The traffic flows from the different processing engines (henceforth called Masters or Initiators) converge into the DRAM memory. With each passing generation, the increasing performance requirements of the SoCs have resulted in increasing bandwidth requirements at the DRAMs.

Popular DRAM memories such as DDR-(1, 2 and 3) impose a minimum access granularity, which is defined by the minimum length of a DRAM burst. For example, in DDR-3, the minimum DRAM burst-length is 8. Assuming a DRAM data width of 16 bits, the DRAM fetches 16 x 8 = 128 bits (16 bytes) of data every access. This in turn takes 8 DRAM half cycles (4 cycles) of transfer between the controller and DRAM. In other words, the DRAM is accessed in a minimum of 4 cycle boundaries. If the request is less than 16 bytes, idle cycles will be introduced to cover the four cycles. Therefore, if all the bursts in the above configuration are 8 bytes long, only 50% of the available DRAM bandwidth can be utilized.

Table 1 lists the access granularity and approximate bandwidth requirement (as a percentage of the total bandwidth requirement) of the cores in the SoC for different initiators. Most of the CPU traffic, which is dominated by cache-line misses, is 32 byte or 64 byte bursts. In addition, the video decoder in most HDTV system issues 32 byte bursts per macroblock row. These two cores, along with the low activity initiators constitute about 45% of the total bandwidth requirement of the SoC. Assuming a bandwidth requirement at the DRAM of 10 GBps for a DD3-1600 (800 MHz clock) DRAM with an efficiency of about 80%, the minimum data width required will be 64 bits, the access granularity would be 64 bits (8 bytes) x 8 (Burst Length) = 64 bytes, resulting in substantial cycle wastage for the short bursts.

Figure 1: SoC with DRAM

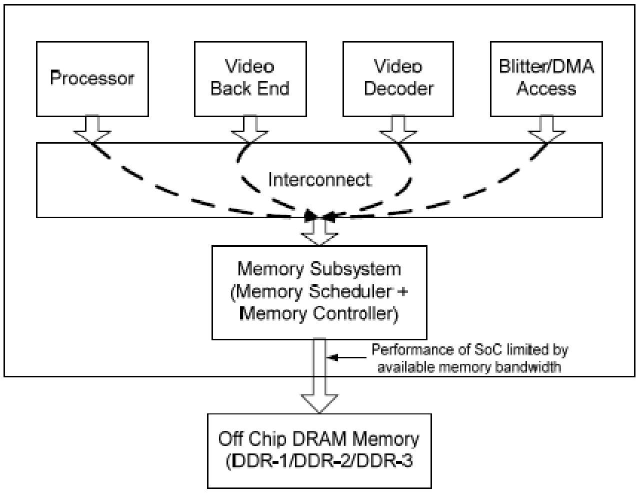

Processor
Video Back End
Video Decoder
Blitter/DMA Access
Interconnect
Memory Subsystem (Memory Scheduler + Memory Controller)
Performance of SoC limited by available memory bandwidth
Off Chip DRAM Memory (DDR-1/DDR-2/DDR-3

Table 1: Access granularities

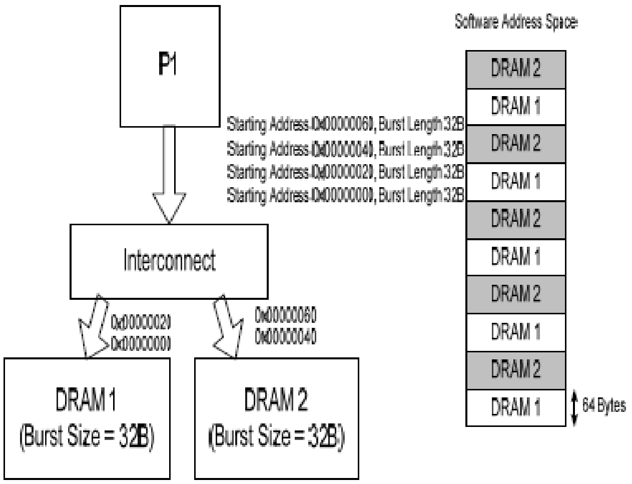
Initiator	Fetch Unit	Bytes	Approx %BW
ARM1176/MIPS 74K	8-word cache line	32/64 bytes	15%
Video-Decoder (H.264 Macroblock)	16R x 16C x 16b per pixel	32 bytes per row	25%
Video Back End	Long bursts	256 bytes	40%
Blitter	Long bursts	256 bytes	15%
Low activity initiators	Short bursts	<32 bytes	5%

The problem described above can be solved by having multi ‐ channel memory architecture, and interleaving the software address space into multiple DRAM channels. An interleaved architecture consists of mapping the entire address space into multiple physical memories. An example of memory interleaving in described in Figure 2. In this example, initiator P1 issues 32 byte incrementing bursts starting from address 0x00000000. The software address space is interleaved at 64 byte boundaries. Assuming a combined bandwidth requirement of 10GBps from the two x32 DDR-3 DRAMs, the access granularity would be 32 bytes, instead of 64 bytes in the case of single memory channel. Thus, idle DRAM cycles can be avoided.

The page-miss overhead in DRAM systems have not scaled with increasing DRAM frequency. As a result, in terms of efficiency calculated as the percentage of the peak bandwidth, the page misses tend to hurt the higher frequency DRAMs more than lower frequency

DRAMs. Multi-channel memories also help in reducing the effect of page misses. They provide better bank level parallelism (8x2 banks for a 2 channel subsystem), and can amortize the page misses by distributing the traffic among the multiple channels.

Figure 2: Interleaved memory architecture



SoC designers have a choice of implementing memory interleaving in software or hardware. Hardware-Software decoupling is a major goal for SoC designers. Software written for a SoC with only one memory should be re-usable in a SoC with two or more DRAMs, with little or no modification. This motivates the need for the implementation of the memory interleaving in hardware. A memory interleaving architecture should address the following: 1) The architecture should be scalable to the increasing complexity of the interconnection network. 2) The architecture should be optimized for area and performance. 3) Due to ordering restrictions at the network interface, deadlocks might occur. The architecture should ensure that deadlocks are avoided. 4) The proposed architecture should be configurable to cater to different application domains and memory subsystems.

In this paper, we propose an on-chip communications network architecture that performs, in addition to routing, memory interleaving for multi-channel memory architecture. Our solution incurs minimal area overhead on the interconnection architecture. We address the deadlock situations associated with memory interleaving and provide solutions to avoid the same. The architecture is completely software transparent, and highly configurable. Our performance analysis environment helps designers configure an optimized interleaved architecture in a very short time.

The following is the summary of contributions of this paper:

- An on-chip network architecture that supports software transparent memory interleaving. For deadlock avoidance, the architecture uses the buffers already available in the on-chip communications system. Thus, it is extremely area and performance-efficient. The architecture scales extremely well for increasing the number of memory channels and initiators.
- Algorithms for deadlock avoidance caused by ordering requirements at the network interface.
- Performance analysis environment for fast architectural exploration and tuning.

The paper is organized as follows. In Section 2 we discuss existing work in the context of our contributions. In Section 3, we present the on-chip communication architecture with focus on multi-channel memory interleaving. In Section 4 we describe our performance analysis environment. In Section 5 we present experimental results, and finally in Section 6, we conclude the paper with pointers to future directions.

2. Previous Work

In the past, several researchers have addressed the problem of interconnection architecture design for multi-core SoC systems[2,3,4,5,6] A quantum of work has also focused on communication architecture synthesis problems[7,8,9,10]. In all these papers, the traffic model is considered at a very high level of abstraction, such as bandwidth and latency requirements between pairs of communicating cores. Deadlock avoidance has been limited to those caused by cycles in the channel dependence graph of the network. In [11] Hansson et al discussed the topic of message dependent deadlock avoidance. Similar to the method proposed in the paper, we also separate the request and response networks to avoid message dependent deadlocks. Their work however, does not address deadlocks arising from ordering dependencies in multi-channel memories, which occurs due to asymmetric network structure, and not because of shared buffers in the request and response paths. Kwon et al. [12] presented a technique to avoid deadlocks due to the ordering requirements at the network interfaces. Their solution is based on incorporating expensive re-order buffers at the interface. In terms of area, the complexity of their solution increases linearly with the number of initiators and as a quadratic function with the number of memory channels. On the other hand, our solution uses the existing buffers in the on-chip network, and therefore, is extremely area efficient. Moreover, the solution is a completely distributed architecture and is only limited by the network size in terms of scalability.

3. Interleaved Memory Architecture (IMA)

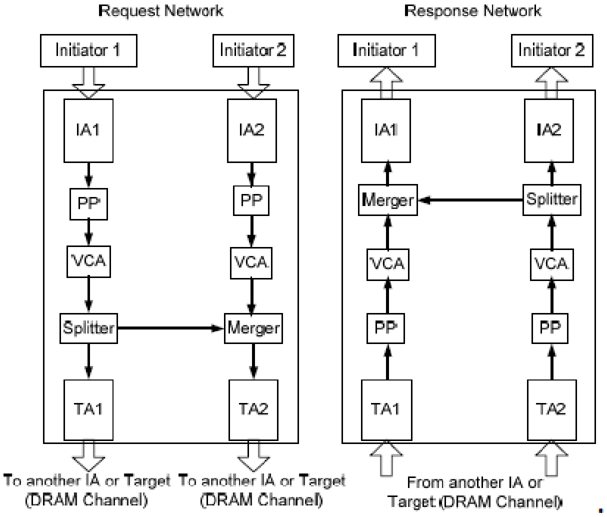
In this section, we present our interleaved memory architecture based on-chip interconnection networks. For these purposes, we restrict our discussion to the architecture optimized around the interleaving of the software address space. Otherwise, we assume a general interconnection network, which performs routing of packets from source to destination.

3.1 Overview of the On-Chip Network

Figure 3 depicts the interconnection architecture utilized in this paper. The network consists of separate request and response phases. The request phase network routes the Read and Write requests from the initiators (processors, video decoder etc) to the targets (DRAM memory channels). In the request phase, address decoding and other functions such as data width conversion and burst chopping are performed by the initiator inter-connect agent (IA). The IA also performs memory channel interleaving as part of the address decoding process. The pipeline point (PP) is a flop that adds a cycle of latency for the request packets. The pipeline point can be composed of virtual channels. The virtual channel arbitration units (VCA) arbitrate among multiple incoming virtual channels to route packets to the outgoing virtual channels, in a time division multiple access (TDMA) manner. The splitter units split the traffic on an incoming link into multiple target links. The merger units merge traffic on different links into one outgoing link.

The response phase network routes responses coming from the DRAMs to their respective initiators. The target inter ‐ connect agent (TA) performs the functions such as determining the initiator for the response and response width conversion. Similar to the request phase network, the response phase network also consists of VCA, splitter, merger and pipeline points.

Figure 3: Interconnect Architecture



The example provided in the figure assumes that IA ‐ 1 communicates with two channels, and has a splitter in the request path, and a merger in the response path. IA ‐ 2 communicates exclusively with TA ‐ 2. In the response path, the response from TA ‐ 2 is split at the splitter and is merged with responses from TA ‐ 1 before reaching IA ‐ 1.

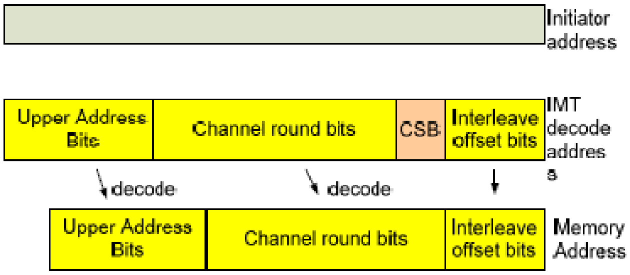
We refer to the IA, TA, PP, Merger, Splitter and VCA as the basic components of the interconnection network. We make no assumption on the topology of the interconnection network, both in terms of the number of basic components and their connectivity. The request and response networks need not be symmetrical. For example, if the response path is not logically complex, a PP component may be removed to minimize cycle latency. Any interconnection architecture can apply the techniques presented in this paper to support deadlock free multi‐channel interleaving. In fact, our interconnection network is generated automatically by a derivation algorithm, based on initiator and target connectivity requirements.

3.2 Support for Interleaving

Memory interleaving is performed at the initiator‐interconnect agent. The technique is depicted in Figure 4. Interleave offset bits denotes the size of the interleave. The channel select bits (CSB) determine the memory channel to which the request should be sent. IMA removes the CSB from the incoming address, and sends the rest of the address downstream to the memory determined by the channel select bits. In the following paragraphs, we present the techniques employed by the architecture for scalability, area and performance, configurability and deadlock avoidance.

Scalability: Typical high performance SoCs are composed of several tens of initiators communicating with relatively few DRAM channels. IMA performs the memory channel interleaving at the IAs, which are furthest away from the memory channels. By doing this, IMA avoids hotspots that would otherwise occur if a centralized interleaving module was placed close to the memory channels, thus avoiding place and route problems.

Figure 4: Channel Interleaving



Area and Performance: IMA uses minimal hardware for memory interleaving. The hardware overhead is only about 8 bytes per IA virtual channel. On the other hand, existing solutions ([12] for example) easily consume several hundred bytes per virtual channel. Our performance analysis environment helps designers to configure IMA to optimize for the desired performance goals.

Configurability: The CSB location determines the interleave size and is runtime configurable. The user can choose any bit position for the locations by writing the value into a memory mapped register, which is used by the router to decide the channel for the burst. This is particularly useful during mode changes in the video application, where the address sequence of the transactions change. Making the CSB location configurable also helps in the reuse of the interconnection architecture when the external memory architecture changes. The width of the CSB determines the number of channels present in the design, which also can be programmed to a new value, if the memory architecture changes for multiple designs.

Deadlock Avoidance: An interconnection architecture that supports memory interleaving as described above must address the problem of deadlock that is caused due to the ordering requirements network interfaces[12]. As mentioned before, the solution presented in this paper does not use expensive re-order buffers, thus minimizing chip area, power and timing. The deadlock problem is described in Figure 5. Due to the asymmetric delays in the path from initiator P0 to the two DRAM memories, request B1 reaches

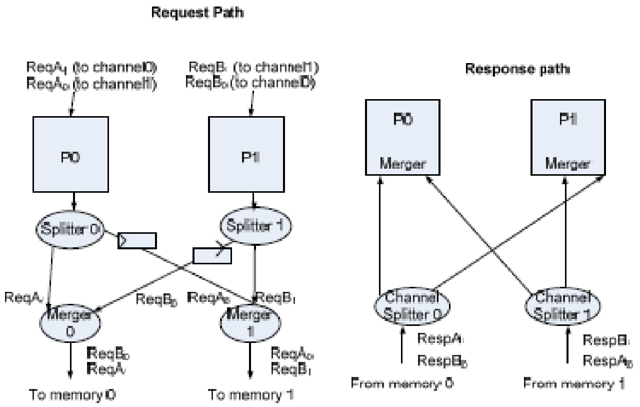
DRAM-1 before request A0. Similarly, request A1 reaches DRAM-0 before request B0. The ordering requirements block the response to request B1 from coming back to P1 before the response of B0. The response for request A1 is also blocked as it cannot reach P0 before A0. This causes a deadlock as A1 is waiting for A0, which is behind B1, and B0 is waiting for B1, which is behind A0. We denote this type of deadlock as "Response Network" deadlock. The deadlock avoidance mechanism consists of a request path mechanism called "Acknowledgement Mechanism" and a response path mechanism called "Response Reorder Mechanism". In the following sub-sections, we describe the two mechanisms in detail.

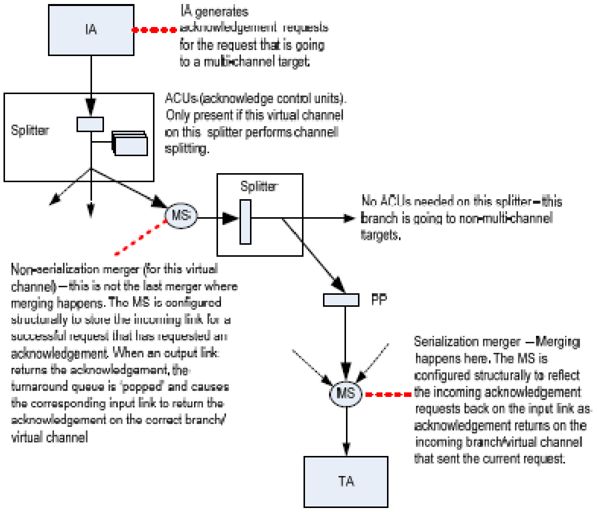
3.2.1 Request Path Acknowledgement Mechanism

In order to prevent the Response Network Deadlock described above, IMA applies an "Acknowledgement Mechanism". The acknowledgement mechanism generates confirmation from the last merge point in the request network at which two connections merge transfers. This information confirms that the channel requests from different connections have been serialized. The last merger will be referred to as the "serialization point". Therefore, if each sender had to wait until the first request reached the channel merge point before sending the next one, then no deadlock could have occurred. The acknowledgement mechanism is propagated upwards to all points in the request network that split the requests into multiple memory channels. If the channel splitter and the last serialization point exist within the same cycle boundary (i.e., there are no registers between them) then no explicit acknowledgement signals are needed - the acceptance of a transfer on the link between the channel splitter and channel merger can also be used to indicate acknowledgement.

Figure 6 depicts an example of a path where the acknowledgement mechanism starts at the IA component shown at the top-left corner, along the path of the packet, and ends at the serialization point channel merger shown at the bottom of the figure. The IA generates an acknowledgement request on the first transfer (flit) of any interleaved burst that leaves the IA, which is going to a multi-channel target. Inside the splitter an acknowledge control unit (ACU) is added. The ACU prevents requests going to a multi-channel target (e.g., the first transfer of such burst has its acknowledgement signal set to 1) from proceeding if the outgoing splitter branch changes from that of the previous transfer and there are no outstanding acknowledge requests for which acknowledgements have not been received. There is at most one ACU for each virtual channel (input) at the splitter.

Figure 5: Response Network Deadlock





IA generates acknowledgement requests for the request that is going to a multi-channel target.

ACUs (acknowledge control units). Only present if this virtual channel on this splitter performs channel splitting.

No ACUs needed on this splitter—this branch is going to non-multi-channel targets.

Non-serialization merger (for this virtual channel)—this is not the last merger where merging happens. The MS is configured structurally to store the incoming link for a successful request that has requested an acknowledgement. When an output link returns the acknowledgement, the turnaround queue is 'popped' and causes the corresponding input link to return the acknowledgement on the correct branch/virtual channel

Serialization merger —Merging happens here. The MS is configured structurally to reflect the incoming acknowledgement requests back on the input link as acknowledgement returns on the incoming branch/virtual channel that sent the current request.

The acknowledgement requests travel in-band with the flit. At some point the flit with the acknowledgement request will reach the serialization merger - this is the last point where the connection merges with another connection on the same merger (outgoing) virtual channel. If the transfer wins arbitration at the merger, the merger will return the acknowledgement back upstream on the same request path. These acknowledgement signals do not encounter any backpressure or have any flow control. Wherever there is a pipeline point, the signals will be registered. The acknowledgement signals are used at each channel splitter ACU along the path. The ACU keeps a count of outstanding acknowledgements. When an acknowledgement returns, the ACU will decrement its count of outstanding acknowledgements. The acknowledgement propagates back to the first channel split point in the request network. For the example shown in Figure 6, this first channel split point is at the splitter just downstream to the IA

3.2.2 Response Path Reorder Mechanism

The response path must ensure that the ordering is maintained between the responses from different targets to requests originating from the same initiator virtual channel. This is achieved by passing information for a request path channel splitter component to the corresponding response path channel merger component. The information is passed via a turnaround queue, which maintains FIFO order. The information passed over tells the merger component which incoming branch the next response burst should come from. The MS component applies backpressure to all branches that map to the same outgoing thread, except for the one indicated by the turnaround queue. When the burst completes the turnaround queue entry is popped. This mechanism ensures that all responses are returned in the correct order.

Figure 6: Acknowledgement Mechanism Implementation

In this paper, we assume that that the core to network interfaces follow a standard protocol such as the Open Core Protocol [13]. Hence, if the underlying memory controller issues requests out of order, it must re-order the responses before presenting them to the interconnect. However, since the number of memory channels is typically much smaller than the number of initiators, and because this problem is localized to individual DRAM channels, the amount of re-order buffers required will be minimal.

4. Simulation Based Performance Analysis

In this section, we present our simulation and performance analysis environment. We have used SystemC models of the architecture for experimentation and performance analysis. Our motivation for using SystemC models is two fold: i) The simulation can run an order of magnitude faster than the corresponding RTL, and ii) we can inject software objects called performance observation points (POP) for architectural analysis of the design. The POPs are especially useful in helping the users obtain the best architectural trade-offs in designs with a high degree of configurability.

Our system is composed of a set of transactors (traffic generators), an interconnection network, and a memory subsystem. The transactors mimic different processing engines. The processing engines are categorized into processors, video decoders, etc, and inject traffic accordingly. The interconnection network performs the traffic routing from the transactors to the memory subsystem. The memory subsystem is composed of a memory scheduler, a memory controller, and a DDR based DRAM system. The DDR memory can be configured to be a DDR1, DDR2 or a DDR3 memory, with user defined frequency and data width.

We have developed a cycle approximate Transaction Level-1 (TL-1) SystemC model of the interconnection network. The transactors and memory subsystem are modeled as a Transaction Level-2 (TL-2) SystemC model, which is at a higher level of abstraction than the TL-1 model. Since the focus in this paper is the on-chip communications network (or interconnect), we preferred a more accurate model for the same. The TL-2 model at the transactors and memory speeds up the simulation considerably. Layer adapters have been put in place for seamless communication between the TL-1 and TL-2 models based on the Open Core

Protocol (OCP)[13] specification. Based on black box testing (monitoring all signals into IMA and signals out of IMA), the SystemC models are 99% accurate with respect to the corresponding RTL.

4.1 Performance Analysis

We divided the performance analysis of the interconnection architecture into two categories: static analysis and runtime analysis. Under static analysis, we analyze the traces to determine the channel interleave size that maximizes performance. Under runtime analysis, we use data collected from the simulation to apply our analysis algorithms to make architectural choices.

Static analysis: Choice of the channel interleave bits plays a significant role in achieving the required system performance. It is important to make sure that all the memory channels are balanced, such that maximum channel level parallelism can be obtained. The amount of memory channel balance that can be obtained also depends on the address pattern of the application. Therefore, the designer needs to use the address pattern of the application, and experiment with multiple channel interleaving combinations to arrive at the final channel interleaving configuration.

We present a systematic way of statically analyzing the address pattern and the configuration options to achieve the best system performance. Our approach is based on the fact that it takes far less time to do a static analysis on address patterns instead of running simulations for millions of cycles, which is extremely time consuming.

For static analysis, the designer runs one simulation to obtain approximate timestamps at which each initiator injects transactions into the system. This simulation is run with a single memory architecture that forms the baseline of the analysis. Now, based on the injection time, the transactions are binned into time windows. Note that choosing the correct sized bin is important. Large bins allow significant imbalances to average out. Small bins highlight imbalances hiding behind queuing delays. Considering that the initiators have in-built fifos which are flushed into the DRAMs when they are filled, a typical bin size would be the time taken to fill the buffers assuming a zero cycle delay in the interconnect.

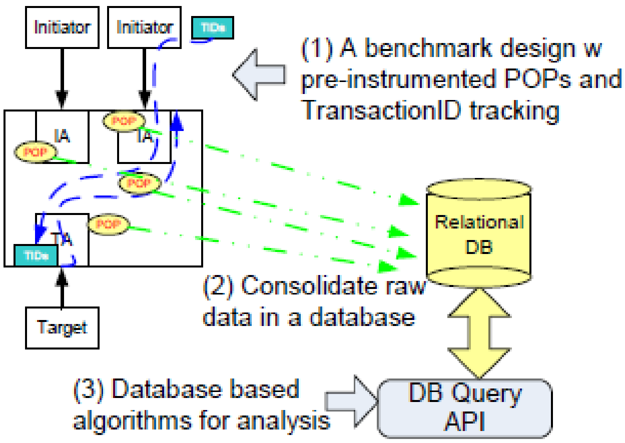
Once the bin size has been chosen, our algorithm minimizes the root mean square error between the numbers of bytes going to the different channels over the different time windows. Mathematically, the problem can be specified as: *Minimize Sqrt (Σ(Pni - Pnj)2) where*

T = total number of clock cycles in the trace divided into N equal bins of size T/N such that
T = {T1, T2 = T1 + T/N, T3 = T1 + 2T/N ….}

Pni is the number of bytes addressed to channel i at bin Tn and Pnj be the number of words addressed to channel j at bin Tn. IMA restricts the location of channel interleave bit to be between 6 and 16. Therefore, currently, we explore the entire design space to obtain the best combination.

Runtime analysis: Our runtime analysis environment is depicted in Figure 10. We instrument the design with pre-defined software objects called performance observation points (POPs). These objects collect performance specific data at different points in the interconnection architecture, and dump them into a database. In order to limit the amount of data that is collected, the software objects use a sampling window, which can be modified at runtime by a SystemC process or thread construct. The software objects also assign a unique transaction ID to the bursts entering the system, which helps in correlating the transactions across the different points in the interconnect.

Figure 7: Performance Analysis Environment


Initiator
Initiator
TIDs
(1) A benchmark design w pre-instrumented POPs and TransactionID tracking
IA
POP
IA
POP
POP
TA
POP
TIDs
Relational DB
(2) Consolidate raw data in a database
Target
(3) Database based algorithms for analysis
DB Query API

Once the performance data has been stored in the database, our environment provides pre-defined algorithms for performance analysis. Some of the algorithms are simple such as determining the average queue occupancy over time window, while others are more complex such as analyzing the average cost of a page-miss in the DRAM. Our environment allows the designer to use the pre-defined algorithms are primitives to build more application/design specific algorithms.

5. Experimental Results

We present results for two HDTV SoC benchmarks. The first HDTV benchmark called DTV-1, is composed of processor, a blitter (graphics processing engine), a video decoder, a video back-end engine, an audio processing unit, and other low activity initiators. The memory bandwidth requirement of the application is about 5 GBps. The second HDTV benchmark called DTV-2, is composed of a processor, three DSP engines that perform vector processing, a video decoder, a video back-end engine and other low activity initiators. The bandwidth requirement of the DTV-2 benchmark is about 10GBps.

The traffic profiles for DTV-1 were derived from the application running on an emulator. In DTV-2, the temporal distribution of traffic was in accordance to the HDTV requirements. However, the starting addresses of transactions were random. We used DDR-3 1600 (800 MHz clock) memory parts with DRAM burst-length of 8 in our simulations. Based on the bandwidth requirements, we used 2 x16 DRAM parts for DTV-1, and 4 x16 DRAM parts for DTV-2.

We compared our results against three different configurations: i) A commercial on-chip communications solution from Sonics [14] called SMX with one memory channel (SMX-1), ii) SMX with two memory channels (SMX-2), iii) SMX with two memory channels such that the two channels are used by different dedicated virtual channels (SMX-3). In SMX-1, the memory data width was set at twice that of SMX-2, SMX-3 or IMA. Therefore, under perfectly ideal conditions, SMX-1 should perform equal or better than IMA. In SMX-2, an outstanding transaction to a second memory channel is back-pressured until the previous request to the first memory channel has been completed. SMX-3 overcomes the problem of SMX-2 by having separate OCP threads (dedicated virtual channels) at the initiator for each memory channel. This removes the ordering requirements. In SMX-3, reordering buffers will be required inside the initiators, which will incur prohibitive area overhead. Hence, SMX-3 is feasible only for small designs.

5.1 Summary of Results

Figure 8 compares the achieved bandwidth for the four experiments in the DTV-1 case. The left most bar in the figure denotes SMX-1, and all other bars are normalized to this bar. The second bar denotes SMX-2, the third bar denotes SMX-3, and the fourth bar denotes IMA. The X-axis denotes the different interfaces at which the bandwidth was measured. The overall memory bandwidth for IMA is 8% more than SMX-1, 9% more than SMX-2. This is a significant result, considering that a difference of 5% or more can result in the application not meeting performance goals, thus requiring expensive addition DRAM resources. As expected, SMX-3 performs better than IMA. However, this result is only of academic interest, as the area cost of SMX-3 is prohibitively expensive, especially as designs get bigger. Reordering buffers are of the order of 512 bytes in size to account for an entire macroblock of data. If a design has 200 initiator virtual channels, the amount of reordering required would be in the order of 100 Kbytes. Assuming 10 gates per flop, this will result in a million gates, which is not scalable at all.

Figure 9 compares the achieved bandwidth for the four experiments in the DTV-2 case. The organization of the figure is similar to Figure 8. In this case, the overall memory throughput for IMA was about 17% more than SMX-1 and 20% more than SMX-2. Again, as expected, SMX-3 performed best in terms of performance, but with a high gate count cost.

6. Conclusion

In this paper, we demonstrated the need for interleaved multi-channel memory architectures, and provided our solutions for the same. We presented our Interleaved Memory Architecture (IMA) that addresses several challenges in SoC design with multiple interleaved memory channels. We addressed deadlock causing scenarios and provided scalable and area efficient solutions for the same. We experimented with representative HDTV benchmarks, and compared our results against Sonics' commercially available interconnect called SMX. In conclusion, Sonics offers SoC designers multiple architecture choices - SMX is a strong and efficient solution for existing DDR2 architectures and IMA allows an upgrade path for new video SoCs, especially those that will move to DDR3 and require scalability.

References

Figure 8: DTV-1 Bandwidth Comparisons

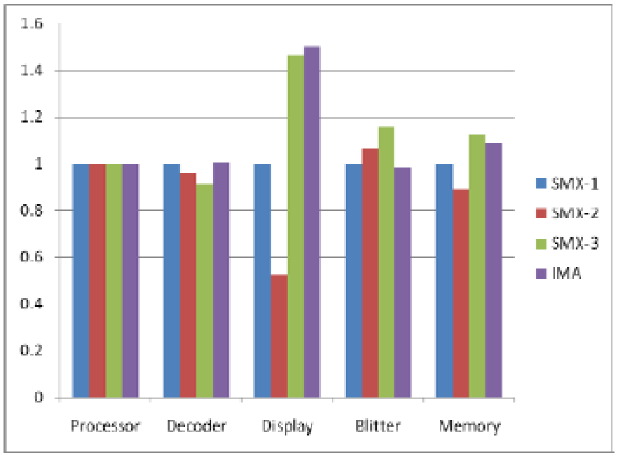
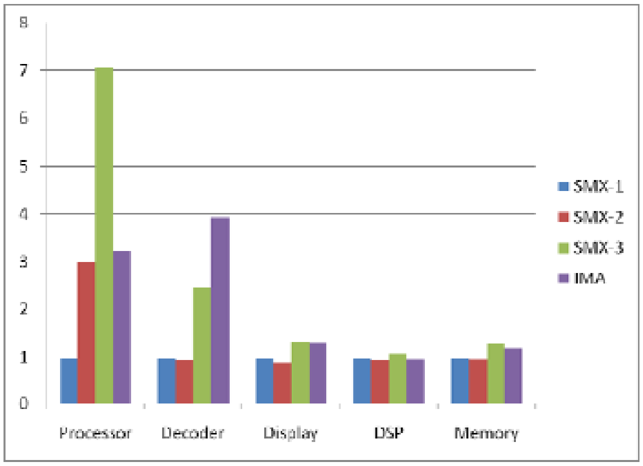


Figure 9: DTV-2 Bandwidth Comparisons



[1]Chambonniere et al, "MPEG Audio Video Decoder with 2D Graphic Engine for DTV", ICCE 2000

[2]Vellanki et al, "Quality of Service and Error Control Techniques for Mesh Based NoCs", Integration, VLSI Journal, Jan 2005

[3]SteenHoff et al, "Network on Chips for High End Consumer Electronics TV Systems Architectures", DATE 06

[4]Marescaux et al, "Introducing the SuperGT Network-on-Chip", DAC 07

[5]Lu et al, "Layered Switching for Networks on Chip", DAC 07

[6]Ogras et al, "Voltage-Frequency Island Partitioning for GALS-based Networks-on-Chip

[7]Srinivasan et al, "Linear Programming based Techniques for Synthesis of Network-on-Chip Architectures", IEEE TVLSI, 06

[8]Chan et al, "NoCOUT : NoC topology generation with mixed packet-switched and point-to-point networks, ASPDAC 2008

[9]Pasricha et al, "FABSYN: Floorplan Aware Bus Architecture Synthesis", IEEE TVLSI, 2006

[10]Murali et al, "A Methodology for Mapping Multiple Use Cases onto Networks-on-Chips", DATE 2006

[11] Hansson et al, "Avoiding Message-Dependent Deadlock in Network-Based Systems on Chip", Hindawi Publishers, VLSI Design, Volume 2007, Article 95859

[12]Kwon et al, "A Practical Approach of Memory Access Parallelization to Exploit Multiple Off-chip DDR Memories", DAC 2008
[13]Open Core Protocol, International Partnership, http://www.ocpip.org/home
[14] Sonics Inc www.sonicsinc.com