On the Performance of Bus Interconnection for SOCs

Liqiang Zhang
Department of Computer Science
Wayne State University
liqiang@cs.wayne.edu

Vipin Chaudhary
Institute for Scientific Computing
Wayne State University and Cradle Technologies, Inc.
vipin@wayne.edu

ABSTRACT

The choice and design of communication architecture are critical for SOC design. The communication architecture may heavily influence the overall performance, and also determines the salability of the whole system. Among several communication architectures proposed for SOCs, shared-bus has been widely used. Some doubts are cast on this approach today, as it is likely to be the bottleneck for the current and future SOCs. In this paper, we use analytical model based simulator and abstracted traces of image processing applications to simulate the Global Bus architecture of Cradle's Universal Micro System (UMS). Our simulation shows that (i) with carefully designed protocols, shared-bus is efficient enough for current and future high performance SOCs; (ii) instead of the shared-bus, the SDRAM is more likely to be the bottleneck, especially for streaming applications. For the applications in our experiments, the SDRAM gets saturated as early as 30% of the Global Bus utilization. We also explore the relationship between the depth of buffers and the performance of the Global Bus in our work.

1. INTRODUCTION

Modern system-on-chip (SOC) design shows a clear trend towards integration of multiple processor cores. The SOC System Driver section of the "International Technology Roadmap for Semiconductors" (http://public.itrs.net) predicts that the number of processor cores will increase dramatically to match the processing demands of future applications. While network processor provider like IBM, embedded processor provider like Cradle have already detailed multi-core processors, mainstream computer companies like Intel and Sun have also discussed such an approach for their high-volume markets. In Dec. 2001, both Intel and Sun laid their first plan to deliver multiprocessing computers on a chip [1]. It is believed in very near future, Chip Multiprocessors (CMP) will replace all the single-core processors.

Typical chip multiprocessors consist of processing cores, on-chip cache hierarchy, interconnection architecture, and channels to external memory. While a large body of research on system synthesis has focused on scheduling, partitioning, and mapping the target application functionality to an optimal set of system components, often equally important is the choice and design of the communication architectures for the CMPs. The communication architecture determines the way in which the components communicate with each other to synchronize and exchange data. With the increasing complexity of the SOCs and computation power of the processing cores, more care has to be taken in the topological choice, protocol design, and parameter tuning for communication architecture to avoid it from becoming a bottleneck of the whole system. Current proposals for communication architectures for CMP include shared-bus, ring-based, multi-channel, TDMA-based, on chip crossbar, etc. Such networking technologies are "micronized" in SOCs to create the on-chip networks.

Performance evaluation of the above proposals for SOC communication architectures is obviously important for strategy decision at the system design stage. Research shows that while all of the proposals for communication architecture have their pros and cons, none of them uniformly outperform others—their performance highly depends on the specific application domains [6][7][8]. In this paper, we focus primarily on the performance analysis of shared-bus communication architecture for SOCs, especially the case study for the Global Bus architecture of Cradle's Universal Micro System (UMS).

In this study, we use a simulator based on the analytical model of the UMS Global Bus and abstracted traces of image processing applications to give a fast performance analysis for the Global Bus architecture. Our simulation shows that:

- With carefully designed protocols, shared-bus is efficient enough for current and future high performance SOCs.
- Instead of the shared bus, SDRAM is more likely to be the potential bottleneck, especially for streaming applications.
- The size of the buffers on the shared bus can be optimized by simulation. Experiments give the reference value for both the Quad and SDRAM interface buffer size, and show that larger size produces no benefit.

The rest of this paper is organized as follows. Section 2 gives a brief introduction to UMS with an emphasis on the Global Bus architecture. Section 3 describes our fast simulation strategy in detail. Section 4 gives the experiment data and analysis. Section 5 reviews related work done on performance evaluation of SOC communication architectures. Finally, section 6 summarizes this paper.

2. HARDWARE ARCHITECTURE OF UMS AND THE GLOBAL BUS

Cradle's Universal Micro System (UMS) architecture consists of dozens of high performance, RISC-like and digital signal processors on a single chip with fully software programmable and dedicated input-output processors. The processors are organized into small groups, with eight digital signal processors

and four RISC-like processors each sharing a block of local data and control memory, with all groups having access to global information via a unique on-chip bus—the Global Bus. It is the fact that data, signal and I/O processors are all available on a single chip, and that the chip is thereby capable of implementing entire systems, which gives rise to the designation "Universal Micro System". The block diagram is shown as Figure 1.

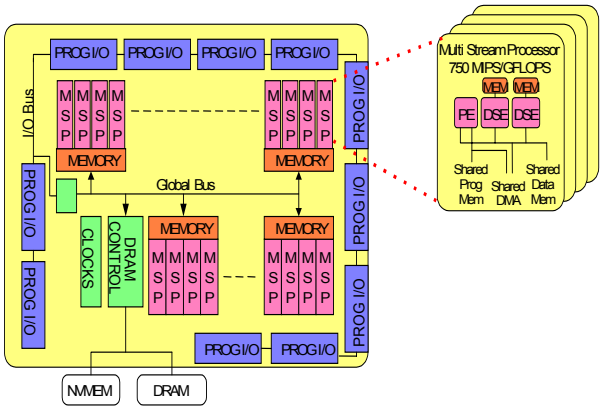

Figure 1: UMS Block Diagram

The UMS is a shared memory MIMD (multiple instruction / multiple data) computer that uses a single 32-bit address space for all register and memory elements. Each register and memory element in the UMS has a unique address and is uniquely addressable.

2.1 Quads

The Quad is the primary unit of replication for UMS. A UMS chip has one or more Quads, with each Quad consisting of four RISC-like processors called Processor Elements (PEs), eight DSP-like processors called Digital Signal Engines (DSEs), and one Memory Transfer Engine (MTE) with four Memory Transfer Controllers (MTCs). The MTCs are essentially DMA engines for background data movement. Within a Quad, PEs share 32KB of instruction cache and 64KB of data memory, out of which 32K can be optionally configured as cache. Thirty-two semaphore registers within each quad provide synchronization between processors. Figure 2 shows a Quad block diagram. Note that the Media Stream Processor (MSP) is a logical unit consisting of one PE and two DSEs.

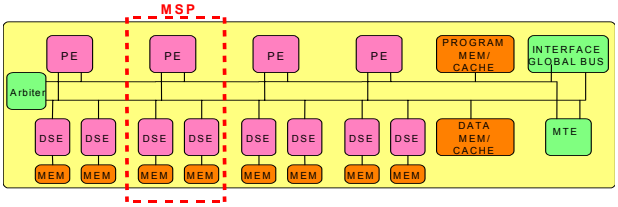

Figure 2: Quad Block Diagram

Processing Element—The PE is a 32-bit processor with 16-bit instructions and thirty-two bit registers. It has a RISC-like instruction set consisting of both integer and IEEE 754 floating point instructions. The instructions have a variety of addressing modes for efficient use of memory. The PE is rated at approximately 90 MIPS.

Digital Signal Engine—The DSE is a 32-bit processor with 128 registers and a local program memory of 512 20-bit instructions optimized for high-speed fixed and floating point processing. It uses MTCs in the background to transfer data between the DRAM and the local memory. The DSE is the primary compute engine and is rated at approximately 350 MIPS for integer or floating point performance.

2.2 Global Bus

The UMS Global Bus (GBus) is a 64-bit high speed bus. It uses uniform addressing with a single 32-bit address for all bus elements and transfers data with 64 bit width. The Global Bus has two 32-bit address spaces: a *data space* and a *control space*. The *data space* is the normal address space for data transfer and computation. The *control address space* is for GBus device configuration registers, control commands, and debug access.

Data is transferred between bus masters and bus targets. Bus masters initiate the transactions, which are completed by the targets. Bus masters are the PEs, DSEs and the MTCs, while bus targets are the SDRAM, host interface registers, internal cache and memories, and global registers, etc. Both masters and targets have their FIFOs and interfaces. Each GBus interface has a unique hardware assigned device number that identifies itself for *receiving data*. The 16 bits device number is wired within the GBus wires (besides the 64 Address/Data lines). A device number of zero selects all devices to receive the data. Each target interface also has a range of GBus addresses, called My Global Address Range that identifies the addresses to which the target will *respond*. Every master or target interface also has two FIFOs for data write and data read. Figure 3 shows the GBus block diagram.

2.2.1 Bus Transactions

The Global Bus is a single transaction write, split transaction read bus. It has four types of transfer types: data write, data read, control write, and control read.

A *data write* operation by a bus master sends the transfer command in the first bus cycle, followed by one or more data octets in the following cycles. The transfer of the data to the bus completes the write cycle. A *data read* operation by a bus master sends the transfer command in the first bus cycle, and then releases the bus. The targeted device receives the command. When the read data is ready, the target arbitrates for the bus and sends the read data to the bus master indicated in the command octet. The transfer of the data to the requesting device completes the read cycle.

A *control write* is an address variant of a data write operation with a single data octet: it writes data to the control address space. All targets receive the command and data octet, completing the cycle. Control writes are used to send the base addresses to each Quad, and to send base addresses and configuration data to all other GBus devices such as the Global

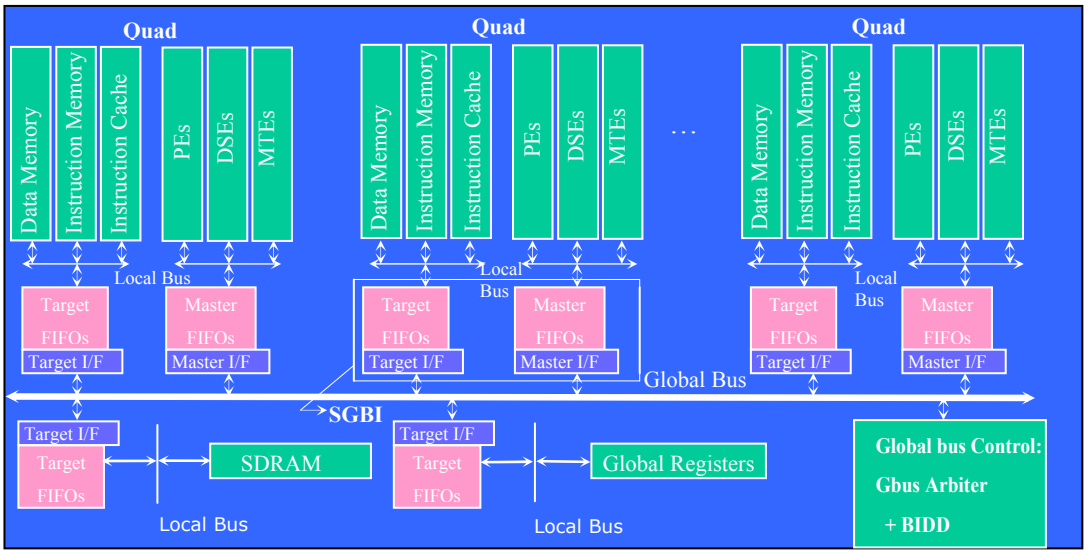

Figure 3: UMS Global Bus Block Diagram

Registers. Control write is also used to send global timing signals and global wake-up interrupts to all Quads. *Control read* is a counterpart to control write. Control read allows the host or configuring device to read base addresses and configuration registers in the GBus control address space as well as write to them.

Each Global Bus Master has only one transaction in process at any one time. It cannot initiate another transaction until its current transaction is complete. But the bus can have many transactions in progress at a time. For example, during the delay between the two steps of the read operation—read initiation and read completion, neither the master nor the target is on the bus. While a master is waiting for completion of its read, the bus can support other transactions-- other masters can perform write transfers and initiate other read transfers.

The recommended Global Bus transfer atom is four words of eight bytes (octets) each, with one and two octet transfers as special cases. A four-octet transfer has a bus efficiency of 80%, at one command octet per four data octet. All transfers are written to a FIFO on the bus. Addresses and data are pipelined.

2.2.2 Transfer Acknowledge, Command Reject, and Back Off Strategy

The target device receives each octet transferred on the GBus and acknowledges the octet by activating the Transfer Acknowledge (TACK) line of the bus. The introduction of TACK line is a good idea for two reasons: first no special transactions for acknowledge will be involved on the bus; second it detects the bus error when no device has responded to a command immediately, without having to wait for a bus time out.

Target devices can reject transfer commands by activating the CRJ (Command Reject) lines. Command reject occurs when the target is busy servicing a prior transfer request while a new request is received. This can happen when two bus devices try to transfer data to or from the same target in quick succession.

Since the bus is much faster than the targets, a second master can request transfer from a target before the target has had a chance to respond to the request from the first master. The command from the second master must be rejected in this case.

Command reject is handled by each Global Bus interface. The response is to retry the command after a waiting period called the back off time. If multiple commands are rejected when the target device is busy, the commands will be retried in the order in which they were rejected. The first command rejected should get the shortest back off code (01). The second command rejected should get the medium back off code (10), and the third and all further commands should get the longest back off code (11). If the GBI receives a command reject, it should wait for 16, 32 or 64 GBus clocks for codes of 01, 10 or 11, respectively, before retrying the command.

2.2.3 GBus Arbiter and BIDD

Each bus, local or global, has an arbitrator for bus transfer. Each bus master element submits a request to the bus arbiter for bus transfer and receives a bus grant from the arbiter. Arbitration is sequential, for example, if a PE in Quad 1 wants to write a word to a memory in Quad 2, it will have to request the local bus of Quad 1 and get a grant; then the Quad 1 master interface will generate a request to GBus and get a grant; when the request arrives to the Quad 2 target interface, it will send a request to the local bus of Quad 2 and get a grant; finally the memory in Quad 2 gets this request and writes the word.

The arbitration priority of the GBus is round-robin. When a Global Bus transaction requests a Quad local bus at the same time as a local transaction requests for that local bus, Global Bus transaction will have the priority.

When the Global Bus is idle, no active device is selected to drive the bus. If no active device is selected, the arbiter selects a default device, the Bus Idle Default Device (BIDD), to drive the bus. Otherwise, the device lines would float, potentially causing noise and errors. The Idle Device drives the bus signals to a safe default state. It drives the bus command lines to the idle state,

the address/data lines are held at their previous values (for low power); the byte enables to inactive; the target device number to all ones and the CRJ lines to inactive.

3. PERFORMANCE ANALYSIS METHODOLOGY

In this section, we describe the proposed simulation strategy for fast performance analysis of the UMS Global Bus. The methodology is shown in Figure 4.

Our performance analysis methodology can be roughly divided into two parts: Traffic Patterns Generation (TPG) and Simulator Generation (SG). For TPG, first the traffic statistics can be gathered from the execution profiles generated by running the applications on Cradle's UMS Inspector. The traffic patterns are then constructed based on the traffic statistics, and different assumptions for data distributions. For SG, an accurate model for the Global Bus is setup. The model is a combination of a series of queuing models of the UMS component; the combination is based on the abstraction of the Global Bus Protocol. UMS hardware parameters are used to give quantitative information for the model. The detailed description of SG and TPG is given in the following two subsections.

3.1 Simulator Generation

In creating the analytical model for the Global Bus, we follow two general rules. It should be: (I) as accurate as possible; (II) as simple as possible. The first rule requires careful analysis of the Global Bus Protocol, while the second rule allows us to abstract out some trivial (for performance evaluation, not for the architecture itself) details. For example, we don't consider control write and control read in our model, because they are unusual bus transactions, especially after the system has been configured. Similarly, BIDD does not show up in our model since it will not produce much traffic on the Global Bus. When these two rules conflict with each other, a trade-off has to be made. The model for Global Bus with 4 Quads is shown in Figure 5. It is quite easy to extend this model to n Quads (n>4).

The model is a Directed Graph (DG), in which the Global Bus and Global Bus devices are the vertices. Every vertex is a single M/M/1 queuing model or a group of them. A path from one vertex to another could be an integrated Global Bus Transaction (GBT), but not all paths are legal GBTs. For example, a path starting at GBus can not be a legal GBT. Thus, the set of GBTs is really a subset of the set of all paths. Also, note that GBT is not vertex disjoint.

We use a simple example to briefly explain the model. Following is the transaction showing a PE in Quad 1 (denotes as Q1) reading data from the SDRAM:

- ▶Q1 LP (Local Process) sends request to Q1 LB (Local Bus) and gets a grant;
- ▶Served by Q1 MW (Q1 master interface write logic set);
- ▶Sends request to the GBus, gets a grant and transferred;
- ▶Sends request to the SDRAM TW (SDRAM target interface write logic set) and gets a grant (if SDRAM TW buffer is not full);
- ▶Sends request to the SDRAM Local Bus, gets a grant and transferred;
- ▶Sends request to the SDRAM.

Till now, the first phase of the transaction has been finished. The second phase of this transaction starts after the SDRAM has prepared the data Q1 required.

- ▶SDRAM sends request to the SDRAM Local Bus and gets a grant;
- ▶Served by the SDRAM TR (SDRAM target interface read logic set);
- ▶Sends request to the GBus, gets a grant and transferred;
- ▶Served by Q1 MR (Quad 1 master interface read logic set);

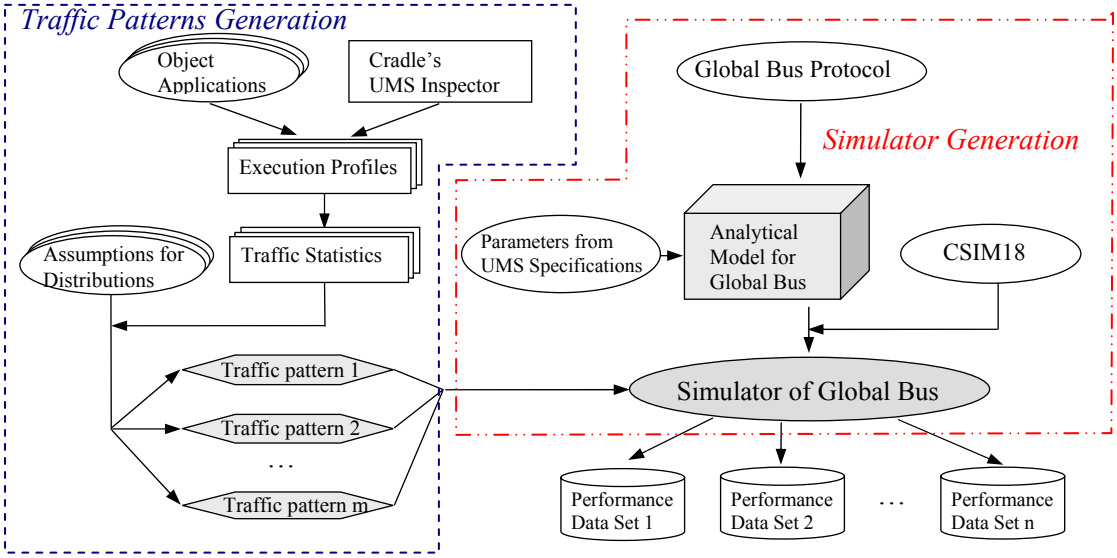

Figure 4: Performance Analysis Methodology

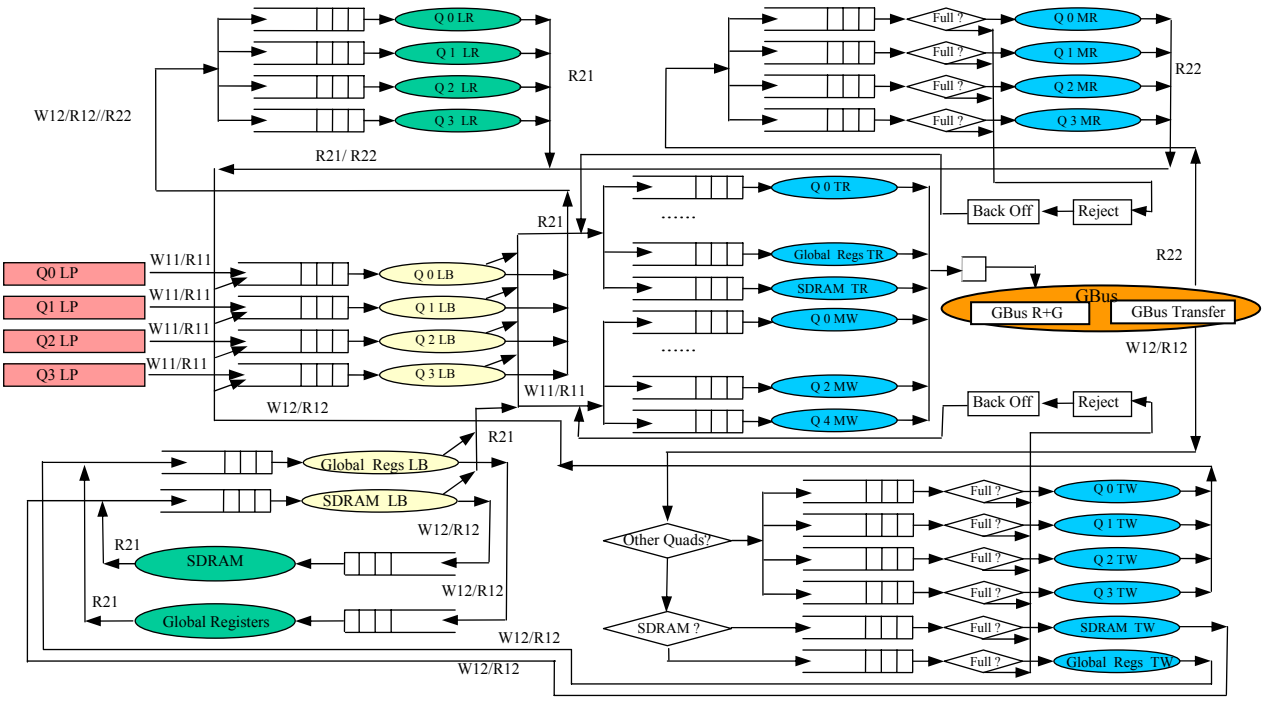

Figure 5: The model for Global Bus with 4 Quads

- ►Sends request to Q1 LB, gets a grant and transferred;
- ►Sends data to Q1 LR (Q1 Local Resource, PE in this example).

Now the transaction is done. During the above transaction, if the buffer of SDRAM TW is full when the request is sent to it, then the command will be rejected and retried. Other transactions are similar.

The parameters of UMS hardware are very important; they provide quantitative information for the model. Parameters like the mean service time of the Quad local bus, and the mean service time of the GBus master/target interface are not difficult to define. However, due to the locality attribute of the traffic, SDRAM and Quad local memory read/write mean service time are difficult to define accurately. In our simulation, we use Average Access Time (AAT) / Average Access Cycles (AAC) to describe the SDRAM and Quad local memory services. The cycles here refer to the Global Bus cycles, thus the combination of the AAT and the Global Bus frequency gives the AAC. For example, if the Global Bus runs at 350MHz and AAT=142.9ns, then the AAC is 50. We use AAC in the model for simulation, and AAT to describe the experiment data in section 4. The access response latency (different from AAT/AAC) depends on the traffic load on the SDRAM and Quad local memory. For example, Table 1 gives the SDRAM read latency (when AAC=50) for different SDRAM utilizations sampled from simulation.

Table 2 gives the description of the parameters we used in the model.

We implement the simulator in C++ using CSIM18, a library of routines for use in constructing process-oriented, event-driven simulation models (Detailed information of CSIM18 can be found at (http://www.mesquite.com).

3.2 Traffic Patterns Generation

We take image processing applications as our object applications, which include image negation, tetrahedron color conversion, image filtering, and JPEG decoding. These applications are implemented on UMS as described in Table 3.

The application are then run on Cradle's UMS inspector, which is a simulator & debugging tool for developing UMS applications. The profiles provided by this tool can give us information like: active cycles, total numbers of local and global operations, and the distributions of the local and global traffic, etc., for PEs, DSEs, and MTEs of every Quad respectively. Traffic patterns are built based on such information. While the implementations on UMS for these applications are not using all the resources of Quads, we scale the traffic patterns to fit 4 Quads, and keep the same computation-communication ratio as they have. For example, the image negation uses 8 MSPs, when creating the traffic pattern for it, we double the traffic abstracted for the profiles, to get the traffic patterns for 4 Quads (16 MSPs). We call the traffic pattern abstracted from the profiles

Table 1: The SDRAM read latency

SDRAM utilization	10%	20%	30%	40%	50%	60%	70%	80%	90%
Read latency (UMS cycles)	56.2	62.6	70.3	84.9	97.8	122.9	169.5	272.3	534.6

Table 2: Parameters for the model of the Global Bus

Services		Queuing Model /Characteristics	Parameters	Values (UMS cycles)
Global Bus	Request and Grant	M/M/1	Mean service time	2.0
	Command transfer	M/M/1	Mean transfer time	Depends on the mean operation size of specific traffic pattern
Global Bus Interface (including MR, MW, TR, TW)		M/M/1	Mean service time	5.0
SDRAM		M/M/1	AAC	50.0
Quad Local Memory		M/M/1	AAC	15.0
Global Registers		M/M/1	Mean service time	10.0
Quad Local Bus		M/M/1	Mean service time	5.0
SDRAM Local Bus		M/M/1	Mean service time	5.0
Global Registers Local Bus		M/M/1	Mean service time	5.0
Back Off after Rejection		Exponential	Back off timer	16.0, 32.0, 64.0 for back off code 01, 10, 11 respectively

Table 3. Implementation of the image processing applications on UMS

	Description	Implementation on UMS
Image Negation	Perform for the data transferring; PEs image negation	8 MSPs involved; DSEs do the image negation row by row; MTEs are responsible do the data partitioning, initialize and control the DSEs and MTEs.
Tetrahedron Color Conversion	Perform tetrahedron color conversion	8 MSPs involved; DSEs do tetrahedron interpolation; MTEs are responsible for the data transferring; PEs do the data partitioning, initialize and control the DSEs and MTEs.
JPEG Decoding	Decode baseline JPEG image	1 MSP involved; DSEs do the Haffman decoding and IDCT, MTEs are responsible for the data transferring; PEs do the data partitioning, initialize and control the DSEs and MTEs.
Image Blurring Image Sharpening Edge Filtering	Blurring / Sharpening / Filtering image edge	8 MSPs involved; DSEs do the image filtering and dithering; MTEs are responsible for the data transferring; PEs do the data partitioning, initialize and control the DSEs and MTEs.

Table 4: The prime traffic pattern for Image Negation

Items	Quantum	Distribution
IRI: the Inter-Request Interval of global operations	49.0 UMS cycles	Poisson
QQ: # of Quad-Quad operations / # of total global operations	35%	Bernoulli
QS: # of Quad-SDRAM operations / # of total global operations	65%	
QQR: # of Quad-Quad read / # of total Quad-Quad operations	15%	Bernoulli
QQW: # of Quad-Quad write / # of total Quad-Quad operations	85%	
QSR: # of Quad-SDRAM read / # of total Quad-SDRAM operations	75%	Bernoulli
QSW: # Quad-SDRAM write / # of total Quad-SDRAM operations	25%	
MOS: Mean of Operation Size.	2.94 octets (1 octet=8 bytes)	Poisson
The possibility for one Quad to be accessed by other Quads	Uniformly	Random

the prime traffic pattern. The prime traffic pattern for Image Negation is shown in Table 4.

Table 5 shows the prime traffic patterns for other image processing applications, note that in this table, only quantum of the patterns are listed, we assume they have the same distribution as the Image Negation; we also assume every Quad has the same possibility to be accessed by other Quads.

The usefulness of a single prime traffic pattern is quite limited. Different traffic patterns could be constructed by changing the specific parameters of the prime traffic pattern. For example, decreasing the inter-arrival cycles of global operations can increase the traffic load; modifying the ratio between the Quad-

Table 5: Prime traffic patterns for other image processing applications

	Tetrahedron Color Conversion	JPEG Decoding	Image Blurring	Image Sharpening	Edge Filtering
IRI	42.0 UMS cycles	147.0 UMS cycles	133.0 UMS cycles	159.0 UMS cycles	201.0 UMS cycles
QQ	40%	40%	35%	35%	35%
QS	60%	60%	65%	65%	65%
QQR	16%	24%	38%	34%	31%
QQW	84%	76%	62%	66%	69%
QSR	70%	65%	60%	60%	60%
QSW	30%	35%	40%	40%	40%
MOS	2.91octets	3.03 octets	3.39 octets	3.31 octets	3.26 octets

to-Quad traffic and the Quad-to-SDRAM traffic can show us how these kinds of traffic influence the performance, etc. We call these constructed traffic patterns derivative traffic patterns.

Our experiments are designed on the above traffic pattern variants to explore the performance, the limits, and the potential problems with the UMS Global Bus.

4. EXPERIMENT DATA AND ANALYSIS

The experiments are designed mainly to address the following questions:

- How do the traffic patterns influence the performance of the Global Bus? What could be the performance space?
- Is the Global Bus the potential bottleneck? If not, which components could most possibly be the bottleneck?
- How do buffers affect performance?

Following subsections give a detailed description on the experiments design and data analysis. All experiments are running on a 4-Quads model, and assume that the Global Bus runs at 350MHz and the Quads run at 200MHz.

4.1 Effects of Traffic Distribution and Inter-Request Intervals

While every parameter of the prime traffic pattern described in section 3 can be modified to construct a new pattern, to keep the important characteristics of the image processing applications, only the Inter-Request Intervals (IRI) and the ratio of Quad-to-Quad communications to Quad-to-SDRAM communications are made as variables to explore the performance space of the Global Bus.

Through simulations we found that for the performance of the Global Bus, the Tetrahedron Color Conversion and Image Negation show similar characteristics, while JPEG Decoding, Image Blurring, Image Sharpening, and Edge Filtering show similar characteristics. We will give detailed description for the experiment results of Tetrahedron Color Conversion and JPEG Decoding. For other applications, we use tables to summarize the results.

Figure 7 shows the effects of IRI and QQ values (see section 3) on the utilization of Global Bus for the Tetrahedron Color Conversion application. In most situations, we find the Global Bus utilization to be less than 30%, which shows that the Global Bus can handle such traffic very well. The utilization of the Global Bus generally increases with decreasing IRI because of the increasing load. For the same inter-request interval, we can see that the utilization is inversely proportional to QQ. This can be explained by Quad-to-Quad communications generally having a smaller request size than Quad-to-SDRAM communications.

One interesting point is that the "bars" for some series in Figure 7 disappear at some inter-request intervals. This is explained by Figure 8, which shows the SDRAM utilization among different combinations of IRI and QQ values. From Figure 8, we see that the SDRAM is totally saturated in several series when IRI drops down below a certain value. For example, the SDRAM utilization of the series for QQ=10% is 100% when IRI is less than 42 cycles. During this situation, the system becomes unstable and finally halts the simulation. Image Negation has very similar result; the SDRAM Utilization goes to 100% as early as Global Bus has less than 30% Utilization.

The Global Bus and SDRAM Utilization for the JPEG Decoding are shown in Figure 9 and Figure 10, respectively. Compared to Tetrahedron Color Conversion and Image Negation, we found that JPEG Decoding has much less utilization of Global Bus and

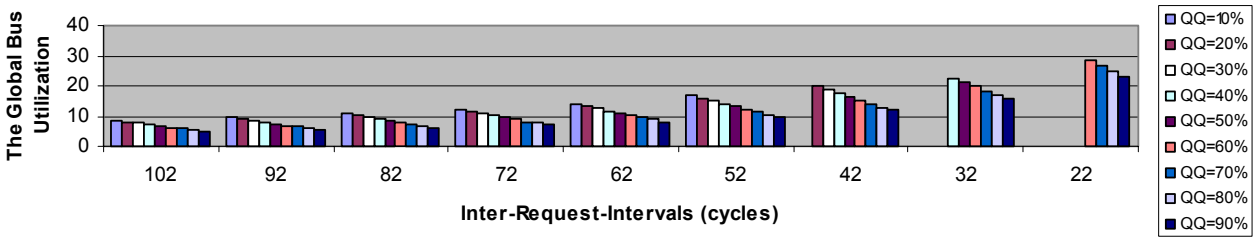

Figure 7: Effects on IRI and QQ on the performance of the Global Bus for the Tetrahedron Color Conversion

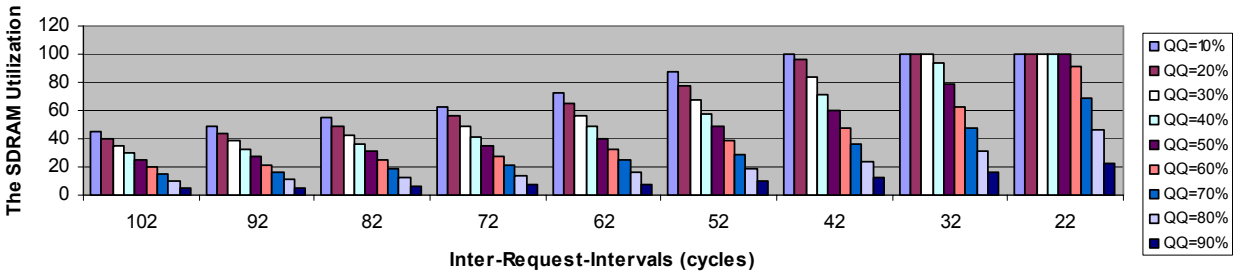

Figure 8: The SDRAM Utilization for Figure 7

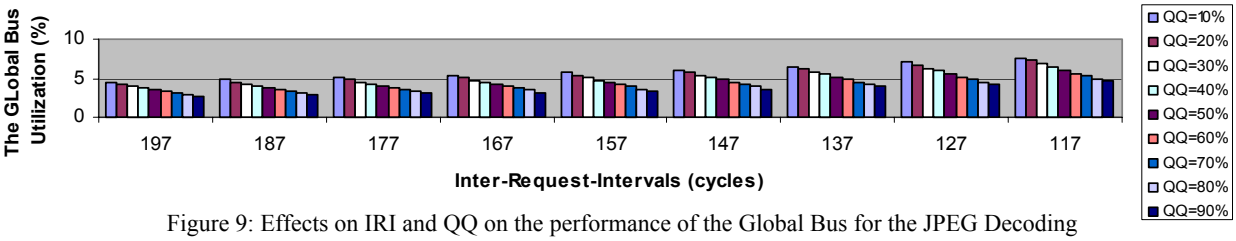

Figure 9: Effects on IRI and QQ on the performance of the Global Bus for the JPEG Decoding

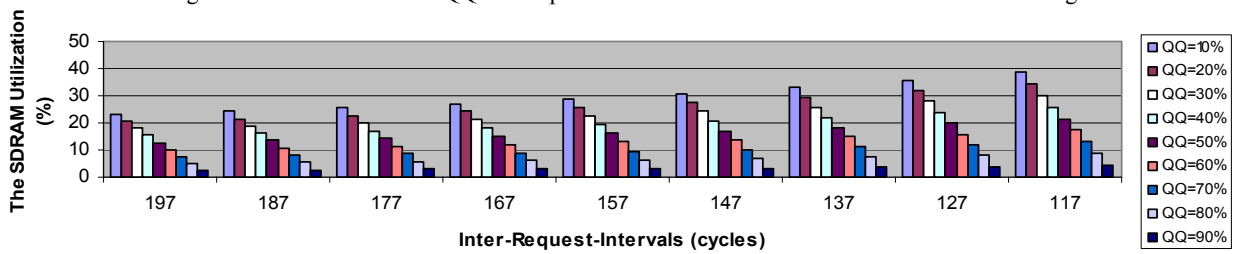

Figure 10: The SDRAM Utilization for Figure 9

Table 6: Summary of the Global Bus and SDRAM Utilization for the image processing applications

	Image Negation	Tetrahedron Color Conversion	JPEG Decoding	Image Blurring	Image Sharpening	Edge Filtering
The Global Bus Utilization	4.41%--24.53%	4.95%--28.78%	2.72%--7.66%	2.97%--8.10%	2.79%--7.53%	2.72%--7.66%
The SDRAM Utilization	2.25%--100%	4.93%--100%	2.54%--38.63%	1.30%--37.88%	1.27%--35.87%	1.04%--26.50%

SDRAM because JPEG Decoding involves more computation, which increases the IRIs. Table 6 summarizes the results for all the image processing applications.

Figure 11 shows that the SDRAM limits the performance space of the Global Bus. In Figure 11, we compare the utilization of SDRAM and the Global Bus for Tetrahedron Color Conversion and JPEG Decoding, respectively. In the left part of this graph, which is for the Tetrahedron Color Conversion, we see that SDRAM gets saturated as early as 30% Global Bus utilization. Although this situation is not seen in the right part of the graph for the JPEG Decoding, the SDRAM Utilization grows much faster than the Global Bus Utilization; it is expected that SDRAM will block the increase the Global Bus Utilization when we have smaller IRIs.

Figure 12 goes further to show how much the performance of the Global Bus can be improved if the SDRAM AAT decreases from 142.9ns cycles to 71.5ns cycles. We see the maximum throughput of Global Bus with SDRAM AAC=71.5ns almost double its counterpart with SDRAM AAC=142.9ns, until it nearly reaches the theoretical limit (2.8GBytes/s for 350MHz Global Bus).

4.2 Effects of the Buffer Size

Every Global Bus master or target device has two FIFOs to buffer read or write commands. With deeper buffers, fewer commands will be rejected, thus the retries are reduced and the effective bandwidth of the Global Bus increases. But buffers are expensive in space utilization. So to find the relationship between the depth of the buffer and the optimal value of the depth has practical significance.

Figure 13 compares the commands rejection rate for Quad's interface at different buffer sizes, when QQ has value of 90% (The cases with QQ value less than 90% are not shown in graph, because they have less commands rejection rate than this case). This graph tells us that buffers of Quad's interface with size larger than 8 commands provide no benefits for general cases (QQ equal or less than 90%). Similarly, Figure 14 shows that

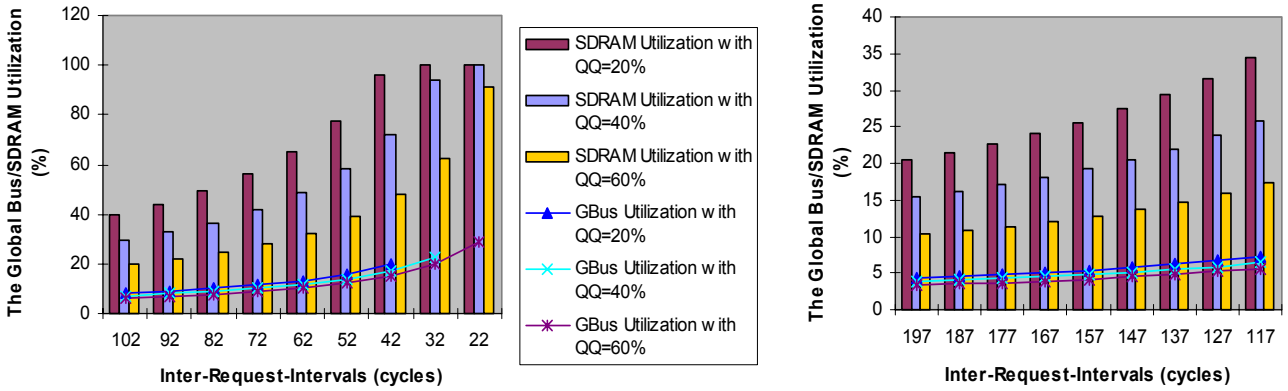

Figure 11: Comparison of the utilization of the Global Bus and the SDRAM for Tetrahedron Color Conversion (left) and JPEG Decoding (right)

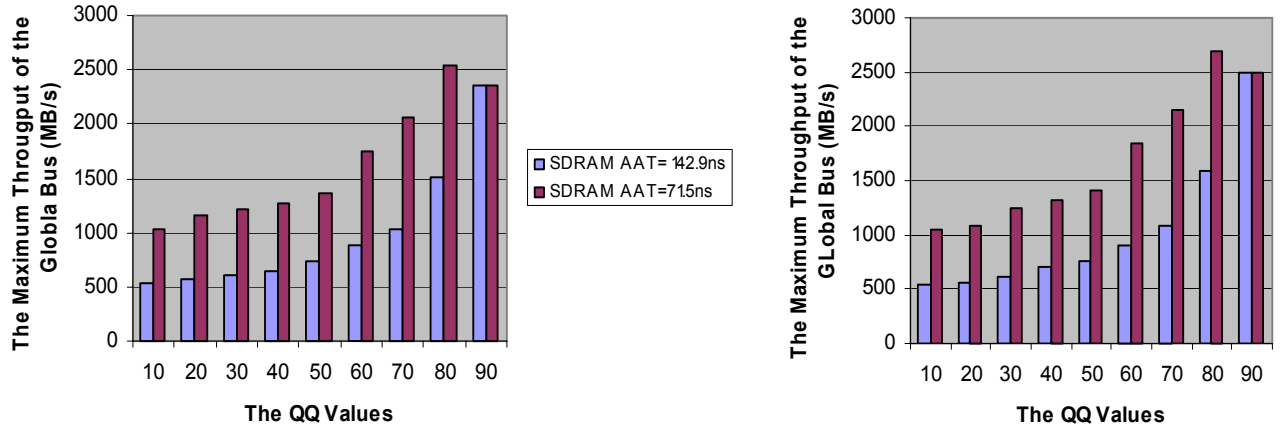

Figure 12: The performance enhancement with lower SDRAM AAT for Tetrahedron Color Conversion (left) and JPEG Decoding (right)

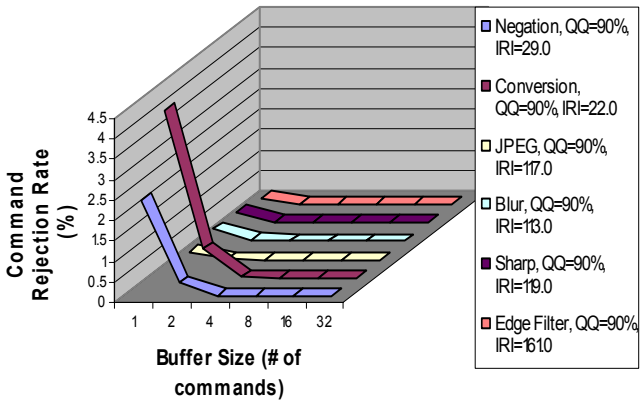

Figure 13: Commands rejection rate for Quad's interface at different buffer Size

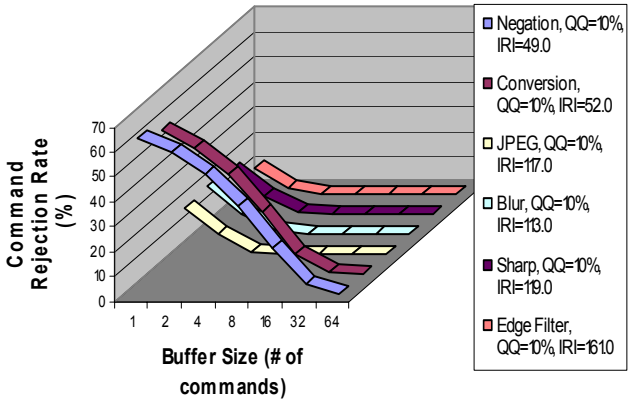

Figure 14: Commands rejection rate for the SDRAM's interface at different buffer Size

the optimal buffer size for SDRAM interface is between 32 and 64 commands.

5. RELATED WORK

Lahiri, et al., compare the performance of several types of popular SOC communication architectures, including priority based shared bus, hierarchical bus, two-level TDMA architecture, ring based architecture, and multi-channel architectures in [5][6][9]. The experiments in these papers show

that while all of these architectures have their pros and cons, none of them uniformly outperforms others--their performance highly depends on the traffic characteristics – thus the specific application domains. Carim, et al., propose a novel communication architecture for a 8-cpu network processor—Octagon architecture in [8]; the authors compare the performance of shared bus, on-chip crossbar and Octagon, arguing that Octagon yields higher performance while keeping low implementation cost making it especially suitable for high speed network processors. Varatkar, et al. investigate the traffic analysis for on-chip network design of multimedia applications, introducing self-similarity as a fundamental property exhibited by the bursty traffic between on-chip modules in typical MPEG-2 video applications [7].

In terms of simulation methodologies, trace-driven simulation is widely studied and used in evaluation of computer systems. But **strict** trace-driven simulation may not be the best choice for evaluating SOC communication architectures, especially when we want to give a fast evaluation. The reason is a number of factors make trace-driven simulation difficult in practice. Collecting a complete and detailed address trace may be hard, especially if it is to represent a complex workload consisting of multiple processes. Another practical problem is that address traces are typically very large, potentially consuming gigabytes of storage space. Finally, processing a trace to simulate the target system is a time-consuming task [10].

Lahiri, et al., give a fast performance analysis of bus-based SOC communication architecture in [9]. The main strategy of performance evaluation can be spilt into three steps: (i) initial co-simulation performed after HW/SW partitioning and mapping, with the communication between components modeled in an abstract manner (*e.g.,* as events or data transfers), (ii) extraction of abstracted symbolic traces, represented as a Bus and Synchronization Event (BSE) Graph, (iii) manipulation of the BSE Graph using the bus parameters, to derive the behavior of the system accounting for effects of the bus architecture.

Recently for improving the performance of memory system, optimizing the access strategy has been proposed as a promising approach, beside enhancing the speed of SDRAM itself by introducing new memory architecture (like SLDRAM or Direct Rambus, etc), Rixner, et al. introduces memory access scheduling, a technique of reordering memory references to exploit locality within the 3-D memory architecture (banks, rows, and columns) in [11]. The experiment in this paper shows 93% bandwidth improvement by using aggressive reordering for media processing applications. Being aware that the access conflict with the increase of embedded-DRAM masters (CPUs, DSPs, etc) will significantly degrade the SOC performance, Wantanbe, et al. propose an "access optimizer", a logic attachment for embedded DRAMs which consists of three control units for self-prefetching, address alignment, and inter-bank non-blocking access in [12]. The access optimizer can successfully suppress the degradation of the CPU performance introduced by the access conflicts.

6. CONCLUSION

This paper focused on the performance evaluation of the share-based Global Bus architecture of the UMS. By the proposed fast performance analysis methodology, we studied the performance space of the Global Bus, especially for image processing applications. We concluded that carefully designed share based bus architecture like Global Bus can be efficient enough as the communication backbone of current and future SOCs. While the Global Bus itself is very unlikely to be the system bottleneck, the SDRAM subsystem may heavily influence the overall system performance. This paper also analyzed the impact of buffer size on the bus performance; references for the optimal buffer depth are given by simulation.

REFERENCES

[1] Rick Merritt, Intel, Sun sketch multiprocessor chip plans, *http://www.eetimes.com/story/OEG20011210S0069*.

[2] Jaehyuk Huh, et al., Maximizing Area Efficiency for Single-Chip Server Processors, *2nd Annual Austin CAS Conference*, Feb. 2001

[3] White paper --The Universal Micro System, Cradle Technologies, Inc., Sep. 2001

[4] UMS Documentation -- UMS2003 Hardware Architecture, Cradle Technologies, Inc., Mar. 2002

[5] Kanishka Lahiri, et al., Performance Analysis of Systems with Multi-Channel Communication Architectures, *Proc. Intl. Conf. on VLSI Design, pp.530-537, Calcutta, India,* Jan. 2001.

[6] Kanishka Lahiri, et al., Evaluation of the Traffic-Performance Characteristics of System-on-Chip Communication Architectures, *Proc. Intl. Conf. on VLSI Design, pp.21-35, Bangalore, India,* Jan. 2001.

[7] Cirish Varatkar, Radu Marculescu, Traffic analysis for on-chip networks design of multimedia applications, *Proceedings of the 39th conference on Design automation*, June 2002

[8] Faraydon Carim, et al., On-chip communication architecture for OC-768 network processors, *Proceedings of the 38th conference on Design automation*, June 2001.

[9] Kanishka Lahiri, et al., Fast performance analysis of bus-based system-on-chip communication architectures, *Proceeding of the 1999 international conference on Computer-aided design*, Nov. 1999.

[10] Richard A. Uhlig, Trevor N. Mudge, Trace-driven memory simulation: A survey, *ACM Computing Surveys, vol. 29, no. 2*, June 1997.

[11] Scott Rixner, William J. Dally, et al., Memory Access Scheduling, *Proceedings of the 27th International Symposium on Computer Architecture,* on June 2000.

[12] Takao Wantanbe, et al., Access Optimizer to Overcome the future Walls of Embedded DRAMs in the Era of Systems on Silicon, *1999 IEEE International Solid-State Circuits Conference,* Feb. 1999.