A Real-Time Streaming Memory Controller

Artur Burchard, Ewa Hekstra-Nowacka, Atul Chauhan

A Real-Time Streaming Memory Controller

Artur Burchard[1], Ewa Hekstra-Nowacka[1] and Atul Chauhan[2]
(1) Philips Research Laboratories, Eindhoven, The Netherlands
(2) Indian Institute of Technology, Delhi, India
artur.burchard@philips.com

Abstract

With ever more complex multimedia applications used in mobile devices, the realization of high performance, flexibility and programmability requirements depends largely on the design of a system communication infrastructure. This infrastructure, often a network, should provide a large variety of services for the transportation of streamed data. When an external memory is also used for streaming communication purposes and, together with the communication infrastructure, forms a part of the streaming, additional support is needed for the memory in order to guarantee the integrity of the communication services provided by the network when data is accessing the memory. This led us to a design of a streaming memory controller (SMC) for off-chip (DDR-)SDRAM memories that enables a shared memory implementation of the streaming based on an off-chip network (PCI Express). In this paper, we present the ideas that gave rise to the SMC, the actual design of the SMC, as well as the evaluation of the design.

1. Introduction

The complexity of advanced mobile devices is increasing. The ever more demanding applications of such devices and the complexity, flexibility and programmability requirements have led to intensified data exchange inside the devices. The devices implementing such applications consist of several functions or processing blocks, here referred to as 'subsystems'. These subsystems are typically implemented as separate ICs, each having a different internal architecture that consists of local processors, busses and memories, etc. At system level, these subsystems communicate with each other via a top-level interconnect that provides certain, often real-time, services. As an example of subsystems in a mobile phone architecture, we can refer to, among others, a base-band processor, a display, a media processor or a storage element. An example of the system level interconnects is a PCI Express network that provides services like isochronous data transport and flow control. Furthermore, an example of data streaming could be when a media processor reads out an MP3 encoded audio file stored locally and the decoded stream is sent to the speakers.

When used to support multimedia applications, these subsystems exchange most of the data in a streamed manner. Such communication can be modelled as a streaming application and described as a graph of processes connected via FIFO buffers, often referred to as Kahn process networks [1]. The Kahn process network can be mapped on the system architecture [2], i.e. the processes are mapped onto the subsystems, the FIFO buffers are mapped onto the memories and the communications are mapped onto the system-level interconnect, as shown in Figure 1.

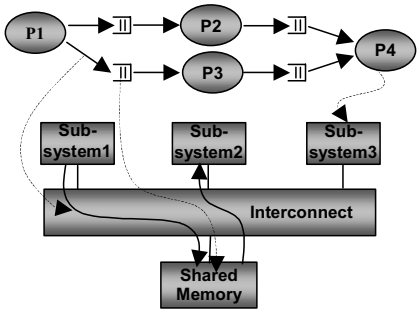

Figure 1. Kahn process network mapped onto a shared memory interconnect-centric architecture

Buffering is essential for the proper support of data streaming between involved processes. It is natural to use FIFO buffers for streaming, and this is in accordance with the (bounded) Kahn process network model of streaming application. As the number of multimedia applications that can run simultaneously increases, there is also a substantial increase in the number of processes, real-time streams and the associated FIFOs.

There exist two extreme implementations of streaming with respect to memory usage and FIFOs allocation. The first uses physically distributed memory, where FIFO buffers are allocated in a local memory of a subsystem. The second uses physically and logically unified memory, where all FIFO buffers are allocated in a shared, often off-chip memory, as in Figure 1. In many cases, the

Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE'05)
1530-1591/05 $ 20.00 IEEE

implementation combines elements from both of the styles mentioned.

The FIFO buffers can be implemented in a shared memory using an external DRAM memory technology. SDRAM and DDR-SDRAM are the technologies that deliver large capacity external memory at low cost, with a very attractive cost-to-silicon area ratio.

Until now, controllers of external DRAM were designed to work in bus-based architectures. Busses provide limited services for data transport, simple medium access control and best effort data transport only. In such architectures, the unit that gains access to the bus automatically gains access to the shared memory. Moreover, the memory controllers used in such systems are no more than access blocks optimised to perform numerous low latency reads or writes, often tweaked for processor random cache-like burst accesses. As a side effect of the low-latency, high-bandwidth and high-speed optimisations of the controllers, the power dissipation of external DRAM is relatively high.

In our novel design of a memory subsystem, we address both: a vast variety of communications services (like bandwidth reservation, synchronisation) and real-time streaming accesses to the memory. Because of this approach, we are able to optimise power consumption and enable it to be traded for speed and access latency.

Since we are considering a network that provides a variety of transport services (e.g. bandwidth reservation, guaranteed delivery or flow control of data) as the system interconnect, it is necessary to ensure that the access to the shared external DRAM is transparent for the network with respect to these services. As a result, the memory subsystem provides the same services as the network (e.g. bandwidth guarantees). Therefore, to support data streaming and to comply with the network services, a specific DRAM memory, which does not support the services on its own, has to be equipped with a streaming memory controller (SMC) that ensures that the shared memory subsystem implements the necessary network services. Additionally, because we are considering mobile architectures, SMC needs to minimize the power consumption for memory access.

It should be mentioned that the SMC decouples the shared memory from the interconnect network, i.e. the memory is not aware of the interconnect, and in turn the interconnect network is not aware of the memory specifics, which are hidden by the memory controller.

In this paper we discuss a design and implementation of the SMC that implements network services and that uses (DDR-)SDRAM for implementation of the FIFO buffering. With this design, we enable experimentation with implementation options of the above concepts and gain greater insight into the associated costs in terms of power consumption and silicon area. We also intend to evaluate the arbitration algorithms and communication protocols, proposed by ourselves, for the streaming memory controller with respect to real-time performance as well as complexity and cost. We do not intend to provide optimum solutions, but instead we use the proposed heuristics to learn more about which aspects of memory access are essential for improving performance and cost figures.

The long-term goal is to design an SMC for a mobile device architecture that uses a network supporting real-time data communication. Such networks are currently addressed in MIPI standardization [3]. As an intermediate step we used *PCI Express* [5] as an interconnect technology and we have designed an SMC that integrates external DRAM into a streaming supported by a PCI Express network.

PCI Express is a well-founded advanced general-purpose I/O technology that targets the PC world. It provides a variety of services as well as mechanisms for isochronous data transfers. We chose it for experimentation purposes, and we do not claim that this choice of interconnect is ideal for mobile and portable devices. Nevertheless, this is a first step towards a mobile solution.

The paper is organized as follows. In Section 2 we explain the design problem in more detail. In Sections 3 and 4 we discuss the requirements for the streaming memory controller that are imposed by the PCI Express and (DDR-)SDRAM interfaces. In Section 5 we elaborate on implementation issues of the system design. In Section 6 we present selected aspects of performance, cost and trade-off analysis. The conclusions are given in Section 7 to complete the paper.

2. Problem statement

PCI Express provides network services, e.g. guaranteed real-time data transport, through exclusive resource/bandwidth reservation in the devices that are traversed by the real-time streams. When an external DRAM supported by a standard controller is connected to the PCI Express fabric, without having any intelligent memory controller in between, bandwidth and delay guarantees – typically provided by the PCI Express – will not be fulfilled by the memory since it does not give any guarantees and acts as a ‘slave’ to incoming traffic.

The design of a standard memory controller focuses on delivering the highest possible bandwidth at the lowest possible latency. Such an approach is suitable for processor data and instruction (cache) access and not for isochronous traffic. To be able to provide the predictable behaviour of the PCI Express network extended with the external DRAM, a streaming memory controller is needed which guarantees a predictable behaviour of the external memory for streaming. In addition, we aim to design the memory controller not only for guaranteeing

throughput and latency, but also for reducing power consumption while accessing this DRAM.

The SMC depicted in Figure 2 has two interfaces: one towards the PCI Express fabric, and one towards the DRAM. The PCI Express interface of the SMC must perform the traffic shaping on the data retrieved from the SDRAM to comply with the traffic rules of the PCI Express. On the other interface of the SMC, the access to the DRAM has a burst-like character, since this mode of accessing data stored in DRAM offers the greatest advantage with respect to power consumption. The SMC itself must provide intelligent arbitration of access to the DRAM among different streams such that throughput and latency of access are guaranteed. In addition, the SMC also provides functionality for smart FIFO buffer management.

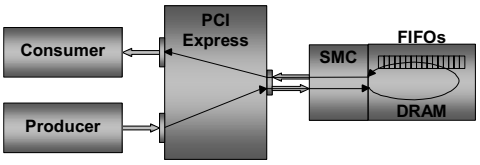

Figure 2. Concept of accessing external DRAM from the PCI Express fabric through a Streaming Memory Controller

3. PCI Express interface

The features of PCI Express [4] [5] that we have taken into consideration in our SMC design are: isochronous data transport support, flow control and specific addressing scheme.

The isochronous support is based primarily on segregation of isochronous and non-isochronous traffic by means of Virtual Channels (VCs). Consequently, network resources like bandwidth and buffers are reserved explicitly in the switch fabric for specific streams. This guarantees no interference between streams in different VCs. Additionally, the isochronous traffic in the switch fabric is regulated by admission control and service discipline, i.e. scheduling.

The flow control that is credit based guarantees that no data is lost in the network due to buffer overflows. The network node is allowed to transmit the network packet through a network link to the other network node only when the receiving node has enough space to receive the data. Every VC has a dedicated flow control infrastructure. A synchronisation between the source and destination can therefore be realised separately for every VC through chained PCI Express flow control.

The PCI Express addressing scheme uses 32 or 64 bit memory addresses. In our design, we do not want to use explicit memory addresses. Instead, we aim to use device and function IDs, i.e. stream IDs, to differentiate between streams. The SMC itself will generate/convert stream IDs into actual memory addresses. In our prototype design we decided to simplify the addressing scheme even further and to use VC ID as a stream identifier. Since PCI Express allows up to eight VCs, we therefore decided to use half of them for identifying incoming streams and the other half for identifying outgoing streams from the external memory. The maximum number of streams that can access the memory through our SMC is therefore limited to eight. Note that such a limitation is caused by PCI Express because it allows for arbitration between streams in different VCs but not between those inside the same VC. However, such a limitation is only specific to PCI Express based systems, it is not fundamental to the concepts presented.

In summary, the PCI Express interface of SMC consists of a full PCI Express interface, equipped additionally with some logic that is required for address translation and stream identification.

4. (DDR-) SDRAM interface

For our SMC design we used the Micron's 128-Mbit DDR-SDRAM [6]. We decided to use this technology because it provides desirable power consumption and timing behaviour. However, our design is parameterised and SMC can be configured to work with single rate memory as well. Since the DDR-SDRAM behaves in a similar way to SDRAM, except for the timing of the data lines, we explain the basics using SDRAM concepts.

The basic operations of the SDRAM, the logical architecture of which is depicted in Figure 3, are for the state when the memory clock is enabled, i.e. the memory is in one of the power-up modes. When the clock is disabled, the memory is in a low-power state (stand-by mode).

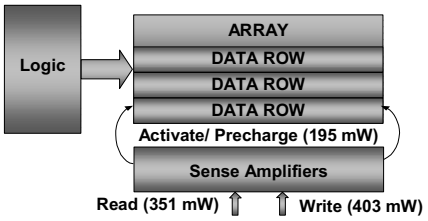

Figure 3. Logical SDRAM model

Typical commands applied to a memory are *activate*, *pre-charge*, *read*/*write* and *refresh*. The *activate* command ensures that a bank and row address are selected and that the data row (often referred to as a page) is transferred to the sense amplifiers. The data remains in the sense amplifiers until the *pre-charge* command restores the data to the appropriate cells in the array. When data is available in the sense amplifiers, the memory is said to be in the active state. During such a state reads and writes can take place. After the *pre-charge* command, the memory is said to be in the pre-charge state where all data is stored in a cell array.

Another interesting aspect of the memory operation is a refresh. The memory cells of the SDRAM store data using small capacitors and these must be recharged regularly to guarantee the integrity of data. When powered up, the SDRAM memory is instructed by the controller to perform a refresh. When powered down, SDRAM is in self-refresh mode (i.e. no clock is enabled) and the memory performs a refresh on its own. This state consumes very little power. It takes about 200 clock cycles to get the memory out of the self-refresh mode into a state in which data can be asserted for read or write, specifically in the case of DDR-SDRAM.

The timing and power management of the memory is essential for the proper design of the SMC, which must provide specific bandwidth, latency and power guarantees. Our analysis and calculations show that to read a full page (equal to 1Kbyte) from an activated SDRAM it takes about 2560 clock cycles (~19.2 us) for the burst length of 1 read, 768 clock cycles (~5.8 us) for the burst length of 8 reads, and only 516 clock cycles (~3.9 us) for the full-page burst. These figures are based on the specific 128-Mbit DDR-SDRAM with a clock period of 7.5 ns [6].

From these calculations it is clear that it is fastest and most efficient in terms of power consumption to access SDRAM in page bursts. Hence, we designed our SMC to use page burst access. However, other burst lengths can be configured. The results of power dissipation for different bursts for the SMC design can be seen in Figure 6a.

5. System design

The proposed SMC architecture, depicted in Figure 4, consists of the following blocks: Stream Access, FIFO Manager, Arbiter and SRAM Memory.

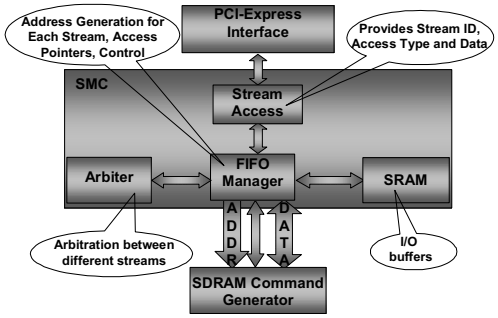

Figure 4. Proposed SMC architecture

The Stream Access provides a stream ID, an access type and the actual data for each stream. For each packet received from the PCI Express interface, based on its VC number, the Stream Access forwards the data to an appropriate input buffer, implemented in local shared SRAM memory. For data retrieved from the (DDR-) SDRAM's FIFOs and placed in the output buffer in the local SRAM, it generates a destination address and passes the data to the PCI Express interface. The Arbiter decides which stream can access the (DDR-)SDRAM. The SRAM memory implements the input/output buffering, i.e. for pre-fetching and write-back purposes. The FIFO manager, which is at the heart of SMC, implements FIFO functionality for the memory through address generation for streams, access pointers update and additional controls. In Figure 5 the logical view of the proposed SMC architecture is shown.

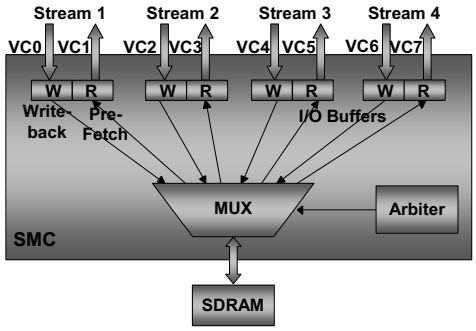

Figure 5. The logical view of the SMC architecture

The arbitration of the memory access between different real-time streams is essential for guaranteeing throughput and bounded access delay. Assume that whenever data is written to or read from the memory, a full page is either written or read. We call the time needed to access one page (slightly different for read and write operations) a 'time slot'. Let us also define a service cycle that consists of a fixed number of time slots. The access sequence repeats and resets every time a new service cycle is started.

The arbitration algorithm between the streams we have proposed is credit based. For each stream a number of credits (time slots) is reserved, the same for every service cycle. The number of credits reflects the bandwidth requirements of the stream. Each time an access is granted to the stream the number of credits available for this stream decreases. The credit count per stream is updated every time the arbitration occurs. Furthermore, credits are reset at the end of the service cycle to guarantee periodicity of the arbitration process. The credit counts can also be refreshed only (e.g. all decreased by the lowest value of all counts) to provide arbitration history of previous service cycles, if adaptive arbitration over a longer time is needed. In extreme case, a single infinitely long service cycle can be used.

When multiple streams want to access the memory in the same time slot, the credit count is used as an arbitration criterion. The stream that has used the least of its credits (relatively, measured as a ratio between the used and reserved credits per current service cycle) gets the access. The denied request is buffered and scheduled (or arbitrated with another incoming request) for the next time slot. If the credit ratios happen to be the same for two requesting streams, the one that requires lower access

latency gets the access first (e.g. read over write).

In this way, every stream (if requesting) will – in the worst case – get the reserved number of accesses to the memory per service cycle, regardless of the order of the incoming requests or the behaviour of the other streams. This guarantees that the bandwidth requirement for every stream is met.

Let us show the example of our credit-based arbitration algorithm. Let us define a time slot equal to a page access (1KB) to SDRAM that, as calculated before, is equal to 3.9 us. Moreover, let us assume the service cycle has 60 time slots, so it is equal to 234 us. Hence, there will be 4273 service cycles per second, what results in the total memory bandwidth of about 2 Gbit/s (4237*60*1KB). Let us also assume there are 3 streams each with 350, 700 and 1050 Mbit/s, respectively, of bandwidth requirements. Therefore, the reserved credit count per service cycle of the first stream will be 350/2100 times 60 slots, which is equal to 10 slots. Streams 2 and 3 will have 20 and 30 reserved credits, respectively. Table 1 shows the stream schedule (row *Sdl*) that is a result of the arbitration. It also shows credit (bandwidth) utilisation levels that determine the arbitration result (rows C_{S1}, C_{S2}, C_{S3} - measured as a ratio between the used and reserved credits per current service cycle) for each time slot (row *Slot*).

Slot	1	2	3	4	5	6	7	8	9	10	11
C_{S1}	0.1	0.1	0.1	0.1	0.1	0.1	0.2	0.2	0.2	0.2	0.2
C_{S2}	0	0.05	0.05	0.05	0.1	0.1	0.1	0.15	0.1	0.1	0.2
C_{S3}	0	0	0.03	0.06	0.06	0.1	0.1	0.1	0.13	0.16	0.16
Sdl	**S_1**	**S_2**	**S_3**	**S_3**	**S_2**	**S_3**	**S_1**	**S_2**	**S_3**	**S_3**	**S_2**

Table 1. Example of the Credit-Based Arbitration

Whereas the reserved bandwidth is always guaranteed for each stream, the reserved but unused slots can be reused by other streams if necessary. This also enables flexible allocation of the bandwidth. While maintaining all guarantees, it enables flexible handling of the unavoidable fluctuations in the network.

Another requirement that must be met to ensure that the above scheme works is that there must be sufficient buffering of the incoming requests, or a mechanism for stalling the requesting streams if other streams are granted the access. The stalling mechanism is implemented using PCI Express flow control, which enables the delay of any stream, separately for each VC. The minimum buffering required can therefore be equal to the size of the data accessed from the memory during one time slot, i.e. page. It is therefore not necessary to increase the access buffering. However, access latency will be reduced because such buffers then behave as pre-fetch or write-back buffers.

The proposed arbitration algorithm is fully parameterised. Most of the aspects of the arbitration can be programmed. For example, the particular arbitration strategy can be chosen at the configuration time, the granularity of memory access (a time slot) can be changed from a page to a burst of another length and, finally, the number of time slots per service cycle can be configured as well.

We have used two types of heuristics for arbitration: time-based and event-based. In the time-based arbitration, every service cycle consists of a fixed number of time slots that are aligned (in time) to each other. Thus, all time slots start at the predefined time and therefore granted access starts at the predetermined moments of time, namely at the beginning of each slot, regardless of when the actual request was issued. In contrast, in the event-based arbitration the time slot starts only when a stream has issued a request and the granted access is served immediately. The service cycle however, for this arbitration takes the same time and is equivalent to a time window that corresponds to the time the fixed number of time slots of time-based arbitration takes.

The differences between the arbitrations mentioned are: the event-based arbitration is more relaxed with respect to power and provides better response latency for requests, whilst the time-based arbitration has simpler control, implementation and lower jitter. Nevertheless, both policies exhibit exactly the same behaviour when the number of requests is equal to or exceeds the total number of available time slots per service cycle.

6. Design evaluation

The SMC proposed in the previous section has been designed in VHDL and synthesized successfully. For the SMC's logic we have used internal Philips CMOS12 (0.12 μm) technology library PcCMOS12corelib (standard V_t). For SRAM we have used internal Philips high-speed high-density single port SRAM technology library C12xSRAM (standard V_t). For simulation and verification, we have assumed 128 Mbits Micron's DDR-SDRAM memory [6].

The DDR-SDRAM memory used in the design operates at a clock frequency of 133MHz. As it accesses the data twice every clock cycle, the SRAM operates at double frequency (266MHz) to be synchronised with the DDR-SDRAM and to provide the same bandwidth. All internal blocks of SMC (FIFO manager, arbiter and SRAM) work at 266 MHz, and all these blocks use the same clock to be synchronised with each other.

In our design we have used two SRAM cells, each with a 16-bit wide data bus and an area of 0.103 mm^2. Each cell has 16 Kbytes. Hence, the total size of buffer space becomes 32 Kbytes (32 pages). The buffer space can be divided between streams based on latency requirements and on the actual data rate of each stream. In our experiments we assumed four pages per stream, although for small and medium data rates this is far too much. The total silicon area is 0.208 mm^2, of which 284

μm^2 is for the arbiter, 1055 μm^2 for the FIFO manager and 0.206 mm^2 for the SRAM. With regard to the power consumption of the SMC, the SRAM consumes 8 mW operating at 266 MHz. The power dissipation of the logic can be ignored. As can be seen from the above figures, the SRAM dominates the silicon and power consumption of the SMC design. The power consumption of the DDR-SDRAM [6] controlled by the SMC in a given playback application (two uncompressed audio streams synchronised in the memory) is shown in figure 6a.

For design verification, a test bench provides the stimulus to the design using test vectors. The test bench pumps data into the SMC from the test vector file and monitors and checks the output ports of the SMC and the internal registers of the SMC to verify the functionality and timing of the design.

By playing with the design and changing its parameters (e.g. buffer and burst sizes, arbitration strategies), it is possible to experiment and obtain results for trade-offs in the design of a real-time streaming memory controller for off-chip memories. Examples of such trade-offs, which can be visualised by exercising the design, are relations between burst sizes and input/output buffer sizes versus a worst-case delay for data access, external memory power dissipation and latency within SMC.

As an example, in Figure 6 we present the power dissipation of an external DDR-SDRAM [6] versus the burst size of the access for a 10 Mbit/s data read from this memory, and worst-case delay versus buffer size in network packets.

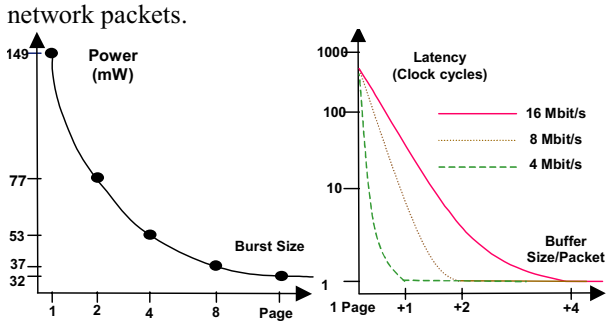

Figure 6. The trade-offs: a. Power versus Buffer Size b. Worst-case delay versus Buffer size

7. Conclusion

We described our design of the novel real-time streaming memory controller that supports off-chip network services and real-time guarantees for accessing external DRAM in a streaming manner. The SMC architecture, the logic view and the design was proposed and discussed. Moreover, we addressed some elements of the heuristics that we have used for implementing real-time stream arbitration.

The SMC has been designed to allow external DRAM to be accessed from within a PCI Express network. This SMC has been designed in VHDL, synthesized and verified. The complexity figures in terms of consumed silicon and power are available. In addition, a design space can be explored for a particular application and certain trade-offs can be visualised by exercising the design with different parameters and arbitration policies. This all enables us to analyse the concept of a streaming memory controller and to understand the problems and issues in its design. We will use this knowledge in the design of a specific SMC for mobile interconnect.

We have not fully optimised the design but the lessons we learned can help in further optimisation of the design presented. Let us mention a few of those important lessons. We have realized an SMC that gives bandwidth guarantees for SDRAM access in a low-power way. The arbitration algorithms, though they always guarantee bandwidth, are still flexible enough to cope with network fluctuations and jitter. PCI Express has a limitation of 8 VC, therefore up to 8 streams that can be arbitrated independently. There are certain important trade-offs for SMC design, such as buffer size (cost) versus power and access delay. The increase of the I/O buffers relaxes the arbitration, lowers the access latency and reduces the cumulated bandwidth required from the SDRAM.

On top of this, we still need to validate the proposed event-based arbitration scheme. In addition, there is a need for an easy way to determine the optimum parameters for the arbitration, including buffer sizes, so that parameters can be chosen that fulfil the guarantees for a certain application (a certain stream set). Actual performance for particular applications will be verified in the future by simulations. It can be realised by hooking up the SMC to models of the PCI Express network and actual DRAM and measuring the execution of actual application.

8. References

[1] G. Kahn. The Semantics of a Simple Language for Parallel Programming. In *J. L. Rosenfeld, editor, Information Processing 74: Proc. IFIP Congress 74, North-Holland*, pages 471–475, August 1974.

[2] E.A. de Kock et al., "*YAPI: Application modelling for signal processing systems*". In Proc. of the 37th. Design Automation Conference, Los Angeles, CA, June 2000, pages 402–405. IEEE, 2000.

[3] Mobile Industry Processor Interface Alliance: www.mipi.org

[4] J. Ajanovic and Hong Jiang, "*Multimedia and Quality of Service Support in PCI Express Architecture*", White Paper, Intel Corporation, September 19, 2002

[5] "*PCI Express Base Specification, Revision 1.0*", PCI-SIG, July 2002, www.pcisig.org

[6] Micron's 128-Mbit DDRRAM specifications, http://download.micron.com/pdf/datasheets/dram/ddr/128MbDDRx4x8x16.pdf