A Bandwidth Optimized SDRAM Controller for the MORPHEUS Reconfigurable Architecture

Sean Whitty, Rolf Ernst
Institute of Computer and Communication Network Engineering
Technical University of Braunschweig, Germany
{whitty | ernst}@ida.ing.tu-bs.de

Abstract

High-end applications designed for the MORPHEUS computing platform require a massive amount of memory and memory throughput to fully demonstrate MORPHEUS's potential as a high-performance reconfigurable architecture. For example, a proposed film grain noise reduction application for high definition video, which is composed of multiple image processing tasks, requires enormous data rates due to its large input image size and real-time processing constraints. To meet these requirements and to eliminate external memory bottlenecks, a bandwidth-optimized DDR-SDRAM memory controller has been designed for use with the MORPHEUS platform and its Network On Chip interconnect. This paper describes the controller's design requirements and architecture, including the interface to the Network On Chip and the two-stage memory access scheduler, and presents relevant experiments and performance figures.

1 Introduction

Reconfigurable architectures have opened the door to exciting new research directions and application domains, many of which have been heavily investigated in recent years. One such project, the "Multi-purpose Dynamically Reconfigurable Platform for Intensive Heterogeneous Processing" (MORPHEUS) project, is a European Integrated Project (IST 027342) which addresses innovative solutions for embedded computing based on a dynamically reconfigurable platform and a corresponding toolset [14]. Its goal is to provide a flexible heterogeneous platform for HW/SW co-design via a unique architecture, composed of reconfigurable computing units of varying granularity, as well as an integrated toolset that can be utilized to easily map and implement target applications.

The potential of the MORPHEUS platform will be demonstrated in several application domains. These include reconfigurable broadband wireless access and network routing systems, processing for intelligent cameras used in security applications, and film grain noise reduction for use in high definition video. The image-based applications have been shown to exhibit immense memory needs. For example, digital film applications require an image resolution of 2K [1], with data rates of up to 2.1 GiBit/s necessary for real-time operation. Higher resolutions of up to 4K and even 8K are on the horizon [15].

Satisfying such memory requirements is no easy task. SDRAM interfaces have long been a performance bottleneck, especially in network processing and multimedia applications. A recurring issue with modern DRAM architectures is relatively long access latencies [6]. DDR-SDRAM and DirectRamBus DRAM (RDRAM) attempt to reduce these latencies by accessing several consecutive data words. This access technique, however, only lowers latencies and does not increase throughput. To this end, optimizations such as *bank interleaving*, which exploits the internal structure of DRAMs by accessing a second bank while another is busy, and *request bundling*, or the grouping of reads and write requests into groups, can be used to ensure maximum possible throughput across the SDRAM data bus.

Using such techniques to increase throughput naturally increases access latencies, as do complex access patterns. However, applications developed for the MORPHEUS platform are data rate hungry and can tolerate such latencies. Furthermore, the onboard ARM processor is used for control purposes and is not expected to make extended use of external memory. Therefore, a bandwidth-optimized memory controller, designed to serve the needs of high-performance reconfigurable architectures such as the MORPHEUS platform, is presented in this paper. For flexibility, the design also supports multiple service levels to reduce latency when necessary. After a brief overview of related work in Section 2, the memory controller requirements are outlined in Section 3, and the architecture itself is defined in Section 4. Section 5 examines synthesis and performance results.

2 Related Work

For the MORPHEUS platform, the ARM PrimeCell MultiPort DDR Memory Controller [2] was considered as an alternative solution to a custom SDRAM controller. Its advantages include basic quality of service features and 8 available client ports. However, this controller was eventually found not suitable because it supports only a 32-bit

[1] 2K implies 2048x1568 pixels/frame, 30 bits/pixel, and 24 frames/s

data bus, does not support DDR access and most importantly, has no support for bank interleaving to optimize data throughput.

The Imagine processor [1] uses a configurable memory scheduler [8, 12], which can be optimized for applications designed for the processor. The scheduler, however, is optimized for specific algorithms, making it less flexible. The Prophid architecture in [9] describes a memory controller for a DSP platform that focuses on data streams using large FIFO buffers and round-robin arbitration. Taking a different approach, Mishra et al. [11] explore optimization techniques for known memory access patterns of a single processor. None of these schedulers, however, support vastly different access types at close to peak SDRAM throughput.

Similar to our design is a memory controller IP offered by Sonics [13], which handles different access patterns at high average memory throughput. It also services more than one priority level. Achieved data rates reflect the performance of our design, which is very close to theoretical maximum values available with DDR-SDRAM. The Sonics solution, however, is a complex seven-stage architecture, which drastically increases latencies over the lean two-stage architecture used in the MORPHEUS SDRAM controller when clock frequencies are roughly equal.

This work is based on the mixed QoS SDRAM controller for FPGA-based image processing described in [5]. However, the controller has now been adapted for the MORPHEUS architecture and optimized to provide maximum throughput to the NoC interconnect.

3 Requirements

3.1 Context

Post-production for high definition video and digital film processing in general require real-time or close to real-time behavior in order to allow immediate feedback necessary for interactive processing. This represents a significant challenge, since the algorithms employed in each step of a digital film processing chain tend to be highly computationally intensive. Performance requirements are far beyond what current DSPs or general purpose processors can provide, especially with large image sizes. Consequently, typical state-of-the-art products in this low-volume, high-price market use FPGA-based hardware systems with fixed configurations. This implies that multiple hardware platforms are required to implement the complete post-production processing chain.

A novel idea is to develop a common platform for all processing steps of the post-production chain. Such a platform has to be reconfigurable to support each of the required algorithms. Furthermore, this platform has to provide the necessary processing power as well as data throughput.

3.2 Application Requirements

The MORPHEUS project implements a key step in the digital film processing chain: film grain noise reduction. This application, amongst others, has been previously implemented in the FlexFilm project [4]. The FlexFilm architecture is a multiple-FPGA design, using three Xilinx Virtex-II units to efficiently distribute the image processing algorithms. In this respect, the FlexFilm architecture is similar to the MORPHEUS platform, which uses three heterogeneous reconfigurable entities (HREs) to carry out processing tasks. The FlexFilm application set consists of several individual algorithms that may or may not be applied to a particular film or video sequence. It is typical that only a subset of these algorithms is used.

The noise reduction application will be mapped across the three MORPHEUS HREs using a distribution like that found in the FlexFilm project. Therefore, data transfer requirements between processing units and to and from external memory will remain similar.

The application must process at least 3 MiPixel per frame and 24 frames per second (assuming a 2048x1568 resolution). This results in approximately 5 to 200 G/operations per second, depending on the complexity of the chosen algorithm. The operations are usually simple arithmetic tasks (e.g. addition, subtraction, and modulus) and frequently access image data.

Additionally, the application requires a significant amount of memory, specifically for the frame buffers required by the motion estimation/motion compensation stage and the synchronization buffers needed by the discreet wavelet transform filtering. These buffers are too large for internal RAM. Consequently, a large external SDRAM is required to meet storage requirements. Furthermore, the memory controller must support a minimum of 2.1 GiBit/s per channel to avoid application data starvation.

3.3 Reconfiguration Needs

A minimum requirement for successful operation of a post-production digital film processing chain is a user-driven reconfiguration between steps of the algorithm. This reconfiguration is intended to allow multiple processing steps to share one processing platform in a time-multiplex fashion. An additional requirement, which is not mandatory, is a data-dependent selection of processing parameters and corresponding adaptation of the algorithms, e.g. the adaptation and adjustment of filter orders, during run-time. Due to its run-time reconfiguration capabilities, the MORPHEUS platform can support this requirement as well.

4 Architecture

The MORPHEUS SDRAM controller (CMC) [2] consists of three main components: the NoC-CMC interface, the two-stage buffered memory access scheduler, and the DDR-SDRAM interface. An architectural overview is shown in Figure 1.

[2] CMC stands for Central Memory Controller

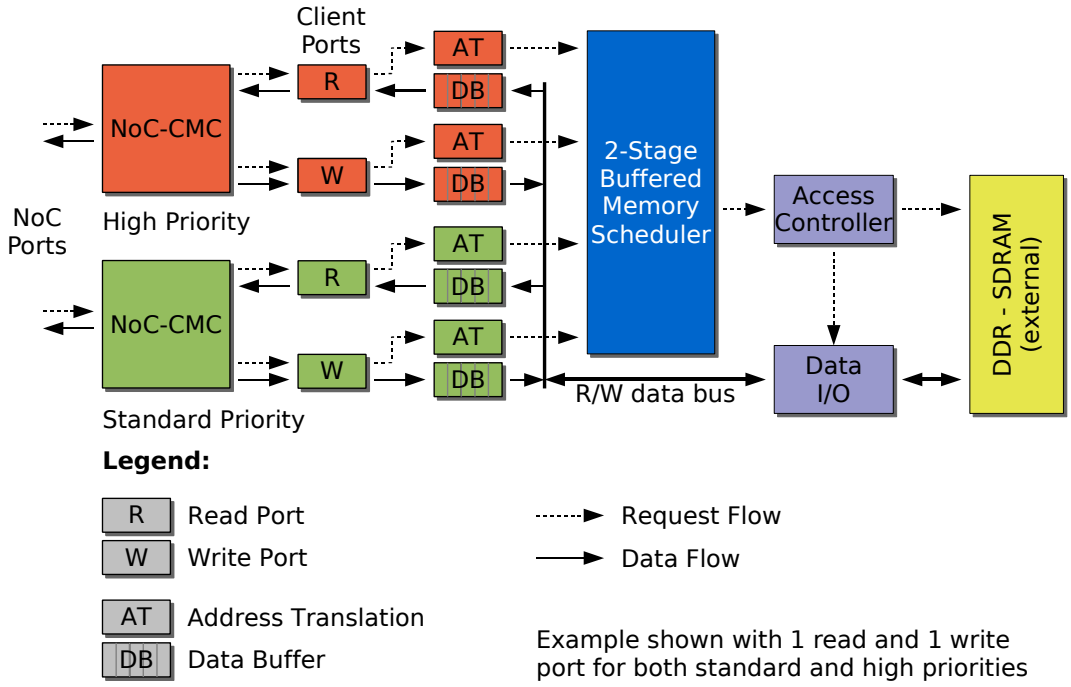

Figure 1: SDRAM controller architecture

4.1 General Architecture

Requests to the SDRAM controller are made by applications via the MORPHEUS Data Protocol interface, which provides the connection from the MOPRHEUS Network On Chip (NoC) to a configurable number (up to 8) of *application read* and *write ports*. The MORPHEUS NoC is based on the STNoC Network On Chip described in [3].

Many applications can perform concurrent memory accesses, however it is *not guaranteed* that requests *to the same memory address from different ports* are executed in order. See also Section 4.6.

Memory access requests first enter the `NoC-CMC Interface`, where read and write requests from MORPHEUS applications are buffered and converted from NoC packets to regular CMC read and write requests and sent to the CMC in burst request format (see Section 4.2). A CMC burst consists of 8 consecutive data words, while a data word is 64 bits in length.

After entering the CMC, memory access requests first reach the *address translator*, where the logical address is translated into the physical rank/bank/row/column quadruple required by the SDRAM, where a "rank" designates a single or group of SDRAM modules controlled by a unique chip select signal.

Concurrently, at the *data buffers*, the write request data is stored until the request has been scheduled; for read requests a buffer slot for the data read from the SDRAM is reserved.

The requests then enter the core part of the SDRAM controller, the two-stage buffered memory access scheduler (see Section 4.3). After one request is selected, it is executed by the *access controller* and the data transfer to/from the corresponding data buffer is initiated by the *data I/O* module. Finally, external data transport and signal synchronization for DDR transfers is managed by the `DDR Interface` and its 64-bit data bus (see Section 4.5).

4.2 NoC-CMC Interface

The `NoC-CMC Interface` provides the connection between the CMC and the MORPHEUS Network On Chip. The interface is responsible for ensuring adherence of NoC requests and CMC responses to their respective transfer protocols, as well as the internal buffering of memory requests and data. In addition, it serves as an error detection unit by identifying unsupported CMC commands and issuing the proper error response to the transfer initiator. The interface consists of four components: *Command Dispatch*, *Command Buffer Read*, *Command Buffer Write* and *Response Dispatch*.

4.2.1 Command Dispatch

The Command Dispatch module is responsible for ensuring the validity of incoming requests, the forwarding of valid requests, and the generation of error response packets. If a request is deemed valid, it is forwarded to the proper Command Buffer. If an error is detected, the Command Dispatch module generates an error response packet, which is transmitted to the Response Dispatch module. See Table 1 for a list of supported commands. It should be noted that the largest commands, `ST64` and `LD64`, exactly correspond to the size of one CMC burst transfer.

4.2.2 Command Buffer Write

The Command Buffer Write module accepts incoming write requests from Command Dispatch and transfers these requests in *burst format* to the CMC.

The internal buffer structure is divided into a data and address buffer. Write requests, which can have different word lengths, require only a single address, which is independent of request size. Therefore, the sizes of the two buffers are independent. The only requirement is that the data buffer should always be able to store more than one complete burst. Also, because applications can best utilize the CMC's scheduling optimizations when working with `ST64` commands (8 data words, a complete burst), an address buffer should be $\frac{1}{8}$ the size of the data buffer.

When data is written from the buffer to the CMC, the conversion of the request into a burst-oriented CMC request takes place. If a request is smaller than a burst, the additional words remaining in the burst, which can be technically deemed "empty," are masked out when the data is written to SDRAM. The data masking is accomplished using a well-known technique that is novel to memory controller data paths. The Command Buffer Write module appends a single "valid" bit to each data word when creating a burst request for the CMC. By increasing the size of a write data word by this single bit, the mask is effectively transported through the memory controller and delivered to the `DDR Interface` without any modifications to the controller core architecture. This bit is finally stripped from the data word before the memory write occurs.

4.2.3 Command Buffer Read

The Command Buffer Read module receives read requests from Command Dispatch and forwards them to the CMC. In addition, it is responsible for receiving data words read from the CMC and converting this burst-oriented data into valid NoC response packets.

The main component of the Command Buffer Read module is a temporary storage FIFO. Using multiple pointers, this buffer keeps track of the position for the next incoming request, the position of the next request that will be sent to the CMC, and the position of the read request that next expects to receive read data from the CMC. A relatively simple pointer management is possible due to the internal CMC re-ordering of read requests into their original request order.

Read requests that are smaller than a complete burst are treated similarly as in Command Buffer Write. Additionally, only the burst address of the request is passed to the CMC. A counter initialized to the lowest three address bits is used to define the position of the first valid data word inside a burst. A second counter, which is set to the number of words to be read, defines the final data word to be read.

Supported memory commands

Signal Type	*Command length (bytes)*
Write Access	`ST08, ST16, ST32, ST64`
Read Access	`LD08, LD16, LD32, LD64`

Table 1: Supported memory commands

4.2.4 Response Dispatch

Response Dispatch controls the distribution of response packets to the NoC, as well as the management of the response packet control signals. Since each of the other three `NoC-CMC Interface` modules can transmit a response packet, the Response Dispatch module has been implemented with two separate buffers and a simple arbitration algorithm. The first buffer handles write responses and error responses, which always consist of a single cell since no real data is transmitted. The second buffer is responsible for read responses, which can consist of 1-8 data words. The transfer of the outgoing packet from the buffers to the NoC is done using round-robin arbitration.

4.3 Two-Stage Buffered Scheduler

The two-stage buffered scheduler comprises the core of the memory controller, performing access optimizations and eventually issuing requests to SDRAM. Figure 2 shows the scheduling stages.

4.3.1 Request Buffer, Request Scheduler

The single-slot request buffers are used to decouple the clients from the following scheduling stages. The first scheduler stage, the request scheduler, selects requests from these buffers, one request per two clock cycles, and forwards them to the bank buffer FIFOs. By applying a round-robin arbitration policy, a minimum access service level is guaranteed. As stated above, high priority requests are serviced before standard priority requests when priority levels are enabled.

4.3.2 Bank Buffer, Bank Scheduler

The bank buffer FIFOs, one for each bank, store the requests according to the addressed bank. The second scheduler stage, the bank scheduler, selects requests from these bank buffer FIFOs and forwards them to the access controller for execution. In order to increase throughput utilization, the bank scheduler performs *bank interleaving* to hide bank access latencies and *request bundling* to minimize stalls caused by read-write switches.

Bank Interleaving exploits the SDRAM structure, which is organized into independent memory banks. SDRAM banks require 4 (read) to 6 (write) passive cycles after a data transfer, during which the active bank cannot be accessed. By reordering memory requests to ensure consecutive accesses occur to inactive banks, a second bank can be accessed during such idle times, effectively hiding these latencies and significantly increasing data rates. As a simple example, one bank can be performing a read access while a second bank is activated. When the first bank completes its data access, the read to the second bank is issued immediately.

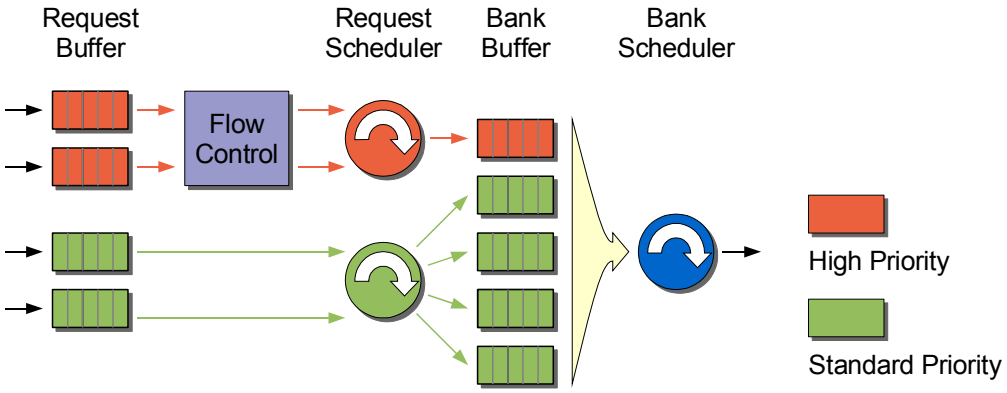

Figure 2: Two-stage buffered scheduler

Request Bundling minimizes the effects of idle cycles required during bus direction switches. These stalls (1 for a read-write change, 1-2 for a write-read change, depending on the SDRAM module) can decrease overall throughput by up to 27% [5]. By bundling like requests together into continuous blocks, these stalls can be avoided.

4.4 Quality of Service (QoS)

While not a consideration for the MORPHEUS platform, Quality of Service is important for modern SDRAM controllers. In general, CPU cache miss and data path memory requests show different memory access patterns. For effective operation, CPU cache miss memory accesses should be served with a *smallest possible latency*, while data path memory requests should be served with a *guaranteed minimum throughput at guaranteed maximum latency*. A more detailed explanation can be found in [5] and [6].

To handle these requirements, two priority levels for memory access requests have been implemented in the CMC. High priority requests (*smallest possible latency*) are always executed before standard priority requests. This is implemented via distinct access paths for high and standard priority requests and a modified bank scheduler which always executes high priority requests first.

With any priority-based design, starvation at the lower levels is often an issue. To avoid possible starvation of standard priority requests (*guaranteed minimum throughput at guaranteed maximum latency*), a *flow control* unit is used to reduce the maximum throughput of high priority requests. The flow control unit can be configured to pass n requests within T clock cycles (known as "sliding window" in networking applications) to allow bursty CPU memory accesses when necessary.

4.5 DDR Interface

A significant challenge for the CMC design was the timing synchronization along the DDR-SDRAM data path.

For flexibility, the `DDR Interface` was designed to be able to adapt to many timing requirements. For this purpose, three features were included in the interface:

- Delay elements (DLLs) used to create proper timing of the Data Strobe signals (`dqs`)
- Flexible Capture Unit for transferring data with system clock
- Moving Data Valid Window for acquisition of data on the internal data bus

In order to briefly describe the interface, as well as illustrate potential timing problems and our solutions, SDRAM write and SDRAM read accesses will be examined below. Consult [7] for more information.

4.5.1 SDRAM Write Access

Figure 3 represents a simplified example of a *write request*. A burst of 8 data words (64-bit word size) is written into DDR-SDRAM. The first data word is masked out (`dqm` = 1), meaning this position is not overwritten in memory. The Data `dq` and the corresponding Data Strobe `dqs'`, which is the reference signal for data acquisition by the memory, are generated by the CMC and therefore synchronous to the system clock. If we assume the simplification that the PCB trace for the `dq` and `dqs'` signals are the same length, meaning that the data arrives at the same time as the system clock, it becomes clear that the Setup Time is violated. In this case, the DLLs shift the `dqs'` signal to the middle (timing-wise) of the data word in order to generate the most tolerant timing situation possible (the DLLs are also subject to jitter, so the delay is not constant). This creates a new delayed Data Strobe signal called `dqs`.

The delay introduced to the `dqs'` signal can only guarantee that the Data Strobe arrives at the SDRAM after the data itself and therefore that the correct data is registered. It must be noted, however, that different memory modules on the chip can be placed different distances away from the memory controller, amongst other possible tolerance issues related to manufacturing. Therefore, the interface was designed with maximum flexibility in mind. Each of the 8 `dqs` signals has its own DLL, and all DLLs can be configured independently of one another.

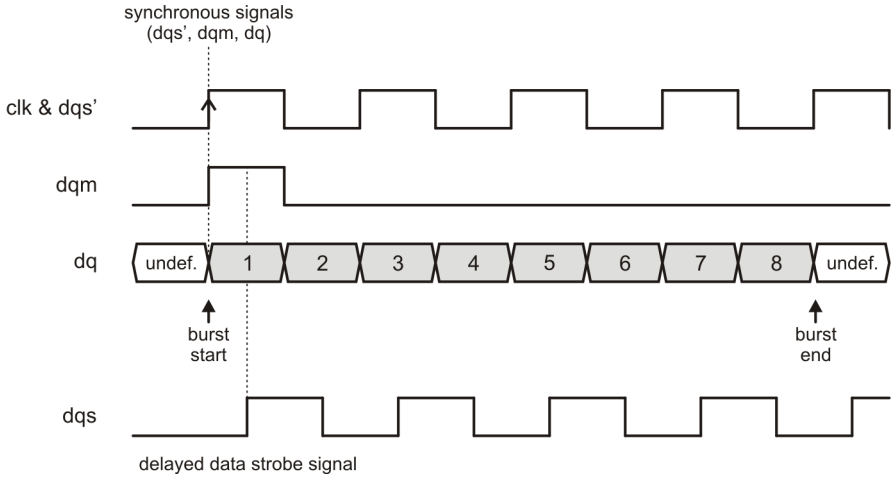

Figure 3: Burst write with data masking

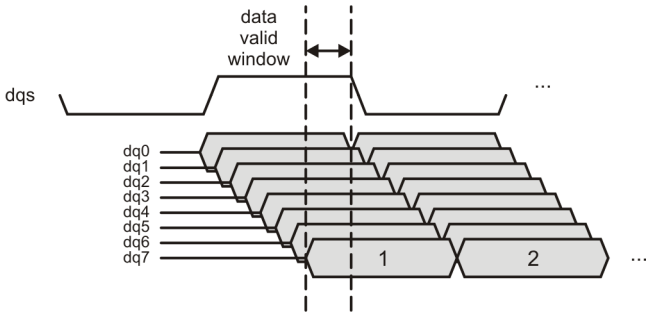

Figure 4: DDR Data Valid Window

Parameter	Description	Unit
t_{CK}	clock period	ns
t_{CH}	clock high-level width minimum	t_{CK}
t_{CL}	clock low-level width minimum	t_{CK}
t_{DQSQ}	DQS to last DQ valid	ns
t_{QHS}	clock half period	ns
t_{QH}	DQ-DQS hold	ns
t_{DV}	data valid window	ns

Table 2: SDRAM Timing Parameters

4.5.2 SDRAM Read Access

Improper timing synchronization can also create communication problems between the CMC and DDR-SDRAM modules during *read requests*. After the CMC sends the `CAS` signal, the memory module does not transfer the data onto the `dq` bus and set the Data Strobe until the amount of time specified by the CAS latency parameter (2, 2.5, or 3 clock cycles) has passed.

The DDR-SDRAM specifications require that all 8 Data signals that belong to a single Data Strobe do not become valid in the same instant. As shown in Figure 4, all Data signals can only be acquired via a DQS edge during a given time window. This time window is known as the *Data Valid Window* and its duration can be calculated as follows, using the SDRAM timings described in Table 2:

$$t_{HP} = min(t_{CH} \cdot t_{CK},\, t_{CL} \cdot t_{CK}) \quad (1)$$

$$t_{QH} = t_{HP} - t_{QHS} \quad (2)$$

$$t_{DV} = t_{QH} - t_{DQSQ} \quad (3)$$

To provide timing synchronization, the Data Strobe signals can be supplied in the middle of the *Data Valid Window*, with the help of the DLL elements provided for the DQS signals generated by the DDR-SDRAM during *read requests*. This, in combination with the three-stage capture unit, allows for proper acquisition of read data.

4.5.3 Alternatives to DLL Usage

As a possible alternative to the use of DLL elements, delays built into the PCB trace itself were considered. Here, the advantage is the removal of the complex logic used to implement the interface using DLLs as well as a significant savings in chip area. However, the disadvantages of this solution prevented it from being considered for serious use. The delay for individual signal lines must be determined via its length on the board. With the memory blocks used during testing [10], a delay of at least 0.85 ns is required. With an approximate propagation speed of $v_{signal} = 20\frac{cm}{ns}$, this would require a signal length of $l = 17cm$. This length is unrealistic for a modern architecture. In addition, for an exact length calculation all delay times must be known, which would force the CMC to be restricted to a specific memory module or in the best case, those modules with almost identical timing behavior.

4.6 Memory Coherency

- Reads and writes from *different ports* to the *same addresses* are potentially executed out-of-order. Within the same priority levels and provided that the bank buffers do not fill up, a distance of $2n$ clock cycles, with n being the number of ports, per priority level, is sufficient to exclude hazards.

- Reads from *one port* to *different addresses* might be executed out-of-order, however they finish in-order. This implies that the application always receives the requested data in-order. The reordering takes place inside the data buffers.
- Writes from *one port* to *different addresses* might be executed out-of-order. This is a non-issue, however, since they occur at different addresses.

4.7 Configuration

CMC configuration parameters clearly depend on the type of DDR-SDRAM used, the clock frequency, and overall board layout. For the MORPHEUS CMC, many parameter values have been determined based on design requirements and cannot be changed. This is in contrast to the flexible nature of the original CMC design, which was created for use in a flexible FPGA environment.

Despite the requirement that many parameters, such as address bus width, data bus width, and the number of application ports must be determined before logic synthesis, a certain degree of flexibility must remain in the MORPHEUS CMC so that it may support different DDR-SDRAM modules and to achieve proper timing under real PCB conditions. To achieve this goal, a programmable `Configspace` module was created, which allows runtime, user-adjustable configuration of SDRAM timing, SDRAM layout, and of the DDR path delay elements used to generate necessary proper timing behavoir for the `DDR Interface`.

5 Experiments and Results

To evaluate the CMC, logic synthesis was first performed using Synopsys Design Compiler X-2005.09-SP4-3. Then, using post-synthesis HDL, both throughput and latency were tested using the MORPHEUS and CMC configuration detailed in 5.1. Experiments were performed in a ModelSim 6.2d environment using the 512MB DDR-SDRAM modules described in [10]. Optimal values for memory controller parameters such as bank buffer sizes (8 buffers of depth 8), data buffer sizes (1 buffer of 512 entries per port), and flow control settings, as well as the effect of request priorities, are provided in [5] and [6]. This allowed our experiments to focus solely on the performance of the memory controller designed for use in the MORPHEUS architecture.

5.1 Resource usage

The MORPHEUS platform is a complex design with numerous integrated IPs, many of which consume significant chip area resources. The CMC was therefore designed to occupy a relatively small chip area and logic was minimized whenever possible. Resource usage heavily depends on the CMC configuration. For the MORPHEUS ASIC design, configuration values are fixed. Therefore, precise resource usage can be reported.

The MORPHEUS CMC relies on the ST Microelectronics 90 nm technology libraries, as well as Delay Locked Loop technology libraries and custom memory modules for internal storage. Custom macros are not included in the synthesizable area report.

Module	Size (KiloGates)
cmc_core	46.2
configspace	1.3
noc-cmc_port_0	26.2
noc-cmc_port_1	26.3
noc-cmc_port_2	26.3
Synthesizable area (KiloGates)	126

Table 3: CMC example resource usage

Table 3 shows the resource usage reported after logic synthesis for the CMC on the MORPHEUS platform with

- 3 NoC-CMC client ports
- 6 standard priority application ports: 3 read, 3 write
- 64-bit client data bus, 64-bit DDR-SDRAM data bus
- 64-bit word size, 8-word burst length
- 200 MHz operating frequency (400 MHz DDR)
- 13-bit SDRAM address bus (13 row, 10 column)
- 4 SDRAM banks
- 2 chip selects
- QoS disabled (prioritization and flow control)
- 8 bank buffers for standard priority requests

The values in Table 3 are well within the target chip area reserved for the CMC and leave room for further improvements and increased internal buffer sizes.

5.2 Throughput

Using access patterns similar to the streaming patterns generated by the film grain noise reduction algorithm outlined in Section 3, both read and write throughput were tested.

The results in Table 4 clearly show that bandwidth is directly correlated to proper utilization of the burst-oriented design of the CMC. Although smaller commands are supported, the controller can best utilize its out-of-order scheduling features when processing a large number of outstanding requests. These values are maximized via `ST64` and `LD64` commands, which correlate to the CMC burst length. With these commands, the CMC data rates come satisfyingly close to the theoretical maximum DDR throughput values at 200 MHz, with a total bandwidth utilization of up to 75%.

As expected, because the CMC's smallest data unit is a burst, requests half the size of a burst access exhibit approximately half the throughput. The slightly lower throughput rates for the larger write requests compared to reads is due to the different designs of the read and

Cmd	Number of Accesses	Throughput (GiBytes/s)
ST64	16384	1.985
ST32	16384	1.306
ST16	16384	0.653
ST08	16384	0.324
LD64	16384	2.360
LD32	16384	1.178
LD16	16384	0.593
LD08	16384	0.295

Table 4: CMC Data Rates

Cmd		Avg. latency (cycles)
ST64		83
ST32		83
ST16		83
ST08		83
Cmd	**Min. latency (cycles)**	**Max. latency (cycles)**
LD64	32	60
LD32	28	63
LD16	26	53
LD08	25	50

Table 5: CMC Latencies

write data buffers and leaves room for future optimization. Smaller burst lengths also negatively impact throughput because with fewer outstanding requests, the scheduler has less flexibility to take advantage of bank interleaving and request bundling, leading to more idle cycles.

Average throughput is therefore application dependent. If an application is designed to heavily utilize `ST64` and `LD64` commands, average throughput approaches the maximum throughput attainable by the CMC.

5.3 Latency

Despite the CMC's focus on optimizing throughput, latency should not be ignored. Large buffer depths have a negative effect on latency, as well as the access optimization techniques employed by the schedulers. However, the CMC's internal FIFOs were kept at reasonable sizes to minimize their effect.

In Table 5, average latencies for write commands are presented. The same access patterns used in the throughput experiments were used to test latency. Here, values were measured from the receipt of an incoming request to the transfer of data across the SDRAM data bus. Because of its burst-oriented design, latencies are identical for write operations of all sizes.

More interesting, however, are read access latencies, which correspond to the time an application must wait for requested data. Read latencies proved to be fully dependent on the size of the read command issued to the controller. As expected, the more data requested, the longer the latency, which was measured from receipt of the request until the transfer of the response packet to the NoC.

6 Conclusion

In this paper, we have presented a novel bandwidth-optimized SDRAM controller for the MORPHEUS heterogeneous reconfigurable platform. An application was described which demonstrates requirements not supported by other memory controllers accessible to the project. Despite the SDRAM controller's focus on throughput and achievement of up to 75% of the theoretical maximum DDR data rate, access latencies also remain low compared to other solutions. In cases where latency is especially critical, the CMC also provides different access paths for high priority and standard priority requests to ensure that latency sensitive tasks, such as CPU cache misses, are serviced as quickly as possible. Finally, despite being developed for the MORPHEUS platform and designed for an ASIC environment, the controller remains flexible. It supports a configurable number of application ports and various DDR-SDRAM modules. It also contains a configurable interface to a NoC, which can be removed for systems not requiring such an advanced interconnect.

References

[1] Jung Ho Ahn, William J. Dally, Brucek Khailany, Ujval J. Kapasi, and Abhishek Das. Evaluating the Imagine Stream Architecture. *SIGARCH Comput. Archit. News*, 32(2):14, 2004.

[2] ARM Ltd. *PrimeCell MultiPort Memory Controller (PL175).* ARM Ltd., 2003.

[3] M. Coppola, R. Locatelli, G. Maruccia, L. Pieralisi, and A. Scandurra. Spidergon: a novel on-chip communication network. In *Proceedings of the International Symposium on System-on-Chip*, pages 16–18, 2004.

[4] Amilcar do Carmo Lucas, Sven Heithecker, and Rolf Ernst. FlexWAFE - A high-end real-time stream processing library for FPGAs. In *DAC '07: Proceedings of the 44th annual conference on Design automation*, pages 916–921, New York, NY, USA, 2007. ACM Press.

[5] Sven Heithecker, Amilcar do Carmo Lucas, and Rolf Ernst. A Mixed QoS SDRAM Controller for FPGA-Based High-End Image Processing. In *Workshop on Signal Processing Systems Design and Implementation*, page TP.11. IEEE, 2003.

[6] Sven Heithecker and Rolf Ernst. Traffic Shaping for an FPGA-Based SDRAM Controller with Complex QoS Requirements. In *Design Automation Conference (DAC)*, pages 575 – 578. ACM, June 2005.

[7] JEDEC. *Double Data Rate (DDR) SDRAM Specification.* JEDEC Solid State Technology Association, jesd79c edition, March 2003.

[8] Brucek Khailany, William J. Dally, and Scott Rixner. Imagine: Media Processing with Streams. *IEEE Micro*, pages 35–46, March/April 2001.

[9] Jeroen A. J. Leitjen, Jef L. van Meerbergen, and Adwin H. Timmer. PROPHID: a Heterogeneous Multi-Processor Architecture for Multimedia. In *International Conference on Computer Design*, pages 164–169, October 1997.

[10] Micron Technology, Inc. 512Mb DDR SDRAM Component: MT46V64M8BN-5B. Data sheet, Micron Technology, Inc., April 2004.

[11] Prabhat Mishra, Peter Grun, and Nikil D. Dutt. Processor-Memory Co-Exploration driven by a Memory-Aware Architecture Description Language. In *14th International Conference on VLSI Design (VLISD01)*, pages 70 – 75. IEEE, Jan 2001.

[12] Scott Rixner, William J. Dally, and Ujval J. Kapasi. Memory Access Scheduling. In *International Symposium on Computer Architecture*, pages 128–138, 2000.

[13] Sonics Inc. Sonics MemMax 2.0 Multi-threaded DRAM Access Scheduler. Data sheet, Sonics Inc., 2005.

[14] F. Thoma, Matthias Kühnle, Philippe Bonnot, Elena Moscu Panainte, Koen Bertels, Sebastian Goller, Axel Schneider, Stéphane Guyetant, Eberhard Schüler, Klaus D. Müller-Glaser, and Jürgen Becker. MORPHEUS: Heterogeneous Reconfigurable Computing. In *Proceedings of 17th International Conference on Field Programmable Logic and Applications (FPL07)*, August 2007.

[15] Thomson Grass Valley. Homepage. http://www.thomsongrassvalley.com, 2007.