Traffic Shaping for an FPGA based SDRAM Controller with Complex QoS Requirements *

Institute of Computer and Communication Network Engineering
Technical University of Braunschweig

Dipl.-Ing. Sven Heithecker
heithecker@ida.ing.tu-bs.de

Prof. Dr.-Ing. Rolf Ernst
ernst@ida.ing.tu-bs.de

ABSTRACT

Today high-end video and multimedia processing applications require huge amounts of memory. For cost reasons, the usage of conventional dynamic RAM (SDRAM) is preferred. However, SDRAM access optimization is a complex task, especially if multi-stream access with different QoS requirements is involved. In [8], a multi-stream DDR-SDRAM controller IP covering combinations of low latency requirements for processor cache access, hard real-time constraints for periodic video signals and hard real-time bursty accesses for video coprocessors was described. To handle these contradictory QoS requirements at high system performance, a combination of a 2-stage scheduling algorithm and static priorities were used. This paper describes an additional flow control which enhances the overall performance. Experiments with an FPGA based high-end video platform demonstrate the superiority of this architecture.

Categories and Subject Descriptors

C.4 [**Computer Systems Organization**]: Performance of Systems; B.8.3 [**Hardware**]: Performance and Reliability—*Performance Analysis and Design Aids*; C.3 [**Computer Systems Organization**]: Special-Purpose and Application-Based Systems

General Terms

performance, design

Keywords

SDRAM, memory access, QoS, traffic shaping, priorities, flow control, FPGA

*This work was partly founded by German BMBF and Thomson - Grass Valley

1. INTRODUCTION

Dynamic RAM memories are important embedded system components that cannot easily be integrated with other parts of a system due to different technologies. The resulting SDRAM[1] interface is a performance bottleneck in many applications ranging from network processors to multimedia systems. The main problem of all DRAM architectures is a long access latency. Current advanced DRAM architectures, such as DDR-SDRAM or DirectRamBus DRAM (RDRAM), reduce the latency overhead by using burst access to several consecutive data words in a memory row. The effect of access latency can be further reduced by exploiting the internal bank structure of a DRAM serving accesses to another bank while one bank is busy accessing. This technique, called interleaving, increases memory bandwidth. It has already been used in early vector processors and has been introduced to digital signal processing and even SoC architectures, e.g. in the SonicsMicroNetwork [15].

While improving DRAM memory throughput, however, burst and interleaving increase memory latency. This is due to access buffering needed to enable these techniques. In a mixed platform containing stream processing elements and processors with caches, increased latency is typically acceptable for streaming data, but it deteriorates cache processor performance. With even higher throughputs and more complex access patterns, memory latency grows due to larger buffers and higher scheduler complexity that leads to further processor cache stall problems.

Considering our FlexFilm flexible digital film processing platform [3] as an example, our contribution to an improved SDRAM performance will be explained and quantified. Digital film processing applications require 2K image resolution[2] with a data-rate of 2.1 GBit per second and channel [5], [2], while higher resolutions of 4K and even 8K are on the way [6]. Real-time processing at this data rate is beyond the scope of today's workstations and single DSP processors, and ASICS are not economically viable due to the small market volume. Therefore, an FPGA-based system approach as shown in figure 1 was followed. The core component of this architecture is a large FPGA[3] containing the reconfigurable main image stream processing *data path*. The FPGA-embedded CPU is used for control and less computation intensive tasks like parameter calculation. Since FPGAs do not provide enough internal memories to hold even a single image, external SDRAM memories are provided, which also contain the code and data for the CPU. This leads to conflicts between processor accesses to code, to internal data, and to shared image data on the one hand and memory ac-

[1] This term shall include both SDR- and DDR-SDRAM
[2] 2048x1536 pixels per frame at 30 bit/pixel and 24 pictures/s
[3] e.g. Xilinx Virtex-II Pro with embedded PowerPC

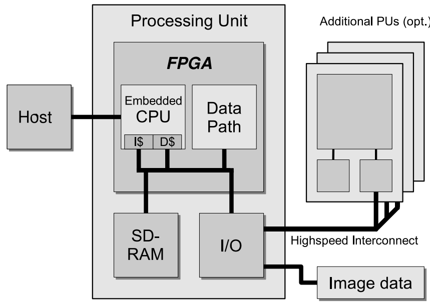

Figure 1: FlexFilm - flexible image processing platform

cesses of the data path on the other hand. In principle, there can be more than one external memory, but the required address busses and 32-64 bit data busses make extensions costly and very limited. Multiple independent memories also do not simplify access patterns since there are still shared data between data path and CPU. Therefore a single external memory is assumed in this paper.

To solve the access problem, previously [8] a 2 stage memory controller architecture with short access latency at high bandwidth utilization was proposed. In this paper we add a flow control mechanism that improves the real-time behavior allowing both high throughput real-time access and reduced cache stalling.

Chapter 2 elaborates the requirements and outlines related work. Chapter 3 explains the architecture and flow control. Experiments are described in Chapter 4 while the last chapter draws a conclusion.

2. REQUIREMENTS, RELATED WORK

2.1 QoS Requirements

A closer look at the FlexFilm architecture reveals that the following distinct types of accesses have to be served by the SDRAM controller:

- *Hard real-time periodic access sequences* and *fixed address access patterns* with optimized fixed scheduling [13] generated by the I/O system and data path. These sequences require a *minimum memory throughput* and can be buffered to increase the maximum allowed memory access time without performance loss. By adapting the buffer size, arbitrary access times are acceptable, but the maximum access time must be bounded to avoid buffer over- or underflows.

- *Random address patterns* generated by *CPU cache misses*, either soft- or hard real-time. Because the processor stalls on a cache miss and since prefetch is of limited use due to the less predictable access patterns, memory access time is the crucial parameter determining performance. On the other hand, (average) memory throughput is less significant. To improve performance, memory access time has to be minimized (soft real-time) or bounded (hard real-time). Such requirements are typically neglected but they are important for the overall system performance, as will be seen in the experiments.

This results in two types of QoS: *guaranteed minimum throughput at guaranteed maximum latency* and *smallest possible latency*.

2.2 Related Work

The Imagine processor [7] uses a configurable memory scheduler [9][14], optimized for the application algorithms that run on the processor. The scheduler is adapted to a specific application and does not distinguish different stream types. The Prophid architecture[11] [17] by Meerbergen et al. describes a dynamic RAM scheduler for the Prophid DSP platform that is focused on streams using large FIFO buffers and round-robin scheduling. Prabhat Mishra et al. [12] provide optimization heuristics for known memory access patterns of a single processor. None of these schedulers support vastly different access types at close to peak SDRAM bandwidth.

Closest to our work is a memory scheduler IP offered by Sonics [18] that also handles different access patterns and service levels at high average memory bandwidth. The bandwidth is similar to that of the scheduler presented here and is close to the maximum possible bandwidth. While high bandwidth can always be reached by a sufficiently long pipeline, latency is the key problem to reduce cache stalling. The Sonics IP is a complex, 7 stage architecture that has an inherently longer latency than the lean 2-stage architecture of similar clock frequency used in [8] and in this paper. Even though the authors of [18] highlight the importance of low latency access to avoid cache delays, they do not elaborate on the latency effect of the relatively long pipeline (In [18] only memory bandwidth and no latencies are published).

3. ARCHITECTURE

Figure 2 shows the block diagram. A more detailed description of the general architecture can be found in [8], this chapter focuses on the QoS implementation.

Requests to the SDRAM are always served at full SDRAM burst lengths in autoprecharge mode. The core part of the controller is the 2-stage buffered memory access controller where streams from various inputs are merged and sorted by bank (request scheduler) and finally scheduled for execution (bank scheduler). To increase throughput and to hide the SDRAM latencies, the scheduler exploits bank parallelism by applying bank interleaving and minimizes data bus direction switches by bundling read/write requests.

As explained in the introduction, two types of QoS requirements have to be handled: *guaranteed minimum throughput at guaranteed maximum latency* and *smallest possible latency*. To handle this requirement, priorities are assigned to memory access requests and requests with a higher priority (*smallest possible latency*) are always executed before lower priorized requests. In this paper, two priority levels are used: *standard priority* and *high priority*.

The key QoS handling extensions are distinct paths, including separate request- and bank buffers, for high- and standard priority requests and a modified bank scheduler. In [8] it was demonstrated that cache memory access times can be reduced significantly by giving cache accesses a higher priority than data path accesses.

As long as the average cache miss rate is relatively low, this approach leads to higher processor performance without any impacts on the real-time streams. The reason is that even if the misses occur in the form of bursts, they can be compensated by buffering the accesses from other sources. In case of higher cache miss rates, buffers must be bigger to avoid data loss in lower priority real-time access. However, buffering video streams in FPGAs is expensive. One approach is a scheduler that limits the execution frequency of *software processes* and, hence, cache miss frequencies, however such an indirect control approach would require a rather conservative design.

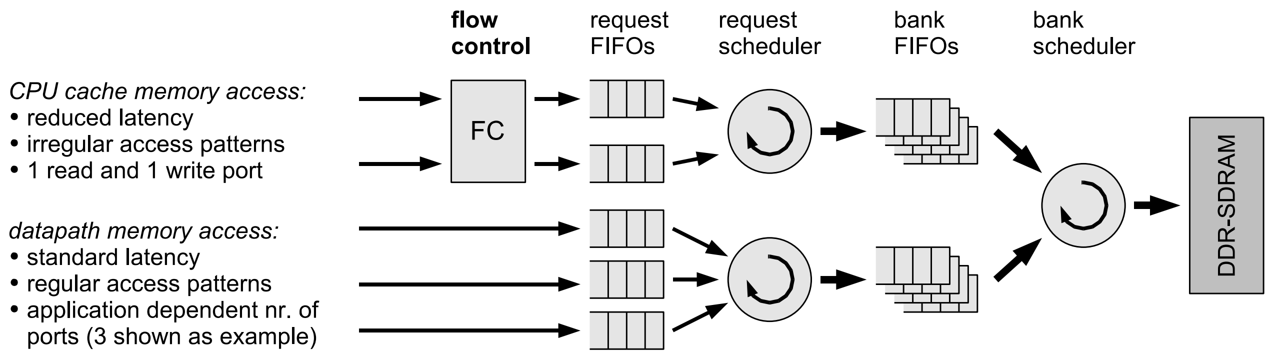

Figure 2: SDRAM controller architecture

An alternative solution, presented in this paper, is to insert a flow control unit into the memory scheduler (see "flow control" in figure 2). The simplest flow control is to require a minimum time distance between two memory read or write accesses. This, however, does not account for burst-like memory accesses closely following each other. A slightly more complicated flow control mechanism considers these access sequences. Appropriate "burst" event models are known from real-time analysis, e.g. [16] (not to be confused with SDRAM burst transfers). The sequences are defined by an outer period T, a number of events n that can occur during T, and an inner period t that defines the minimum distance between any two events. In our design, t is set to 1.

As an example, take $n = 1$, $T = 4$: only one request is allowed within a period of 4 clock cycles. Since, for a DDR burst length of 8, the bus tramsfer time of a burst is 4 clock cycles, this is also the maximum rate at which requests can be processed, resulting in a throughput of 2 words/cycle (DDR-SDRAM data bus utilization of 100%).

Now, take $n = 2$, $T = 16$: two requests within 16 clock cycles. This results in an average of 1 request per 8 clock cycles ($T_\phi = \frac{T}{n} = 8$), but it allows for 2 successive memory requests.

For the current implementation, only the high priority stream is flow controlled and both read and write paths share one flow control unit; it might be conceivable to control read and write paths separately and to have independent flow control units for different clients or priority levels. However, this would not alter the flow control principle, and therefore in this paper only one flow control unit is used.

4. EXPERIMENTS

4.1 Goals & Simulation Environment

We were interested in the impact of priorities and flow control on different memory access streams. For this purpose an example architecture which resembles the setup in figure 1 was developed as a clock cycle accurate SystemC model. The experimental setup is shown in figure 3. Beside the DDR-SDRAM controller and the DDR-SDRAM, it consists of a hardware implementation of a discrete wavelet transformation (DWT) algorithm and a CPU with caches.

Besides the timing model, synthezisable models of the SDRAM controller and DWT core for Xilinx Virtex II FPGAs were created using the *Xilinx ISE 6.3* toolsuite and implemented in real hardware. This allowed us to verify the complete functionality and to validate maximum operating frequencies.

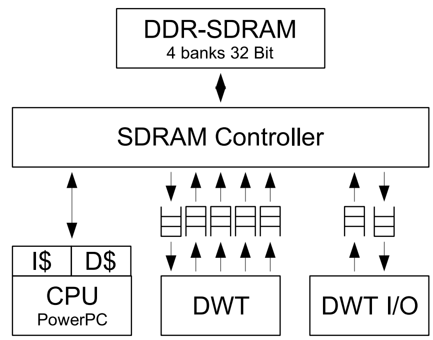

Figure 3: Experimental setup

DDR-SDRAM & Controller. The DDR SDRAM parameters were set to 120 MHz clock speed (240 MHz data transport), 32 bit data bus, 4 banks, a burst length of 8 words (4 clock cycles) and a 64 ms refresh. These parameters were kept constant.

DWT and Stream I/O. The DWT implementation based on [19], the VHDL implementation resulted in an maximum frequency of 100 MHz. The DWT read a strictly periodic data stream of 2 pixels (32 bit) per clock cycle through one input port and wrote strictly periodic data streams to 4 outputs ports with $1/4$ of the input data rate. Data transfers from and to SDRAM run via five 1024 pixel-FIFOs, which are also used to cross the different clock domains (120 MHz SDRAM, 100 MHz DWT), to allow deep input data prefetching, to convert to and from the bursty SDRAM access and to compensate varying memory access times.

For image input and output, two additional ports read and write a periodic pixel stream (2 pixel, 32 bit per clock cycle). Both streams are buffered by FIFOs of the same properties and functionality as the DWT FIFOs.

The complete DWT system was set up for a pipelined continuous 3-level DWT of 2048 x 1536 12-bit-luminance images at 24 frames per second (41.7 ms per frame), resulting in an I/O frequency of 37 MHz. A 3-level DWT needs to process each image three times width the image height and with being halved each turn. Running at 100 MHz, the DWT needs 20.6 ms per frame.

CPU and Cache. As CPU model the PowerPC 750 simulator from MicroLib [1] was used and adapted to the PowerPC 405 processor provided by VirtexII-Pro FPGAs (one execution unit, simplified static branch prediction, 16 KByte 2-way associative instruction and data caches). Since prefetching is of limited use, the CPU is directly connected to the SDRAM controller. The selected "peg-

CPU pri	CPU has higher memory access priority yes/no
fc T_ϕ/n	flow control setting: maximum of n consecutive memory accesses, average one memory access per $T_\phi = \frac{T}{n}$ clock cycles
CPU cycles	CPU execution time in million CPU clock cycles (360 MHz)
CPU SpU	CPU speedup relative to non-priorized CPU memory access
Mem lat r/w	average cache to memory access latency, read and write, in memory clock cycles
loss	DWT lost pixels, average percent of pixel per image

Table 1: Results, legend

Nr.	CPU pri	fc T_ϕ/n	CPU cycles	SpU	Mem lat r	lat w	loss [%]
1.1	no	none	195.02		96.7	32.6	0.00
1.2	yes	none	63.43	3.07	17.5	11.4	10.35
2.1	yes	10/1	63.02	3.09	17.3	11.3	10.80
2.2	yes	20/1	64.93	3.00	18.0	17.1	10.81
2.3	yes	30/1	71.23	2.74	21.2	27.4	10.71
2.4	yes	40/1	84.55	2.31	28.7	36.8	6.71
2.5	yes	50/1	100.48	1.94	37.8	47.3	3.12
2.6	yes	60/1	117.28	1.66	47.4	56.8	0.00
3.1	yes	60/2	117.00	1.67	47.3	54.5	0.34
3.2	yes	61/2	118.70	1.64	48.3	55.4	0.00
3.3	yes	61/10	117.32	1.66	47.4	55.4	0.00
3.4	yes	61/20	116.89	1.67	47.1	55.1	0.00
3.5	yes	61/30	116.78	1.67	47.0	54.3	0.34

Table 2: Results

witdecode" application from the mediabench [10] benchmark applications had an instruction cache hitrate of 99.99%, a data cache hitrate of 89.8% and generated 518,777 burst read and 35,229 burst write memory accesses. The CPU was clocked at 360 MHz.

4.2 Test Setup & Results

We ran several tests with CPU memory access priorisation enabled versus disabled and with different flow control settings (maximum n consecutive SDRAM requests with an average of one memory request per $T_\phi = \frac{T}{n}$ clock cycles). We recorded the CPU execution time in clock cycles and the number of lost words per image due to possible buffer over- and underflows of the DWT video input and output, respectively. Lost words other than 0% means that the DWT failed to process in real-time. For completeness, the cache to memory access latency was also recorded. Table 2 shows the test results, table 1 the according legend.

With priorites and flow control disabled, it can be seen that CPU does not adversely affect the DWT performance, that means the DWT does not loose any pixels and processes in real-time (1.1). Assigning a higher priority to CPU memory accesses leads to a tremendous CPU speedup of a factor of 3, but the DWT fails to operate with an average pixel loss per image of 10% (1.2).

Since at this point the SDRAM operates at its bandwith limit and the latencies are affected by full buffers in the scheduling stage, activating the flow control with a slight throughput restriction of one burst every 10 clock cycles ($T_\phi = 10$, $n = 1$) does not show any reasonable effects (2.1).

Further increasing of T_ϕ results as expected in a monotonous increase of the CPU execution time, in parallel the DWT pixel losses drop until the DWT is back to real-time behavior again (2.2 to 2.6). At this point (2.6), the CPU still shows a noticeable speedup of 1.66 compared to the non priorized access.

Increasing n to allow for burst-type CPU memory access sequences leads again to a small CPU speedup, even though it was necessary to increase T_ϕ to keep the DWT pixel losses at zero (3.1 to 3.5, best result 3.4)

Throughout all experiments, the total SDRAM bandwidth utilization was close to 75%.

More simulation results with different setups, e.g. an ARM instead of a PowerPC processor, can be found in [4].

5. CONCLUSION

Based on experience with an architecture for a reconfigurable image processing system, an SDRAM scheduler IP was presented that supports several concurrent access sequence types with different requirements including predictable periodic real-time sequences and cache accesses with a minimum latency objective. By introduction of a combination of prioritized scheduling and flow control the real-time behavior and system performance was significantly improved.

We also showed that allowing multiple accesses closely following each other (access bursts) also improved the system performance while the real-time behavior was not affected.

More important, however, is the increased level of controllability. The flow control parameters can be used to smoothly adapt the processor accesses to the required real-time behavior. Flow control in combination with access priorities obviously leads to better system performance predictability. Furthermore, the monotonous behavior should simplify design space exploration.

6. REFERENCES

[1] http://microlib.org.

[2] http://www.discreet.com.

[3] http://www.flexfilm.org.

[4] http://www.ida.ing.tu-bs.de: svenh/.

[5] http://www.quantel.com.

[6] http://www.thomsonbroadcast.com.

[7] Ahn, J. H., Dally, W. J., Khailany, B., Kapasi, U. J., and Das, A. Evaluating the Imagine Stream Architecture. *SIGARCH Comput. Archit. News 32*, 2 (2004), 14.

[8] Heithecker, S., do Carmo Lucas, A., and Ernst, R. A Mixed QoS SDRAM Controller for FPGA-Based High-End Image Processing. In *Workshop on Signal Processing Systems Design and Implementation* (2003), IEEE, p. TP.11.

[9] Khailany, B., Dally, W. J., and Rixner, S. Imagine: Media Processing with Streams. *IEEE Micro* (March/April 2001), 35–46.

[10] Lee, C., Potkonjak, M., and Mangione-Smith, W. H. Mediabench: a Tool for Evaluating and Synthesizing Multimedia and Communicatons Systems. In *International Symposium on Microarchitecture* (1997), pp. 330–335.

[11] Leitjen, J. A. J., van Meerbergen, J. L., and Timmer, A. H. PROPHID: a Heterogeneous Multi-Processor Architecture for Multimedia. In *International Conference on Computer Design* (October 1997), pp. 164–169.

[12] Mishra, P., Grun, P., and Dutt, N. Processor-Memory Co-Exploration driven by a Memory-Aware Architecture Description Language. In *14th International Conference on VLSI Design* (Jan 2001).

[13] Panda, P., Catthoor, F., and Dutt, N. Data and Memory Optimization Techniques for Embedded Systems. 140–206.

[14] Rixner, S., Dally, W. J., and Kapasi, U. J. Memory Access Scheduling. In *International Symposium on Computer Architecture* (2000), pp. 128–138.

[15] Sonics, Inc. Sonics SiliconBackplane MicroNetwork Overview.

[16] Tindell, K. W. An Extendible Approach for Analysing Fixed Priority Hard Real-Time Systems. 133–152.

[17] Verhaegh, W., Lippens, P., and Aarts, E. Multi-dimensional periodic scheduling: model and complexity. Springer Verlag, pp. 226–235.

[18] Weber, W.-D. Sonics MemMax Memory Scheduler.

[19] Wu, P.-C., and Chen, L.-G. An Efficient Architecture for Two-Dimensional Discrete Wavelet Transform. *IEEE Transactions on circuits and systems for video technology 11*, 4 (April 2001).