Modeling a Memory Subsystem with Petri Nets: a Case Study

Matthias Gries

Computer Engineering and Networks Laboratory (TIK),
Swiss Federal Institute of Technology (ETH)
CH-8092 Zürich, Switzerland,
gries@tik.ee.ethz.ch

Abstract. Memory subsystems often turn out to be the main performance bottleneck in current computer systems. Nevertheless, the architectural features of common RAM chips are not utilized to their limits. Therefore, a complex Petri Net model of a memory subsystem was developed and investigated to explore possible improvements. This paper reports the results of a case study in which widely used synchronous RAMs were examined. It demonstrates how impressive throughput increases can be obtained by an enhanced memory controller scheme that could even make second level caches redundant in cost or power dissipation critical systems.
Furthermore, using colored time Petri nets such as they are supported by the CodeSign[1] tool leads to a descriptive view of the memory subsystem because a Petri Net model combines data and control flow as well as structural information in a natural way. The case study finally underpins the advantages of the CodeSign approach.

1 Introduction

The memory subsystem (see Fig. 1) in a computer usually consists of a controller and several memory chips. The controller receives read and write requests from the central processing unit, modifies them, and generates additional instructions for the memory chips if necessary. Common memory chips like Fast Page Mode (FPM) [11] or Extended Data Output (EDO) [11] chips are not able to process several instructions concurrently. Thus, common controllers schedule requests from the CPU sequentially for the memory chips. However, recent trends in memory architectures show an evolution towards concurrent structures, e.g. synchronous DRAM chips (SDRAMs [11]). Ordinary controllers supporting FPM/EDO and SDRAM chips do not distinguish between these two types of memory. Therefore, SDRAMs are rarely utilized to their full potential, that is, they are not capable of obtaining higher throughput rates than FPM and EDO

[1] CodeSign [4] is developed at the Computer Engineering and Networks Laboratory (TIK) and is available at *http://www.tik.ee.ethz.ch/~codesign.*

chips. At the same time, clock rates and throughput demands of modern CPUs grow constantly.

One reaction of the computer manufacturing industry is the increase of the clock frequency of memory interfaces (not the memory array itself). Moreover, memory array sizes can be reduced in order to decrease RAM access delays accepting a certain cost penalty because more arrays are needed in parallel. New trends and a good overview of current DRAM technology are given in [9]. Finally, the costs for cache memories are not negligible within the memory subsystem.

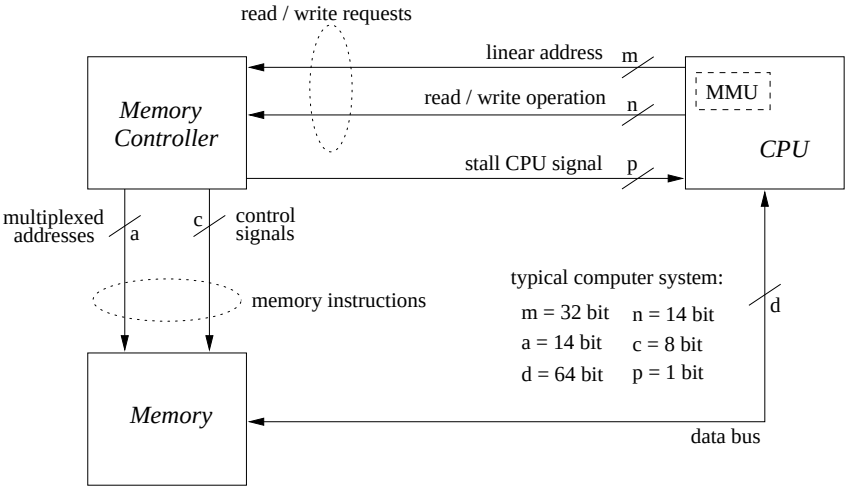

Fig. 1. Generic view of a computer system.

Nevertheless, a major part of wasted memory throughput is explained by the fact that memory controllers are still not able to take advantage of concurrently operating structures within memory chips. The case study in this paper therefore indicates the potential in terms of throughput increases by modifying the memory controller instruction schedules.

Requirements for a modeling tool

The design of complex computer systems usually involves people from different fields of expertise, e.g. control and data flow dominant parts of the system could be developed by separate teams in parallel. Accordingly, the following requirements for a suitable modeling tool can be deduced:

– Support for structuring the model into modules and components and reusability has to be offered.

– The process of modeling must be supported by a natural representation of control and data flow as well as structural properties.
– Simulation and documentation (e.g. getting statistics data out of a simulation run) of the system must be provided.
– Configurability and parameterization of modules and components should be possible.
– Modeling of concurrency and a notion of time are mandatory for the detection of timing violations, for performance evaluation, and for verification of run-time behavior.

Modeling alternatives

In this section, several common modeling formalisms and environments are briefly reviewed.

State machine based approaches, such as StateCharts [5] or ROOMchart [13], are best suited for control dominated systems and suffer from their inability to express data flow. Another discrepancy between a system and its model representation can be found looking at all the tools that do not allow to express structural similarity between a system and its model, e.g. spreadsheet based models and models written exclusively in the form of programming language code. Recently, the use of object-oriented modeling [2],[12] becomes more and more common. Although OO-formalisms contain several features to produce detailed models, they are not intended to be executable as such. Place/Transition Petri nets [10] have several desirable properties, such as being intuitive, graphical, able to express concurrency and data flow. However, they are confined to the use in small scale models because a concept of hierarchy is missing. High-level Petri nets (such as Coloured Petri Nets [8]) are better suited, since they have an expressive inscription language and also some structuring features.

In the next section, a short overview of present memory systems is given including currently available cache and main memory technologies. In section 3, the main features of the CodeSign modeling and simulation tool are outlined and checked against the mentioned requirements. In the remaining part of the paper, section 4, the modeled memory system is presented in detail.

2 Memory bottlenecks

2.1 Commonly used memory chips

Presently, asynchronous memory chips like FPM or EDO RAMs are found in a variety of personal computers, printer devices, workstations, digital assistants, etc. and hence dominate the memory market. The term *asynchronous* describes the fact that a memory controller has to hold control signals for the memory array on the same voltage during the whole memory transfer. In consequence, the controller cannot issue another memory instruction unless the previous one is

finished. That way, even though several memory banks are available on a single chip, the controller is forced to schedule memory instructions strictly sequentially since only one set of control signals can be accessed by the controller at a time.

2.2 Synchronous memories and caches

Synchronous memories: RAMs like SDRAM, SLDRAM [15] or RDRAM [16] are enhanced versions of asynchronous RAMs. They additionally have a synchronous interface that isolates the main memory cell array from the signals of the memory controller. As the interface has one command pipeline for every memory bank, the controller now has the option to issue memory instructions on each clock cycle. Undoubtedly, it is normally not advisable to transfer a memory instruction on each clock cycle since parallel memory banks have to share a single data bus and an input/output buffer pair. But a memory bank can be prepared for a data transfer while another bank is actually transferring data concurrently. This preparation is required due to address decoding and memory access times as well as the fact that dynamic memory cells must be precharged after they were accessed. That is, an intelligent memory controller should be able to hide most of the delays that are caused by access changes between or within memory banks. However, it has to be pointed out that not all delays can be hidden completely. For instance, when the memory access pattern changes from read to write or vice versa, the data pipeline has to be fully cleared until the next access can occur because the direction of data flow through the input/output buffers has changed.

Moreover, if timing constraints of the memory chips (e.g. pipeline and memory array timings) are not met when transferring instructions, there is a danger of data contention inside the memory arrays, that is, memory chips do not check against forbidden control sequences. Furthermore, the controller has the ability to schedule commands in different ways since parallel memory banks can be used in many fashions concurrently (e.g. burst or interleaved data transfers).

Unfortunately, current controllers schedule memory instructions in almost the same sequential manner for synchronous RAMs as for asynchronous ones.

Second level caches: Typical second level caches and synchronous RAMs exhibit almost identical behavior. They are both synchronous and pipelined and can be programmed to burst transfers, that is, data on successive addresses are transferred without the help of the memory controller. In systems where external caches run at the same cycle frequency as the main memory (as in most currently used personal computers), the speed difference is less than a factor of two (see section 4.4 and Tab. 1 for a quick overview). Even though caches are still a little bit faster because their charge does not need to be refreshed using static RAM technology and their memory bank size is usually smaller, an optimized memory controller could compensate the need for caches in power dissipation or cost critical systems.

3 The modeling environment

In this section, the CodeSign tool is presented and it is pointed out how the requirements for a modeling tool are met. A more detailed description can be found in [4]. CodeSign is based on a kind of colored Petri nets that allow efficient modeling of control and data flow.

Components, composition, and hierarchy: Components are subnets of Petri nets with input and output interfaces that are applied to interconnect components. Inside components, input interfaces are connected to places and transitions are connected to output interfaces. Linking output with input interfaces, components are directly interconnected maintaining Petri Net semantics. In Fig. 2 the model of a RAM basic cell with its interfaces is shown as an example. The model is explained in detail in section 4.1.

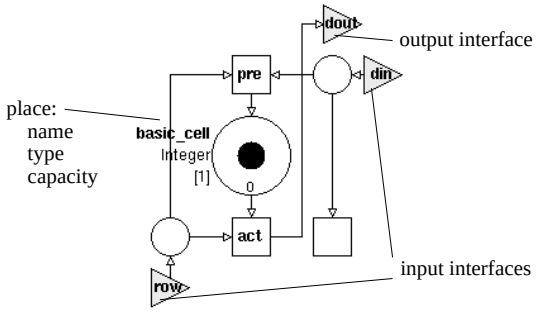

Fig. 2. Petri Net model of a RAM cell component.

If input interfaces of components are connected together, a single token will produce several tokens with the same data value for each component. Connecting output interfaces together is the same as connecting several transitions to a single place. Therefore, the examples in Fig. 3 are equivalent.

Thus, models can be hierarchically structured and verified components can be reused. Moreover, as interfaces do not disturb Petri net semantics, a flat Petri Net with the same functionality can always be generated from the hierarchically structured net.

Object oriented concepts: Components are instances of classes that are arranged within an inheritance tree. That is, classes inherit features and (token) data types from their parents. They may contain functions which can be used in transitions. Functions are written in an appropriate imperative language. With these facilities, incremental updates and evolution of models and components are supported as well as configurability and parameterization.

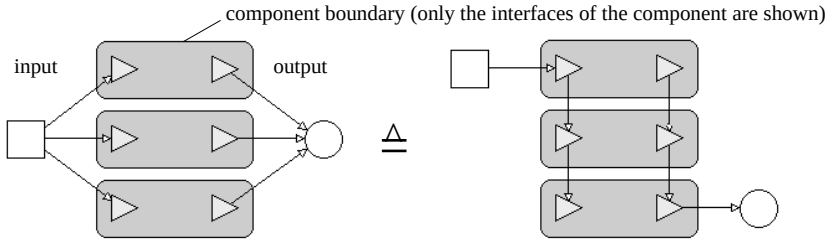

Fig. 3. Using interfaces in CodeSign.

Notion of time and simulation properties: An enabling delay interval can be associated with each transition. As a consequence, tokens carry time stamps containing information about their creation date.

Models and components can be inspected and simulated at all levels of abstraction. That is, performance evaluations and functionality checks like race conditions and timeouts can easily be performed. Finally, the simulation can be animated at runtime if desired.

Instrumentation: Observation of the system is possible without affecting the main model structure. This is realized by introducing so-called *probe* elements which collect statistical data like firing times of transitions or generation dates of certain tokens in certain places. This information can be further processed by Petri Net components or stored. Probes are invisible in the main model because they are defined in a special editor. Bindings associate probes with transition or place elements.

4 Petri Net model of a memory subsystem

In this section, the modeled memory subsystem is described which includes a memory controller, a synchronous memory chip as well as the relevant parts of the CPU and the data bus. In Fig. 4 an overview of the main system structure is given.

The CPU issues *data read* and *data write* requests. They are modeled as tokens which contain a list of values. Each request token carries information about the type of the request (read or write) and the address (an integer value) where the data can be found in memory. The sequence of requests can be arbitrarily chosen and may be extracted from address traces generated by a tool like [3].

After having received a request token from the CPU, the memory controller preprocesses the token for the SDRAM chip. The address value of the token has to be split into memory bank, row, and column addresses as these values must be transferred at different times to the RAM chip. Besides, additional tokens

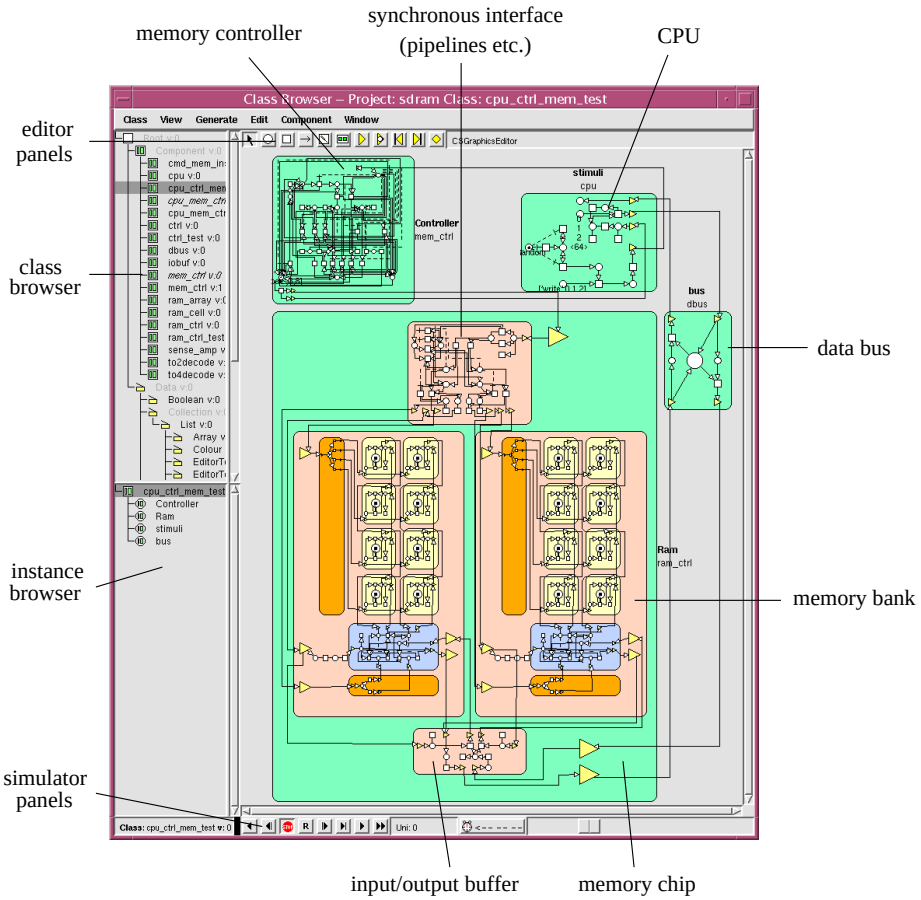

Fig. 4. Screen shot of the CodeSign [4] tool showing the model of a memory subsystem.

are generated which are necessary in order to prepare the memory, in particular tokens that initiate activate or precharge behavior of the memory cells. The memory controller is explained in more detail in section 4.2. At this stage, the request tokens may be seen as micro instructions for the control logic of the memory chip which were created from the relatively coarse grained read and write instructions of the CPU.

Finally, the memory chip obtains the modified request tokens from the memory controller. Depending on the token values, the appropriate memory data transfer is initiated. The model of the memory chip is described in the next section. At last, the requested data item (that is, a token with an integer value) is put on the data bus and either received by the CPU in case of a read request or received by the RAM in case of a write request.

The explanation of the model is organized "bottom-up" beginning with a description of the memory chip itself and ending with the subsystem overview.

4.1 SDRAM architecture

Asynchronous RAM array: As already stated in section 2.2, the core logic and the memory array of synchronous and asynchronous RAMs are identical. In Fig. 5, a conventional RAM array with its sense amplifiers, row, and column decoders can be seen (a *memory bank*). The decoders are necessary to access the contents of a single RAM cell which in general holds four to 16 bits of information. The decoders are realized using mutual exclusive guard functions within the transitions because only a single transition may be enabled per token. Token values within the decoders are interpreted as row or column addresses. The data within a memory cell is represented by a token of the type integer in the centre place as it can be seen in Fig. 2. When a row of the memory array is selected by an *activate* instruction (a token containing a row address value, a memory bank, and an operation identifier) issued by the memory controller, the information of that array row, that is, all data tokens in a row of memory cells, is transferred into the so-called *sense amplifiers*. In the row decoder, the corresponding transition fires and all memory cells of that row are activated through their *row* input interfaces. As it can be seen in Fig. 2, the transition *act* is enabled and finally the information of the memory cell (the integer token in the centre place) is transferred via the output interface *dout* to the sense amplifiers.

Sense amplifiers: The sense amplifiers are necessary since the electrical charge of dynamic memory cells is too weak to be used directly. After activating a memory row, the data tokens in the sense amplifiers can be read or updated column by column depending on the current mode of operation (read or write). The read or write state is reached when the memory controller issues *read* or *write* instruction tokens each containing an operation identifier and a column address.

Read and write operations are destructive, that is, once the information is transferred into the sense amplifiers, the charge in the memory cells is lost. Therefore, the information in the sense amplifiers must be written back into the memory cells when the data of that special row is no longer needed. This type of operation is forced by a *precharge* command of the memory controller. In this case, the integer tokens carrying the memory information are written back to the memory cells through the *din* input interface of the memory cell. The transition *pre* is enabled in case the corresponding row is decoded and the possibly modified data token is returned to the centre place of the memory cell.

All phases of a complete memory cycle may be seen in an animated simulation in CodeSign. It consists of an activation of a specific memory row, several read or write operations, and finally a precharge of the memory cells in that row. Depending on the modeled memory chip, the different operations require variable

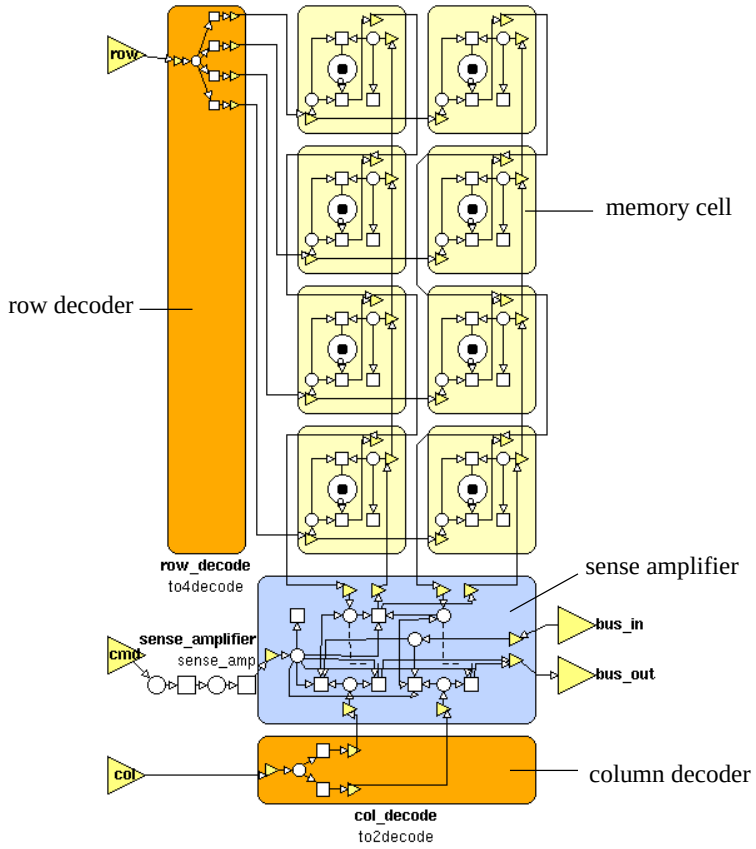

Fig. 5. A conventional asynchronous RAM array with $4 \times 2 = 8$ memory cells.

delays, e.g. in this case [6], an activation takes three clock cycles distributed through decoders and sense amplifiers.

Synchronous interface: The memory chip is completed by adding an input/ output buffer pair, registers, control logic (the synchronous interface), and by using multiple memory banks that have to share buffers. Buffers and registers are modeled as additional places for data or instruction tokens with limited capacity. The control logic implements several instruction pipelines, one for each memory bank. The pipelines are represented by successive transitions with an enabling delay of one clock cycle and places with limited capacity. That way, tokens carrying operation identifiers and addresses are delayed until the corresponding asynchronous memory array has finished the previous operation. Furthermore, the address may be incremented for burst transfers of consecutive addresses.

Thus, memory banks are able to work concurrently due to separated instruction pipelines, but only a single one is actually able to transfer data through the shared input/output buffers at a time.

In the CodeSign model, four timing parameters of the modeled synchronous RAM [6] are considered, e.g. activation and precharge delays. All modeled delays within a SDRAM component are derived from these constants. Thus, the memory chip model (a component class in CodeSign) can quickly be adapted to SDRAMs from other manufacturers or even to other memory technologies.

4.2 Memory controller

Functionality: A memory controller receives requests from the CPU at arbitrary times. They consist of the type of operation (read or write) and address information (typically 32 bit addresses for a bytewise linearly addressable memory space). Sometimes, this address information has been preprocessed by a memory management unit (MMU, see Fig. 1) that usually maps addresses to other memory regions due to memory page limits or protected memory regions (e.g reserved for memory mapped IO). The memory controller now has to take care of delays of the memory array according to the corresponding data sheet. Since there are typically several memory banks within a memory chip, the controller must prevent data tokens from collision with other tokens e.g. on the data bus. That is, the controller has to transfer instruction tokens to the memory chip at points of time which are sufficiently apart from each other because the memory chip will translate the instructions into actions on the memory cells without any timing checks.

Petri Net model: At first, the controller has to split the address information of the request tokens from the CPU into several tokens since memory chips are organized as two dimensional arrays and row and column addresses must be transmitted separately. The split address information is now transferred to the selected memory chip at different times within the memory access cycle. That is, the respective memory row must be opened with an *activate* command (token) which consists of the corresponding instruction identifier and the bank and row addresses. Then, the request token itself is transmitted to the memory chip which consists of an instruction identifier of a read or a write operation and the bank and column addresses. Read and write operations on the same memory row can be repeated several times until finally the memory row must be closed. For this, the controller transfers a *precharge* instruction token holding an identifier for this type of operation as well as bank and row addresses. This control flow dominant part of the system resembles the behavior of finite state machines. When the current state changes, corresponding actions are performed, that is, an instruction token according to the current request token of the CPU is issued. Since the explanation of the whole memory controller requires a profound knowledge of SDRAMs, it is left out in this paper. However, in Fig. 6, the entire memory controller is shown. The dotted arrows are so-called *read only* connections. In

principle, they could always be replaced by a read and a write connection because the token data is read from a place, never modified, and returned to the place. In Fig 7, an example is given in which three arbitrary request tokens from

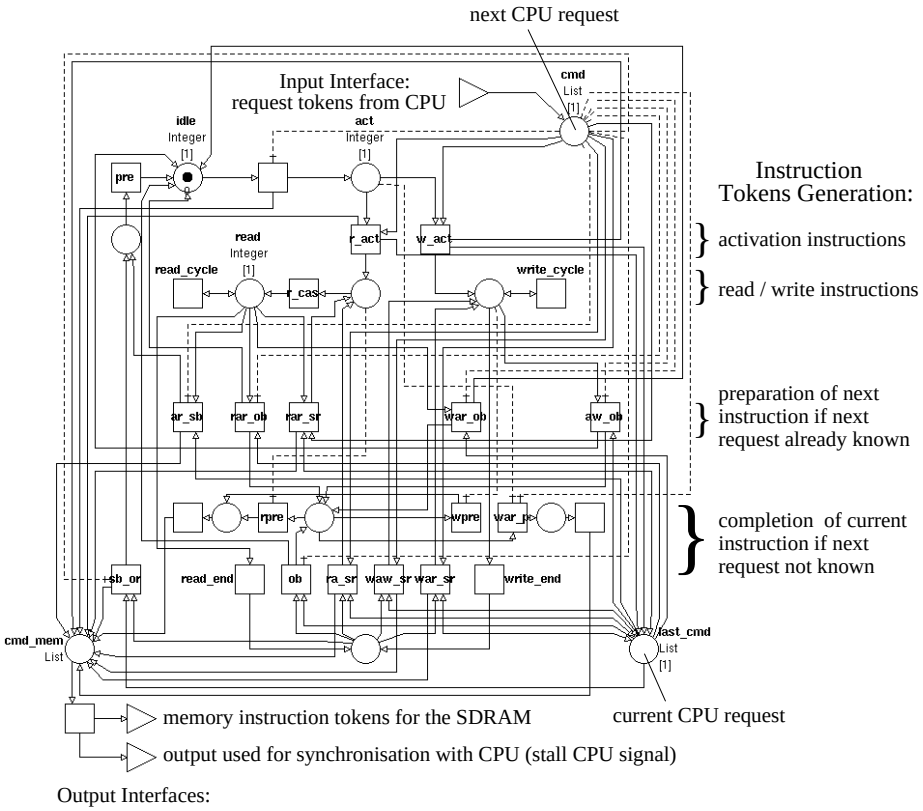

Fig. 6. Model of the memory controller.

the CPU produce six instruction tokens on the output interface of the memory controller. All tokens for a whole memory cycle containing an activation, two reads, and a precharge followed by another activation instruction can be seen. The time stamps in this example are integer values as requests and instructions are transferred at multiple times of clock cycles.

Model characteristics: Basically, the controller behaves like commonly used controllers in PCs, i.e. it schedules read and write requests of the CPU sequentially without affecting their order. However, it is capable of abbreviating the latency penalty which is usually introduced by a change of the memory row or

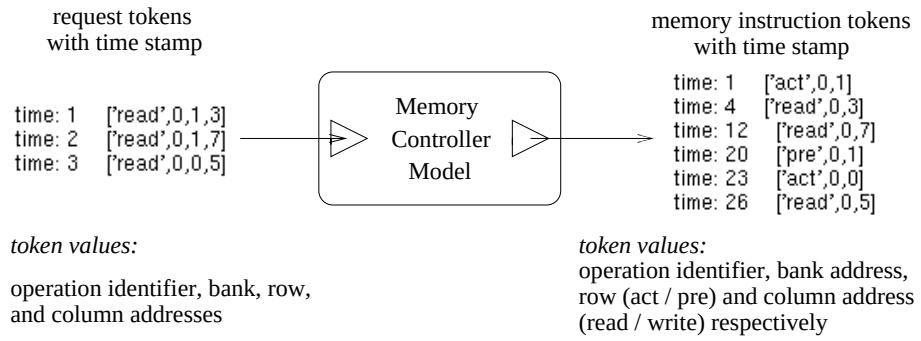

Fig. 7. Token changes within the controller.

bank in case the next request by the CPU is already known. Thus, additional distinctions of cases which consider the previous instruction more precisely and the fine adjustment of enabling delays of transitions are the core enhancements in comparison with common memory controllers.

In the CodeSign model, six timing parameters of the modeled controller are considered that depend on the memory used, that is, they are a property of the according class. Thus, the memory controller model can be quickly adapted to other memory standards or memories from other manufacturers.

The flow of instructions from the controller to the RAM is modeled using tokens of type *list* which consist of a string value and two integer values. This way, the tokens emulate commands with an operation field (read, write, precharge, activate) and two address fields (bank address, row and column address respectively). Besides, a token with an integer type is used to model the current state of the controller. The value is used as relative time reference in clock cycles during the whole memory cycle of the corresponding read or write request.

4.3 Memory subsystem

The memory subsystem is completely modeled by adding a data bus and a CPU. The data bus is shared for read and write data and therefore realized by a mutual exclusive access scheme. Using a random number generator in CodeSign, the CPU transfers read and write request tokens at arbitrary times if desired. These tokens may be extracted from previously generated address traces. The flow of read and write requests from the CPU to the memory controller is modeled using tokens of type *list* which consist of a string value and three integer values. Thus, the tokens may be seen as instructions with an operation identifier (read or write request) and one big address field which is already grouped into three main areas (bank, row, and column address) e.g. by a memory management unit. In addition, the controller may stall the CPU in order to synchronize data transfers between CPU and RAM.

4.4 Results

The complete model consists of 114 places, 140 transitions, 535 connections, and 12 probes and was created by the author with only little prior experience of Petri Nets in a period of approximately three weeks. A kind of high level Petri Nets was chosen as modeling formalism instead of Place/Transition nets. Since the main focus of the investigations were performance issues as the determination of the data bus usage and throughput as well as the duration of schedules, the possibility to associate periods with transitions in timed Petri Nets was preferred to the option to formally check properties like liveness or reachability on a Place/Transition net in all details. Furthermore, it was a great advantage to use colored tokens and guard functions because the size of the whole net became sufficiently small by shifting some complexity into transitions and tokens to facilitate a quick overview of the entire system. Finally, the similarity between the hardware system and its model was additionally supported by (hierarchically) using subnets as components.

Model analysis with CodeSign: In particular, the development and improvement of the model has been accelerated by the instrumentation and graphical simulation features of CodeSign. Several race conditions have been found quickly as well as architectural faults of the model. See Fig. 8 for an example. Race conditions within the SDRAM model may occur in case the memory controller schedules memory instructions to close in time for the memory chip. For instance, a precharge instruction may be received too early so that data may be written back to the memory cells during an update by a write instruction. Accordingly, data within memory cells may be destroyed, that is, data tokens are not of the expected value. Data collisions on the bus may happen in case the CPU and the memory controller (and consequently the memory chip) are not synchronized. Collisions usually hold the entire system model due to shared resources. Frequently, wrong schedules of the memory controller such as missed out instructions or too dense instruction issues are the main cause for erroneous conditions.

Model development and verification: The model of the memory chip was developed first. It was checked with the help of fixed, previously composed memory instruction sequences for which the default behavior of the memory chip was already known. After having adjusted all transition enabling delays according to the corresponding data sheet [6] and corrected some minor faults in the architecture model such as too short an instruction pipeline leading to a deadlock of the whole memory chip, the memory controller and its surrounding components (bus, CPU) were added to the model. Again, previously composed sequences of instructions were used, in this case sequences of read and write request tokens, in order to check the model. False schedules of memory instructions generated by the controller were quickly found and valid ones were verified with the help of the SDRAM model. Enabling delays were minimized in order to be able to

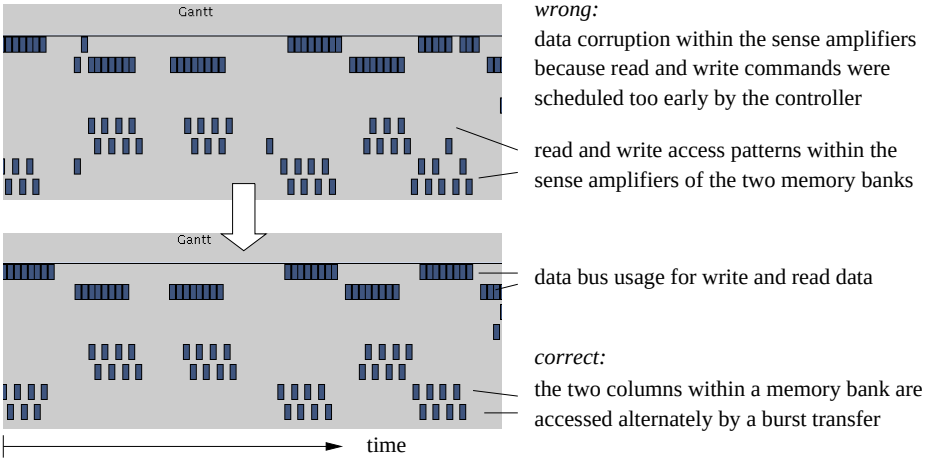

Fig. 8. Instrumentation: The firing times of certain transitions were collected and visualized by CodeSign.

schedule the next memory instruction as early as possible but without causing any deadlocks within the memory chip.

Performance comparison: The shortened schedules of memory instructions issued by the memory controller in comparison with conventional controllers have been realized by about ten additional transitions and several places as well as the fine tuning of enabling delays of transitions within the controller. In Tab. 1 some results for a four times burst transfer are shown. These results depend on the memory access type used. That is, four column entries in a memory array row are transferred consecutively. This is the typical mode of operation in common PCs. The access types shown are the most frequently found access schemes in address traces. For instance, a read operation after a read operation on another memory row of the same memory bank is called a *read row miss* because the same row cannot be used for both operations. Otherwise, the second operation would be called a *read row hit*. The delays in clock cycles are given in the table for each column entry. An expression like *7-1-1-1* means that the first entry needs seven clock cycles until it is delivered to the CPU in case of a read request or saved into the sense amplifiers in case of a write request. The next three entries only need one cycle each due to pipelined transfers. In the last column of Tab. 1, the potential performance improvements are shown for the corresponding access type. It can be seen that the enhanced controller modeled in CodeSign is almost always faster than the quoted common controllers. The performance gain has been mainly achieved by optimized, compressed command schedules of the controller in case the next read or write requirement of the CPU is already

known. In particular, the property of preparing a data transfer of one memory bank and transferring data with another bank concurrently has been exploited.

SDRAM[1] access type		Memory Controller (delays in *clock cycles*) Intel[7]	SiS[14]	AMD[1]	CodeSign Model max.[3]	min.[4]	speed increase max.	min.
read	row hit	7-1-1-1	7-1-1-1	6-1-1-1	7-1-1-1	1-1-1-1	60%	0%
	row start	9-1-1-1	10-1-1-1	9-1-1-1	7-1-1-1	1-1-1-1	69%	17%
	row miss	12-1-1-1	11-1-1-1	11-1-1-1	10-1-1-1	7-1-1-1	33%	7%
	b-b[2] burst	2-1-1-1	3-1-1-1	3-1-1-1	1-1-1-1		33%	20%
write	row hit	3-1-1-1	2-1-1-1	not	1-1-1-1		33%	20%
	row start	6-1-1-1	6-1-1-1	specified	4-1-1-1	1-1-1-1	56%	22%
	row miss	9-1-1-1	9-1-1-1		7-1-1-1	4-1-1-1	42%	17%
2. level cache[5]								
	read/write	3-1-1-1	3-1-1-1	3-1-1-1				
	b-b[2] burst	1-1-1-1	1-1-1-1	1-1-1-1				

1) *66 MHz clock, t_{CAS} =3 cycles, see [6] for details.*
2) *back to back burst read, row hit*
3) *Maximum delay: The next request by the CPU is not known in advance.*
4) *Minimum delay: The next request is known while processing of the current one.*
5) *All quoted controllers also control the second level cache of the computer system.*

Table 1. Speed comparison between popular controllers and the CodeSign model.

That is, with the help of the quickly developed model, promising performance increases were deduced. Further investigations will focus on examinations of address traces of real applications in order to quantify how often the different address types are actually used. Accordingly, virtual performance improvements will be determined which will depend on the application analyzed and the properties of the memory used in the computer system.

5 Conclusion

In this paper, a complex model of a memory subsystem of a computer system was shown. With its help remarkable speed increases with small architecture enhancements were realized in comparison with common solutions. The improvement was derived by using parallel memory banks in memory chips concurrently. The complete model with optimizations was created within a short period with the help of the CodeSign Petri Net modeling environment which is based on colored time Petri nets and which fulfills all necessary requirements in order to represent control and data flow as well as structure properties of real systems in an intuitive way.

The case study showed that the functionality of memory controllers should be improved primarily as RAM array delays remain the same and only memory interface speed increases. Furthermore, an optimized controller may be considered as an alternative to second level caches in power dissipation or cost critical systems.

The model is easily adaptable to new standards and technologies using its configurability and the inheritance feature of the CodeSign environment.

Acknowledgement

The author would like to thank R. Esser, J. Janneck, M. Naedele, and L. Thiele for valuable comments and discussions.

References

1. Advanced Micro Devices Inc. *AMD-640 System Controller*, June 1997.
2. G. Booch. *Object-oriented analysis and design, with applications.* Benjamin/Cummings, 2nd edition, 1994.
3. Bob Cmelik and David Keppel. Shade: A fast instruction-set simulator for execution profiling. *Proceedings of the 1994 ACM SIGMETRICS Conference on the Measurement and Modeling of Computer Systems*, pages 128,137, May 1994.
4. Robert Esser. *An Object Oriented Petri Net Approach to Embedded System Design.* PhD thesis, ETH Zurich, 1996.
5. David Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8:231–274, 1987.
6. IBM Corp. *64Mb Synchronous DRAM, IBM0364164C*, March 1997.
7. Intel Corp. *430TX PCISET: 82439TX System Controller (MTXC)*, February 1997.
8. Kurt Jensen. *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use*, volume 1: Basic Concepts of *EATCS Monographs in Computer Science.* Springer-Verlag, 1992.
9. Yasunao Katayama. Trends in semiconductor memories. *IEEE Micro*, 17(6):10 – 17, November 1997.
10. Tadao Murata. Petri nets: Properties, analysis, and applications. *Proceedings of the IEEE*, 77(4):541–580, April 1989.
11. Betty Prince. *High Performance Memories.* John Wiley & Sons Ltd., 1996.
12. J. Rumbaugh, M. Blaha, W. Premerlani, and F. Eddy. *Object-oriented modeling and design.* Prentice Hall, Englewood Cliffs, New Jersey, USA, 1991.
13. Bran Selic, Garth Gullekson, and Paul Ward. *Real-time object-oriented modeling.* John Wiley and Sons, 1994.
14. Silicon Integrated Systems Corp. *SiS5597 Pentium PCI/ISA Chipset*, April 1997.
15. SLDRAM Inc. *4M x 18 SLDRAM, pipelined, eight bank, 2.5V operation*, February 1998.
16. Frederick A. Ware. *Direct RDRAM, 64/72-Mbit (256kx16/18x16d).* Rambus Inc., 3465 Latham Street, Mountain View, California USA, October 1997.