SDRAM-based packet buffer model for high speed switches

A. Rasmussen, S. Ruepp, M. Berger, H. Wessing, H. Yu
DTU Department of Photonics, Technical University of Denmark
E-mail: {anras,srru,msbe,hewe,haoyu}@fotonik.dtu.dk

Abstract
This article investigates the how the performance of SDRAM based packet buffering systems for high performance switches can be simulated using OPNET. In order to include the access pattern dependent performance of SDRAM modules in simulations, a custom SDRAM model is implemented in OPNET Modeller based on the specifications of a real-life DDR3-SDRAM chip. Based on this model the performance of different schemes for optimizing the performance of such a packet buffer can be evaluated. The purpose of this study is to find efficient schemes for memory mapping of the packet queues and I/O traffic shaping to provide the best performance in terms of latency and throughput.

Introduction
The rapid growth of internet traffic is setting higher and higher requirements for the throughput of the network nodes, which route the packets through the internet. As a consequence, significant strain is also put on the internal buffering systems, in terms of throughput, latency and capacity. Current memory technology such as static RAM (SRAM) and dynamic RAM (DRAM) cannot simultaneously satisfy the requirement for high throughput at low latency while providing the high memory capacity required to avoid packet loss under uneven traffic distribution. SRAM is fast enough, providing low access latency and high throughput, but current technology limits the practical size of these buffers to a few megabytes, which is not sufficient for high-end systems. Another drawback is the relatively high power requirements of such devices. DRAM chips on the other hand can have capacities in the gigabyte region with much lower power requirements, but have significant issues with access latency, especially for random access patterns. This is becoming a problem as modern line cards are moving from 10Gbps operation towards 100Gbps operation to accommodate new standards such as OTN4 and 100G Ethernet [6][7]. As a consequence a lot of research has gone into more advanced buffering architectures, which combines the two memory types to exploit their individual advantages [2]-[4].

In this paper, we look into how OPNET can be used to simulate these buffering schemes, either by themselves or as part of a larger switch or router model. We put special focus on modelling the DRAM, which is a common component for all DRAM/SRAM based buffering systems. Unlike SRAM buffers, the performance characteristics of DRAM cannot be easily modelled by the build in OPNET functions such as lists and simple queues because the read/write latency and throughput is highly dependent on the access pattern to the memory. Hence we need a custom made process model, which take this into account.

The rest of the article is divided into several subsections: First, we give a brief overview of the basic architecture and operation of the common DRAM type called Synchronous DRAM (SDRAM). Secondly, we will give an introduction to the basic idea behind hybrid SRAM/DRAM buffering architectures. Finally, we describe how this system can be modelled using OPNET Modeller with special emphasis on modelling the DRAM and show some results from our own model, based on a commercially available DDR3-SDRAM chip from Micron [1].

SDRAM Overview
Synchronous DRAM is a technology which is commonly found in personal computers around the world, usually just referred to as the computer's "RAM" or "main memory". Common types of SDRAM chips are DDR, DDR2 and DDR3 SDRAM, all of which have the same basic structure but operate at different speeds. In order to increase the bandwidth to the memory, modern computers use so called DIMM modules containing multiple SDRAM chips, which are accessed in parallel.

The model presented in this paper is based on the MT41J256M8 DDR3-SDRAM chip from Micron, but is easily modified represent other DRAM memory modules. The chip consists of eight memory banks, which can operate in parallel, but share the same control logic and I/O data path. The performance of the chip is highly dependent on the access pattern to the memory and the proper scheduling of requests from the memory controller, which interfaces with the SDRAM chip. Under optimal conditions full throughput can be maintained to and from the memory.

The individual memory banks are divided into rows and columns as depicted in Figure 1. The distinction between rows and columns is very important in SDRAMs, since jumping between rows and jumping between columns have very different delay properties. The difference between these two operations will be explained in the following section.

	Column 0	----------	Column M
Row 0			0/M
-			
Row N			N/M

Figure 1 - Bank organization

Bank functionality
Each bank has an internal state machine as depicted in Figure 2. For simplicity, the power saving and self refresh features have been left out.

Row Activation
The banks are initially in the idle state waiting for instructions. Before a row of memory in a certain bank can be accessed, this row first needs to be activated. After the activation delay (tRCD), the bank is active and the selected row is ready to be manipulated by read or write commands.

Read and write operations
Once the correct row has been activated, the actual read or write operations can be performed on the columns of the row, which in this case holds a byte each. The operations are performed in

bursts of 8 bytes known as the "burst size", which can be concatenated to form a continuous stream of data to or from the same row, given that the spacing between each request is small enough, and given that the direction (read/write) is the same. How closely the requests must be spaced depends on the individual design of the SDRAM chip. When changing direction, there is also a delay penalty, which must be taken into account.

Precharging
Once the required operations on the active row have been completed, the row has to be deactivated or *precharged* before another row can be activated. After the precharge delay (tRP), the bank enters the *idle* state and is ready to activate a new row. There is no requirement to perform precharging immediately after the last read/write operation, but the precharge delay will add to the total activation delay when accessing a row, if a previous row has been left active.

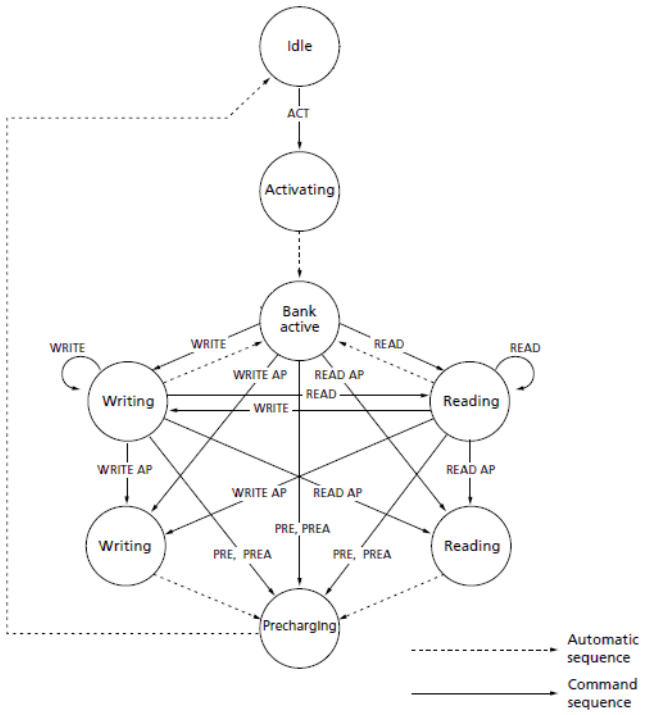

Figure 2 - Simplified SDRAM Bank State Diagram [1]

SRAM/DRAM hybrid system

The basic idea behind the SRAM/DRAM buffering system depicted in Figure 3 is somewhat similar to the concept of caches commonly used in computer systems [2] [8]. When packets arrive, they are first stored in a small SRAM based FIFO queue (Tail FIFO). Once a sufficient amount of data has been collected, the content of the FIFO is moved to the DRAM in one consecutive write burst, thus freeing up the SRAM memory. In a similar fashion, the packets are moved in read bursts from the DRAM to the Head FIFO before departure. The purpose of this three step FIFO system is to keep the bulk of the buffered packets in the large low-cost DRAM memory, while maintaining just enough data in the Head FIFO to compensate for the access latency of the DRAM read bursts, thus reducing the amount of SRAM required for congestion handling.

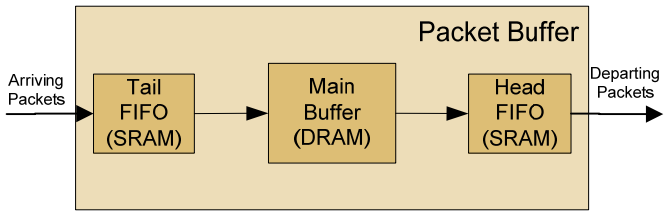

Figure 3 - SRAM/DRAM Packet Buffer

SDRAM OPNET model

The SRAM/DRAM packet buffer can be modeled in OPNET as depicted in Figure 4. To facilitate integration with existing switch and router models, such as the depicted line card, the head- and tail FIFOs are implemented as generic FIFO queues, which can be accessed from the top level process model as well as the underlying child processes. From here, the Packet Buffer Model (PBM) will control the flow of packets from the tail FIFO to the head FIFO through the SDRAM. Exactly how the PBM performs this task and how well it performs will depend on the modeled buffer design which is to be investigated.

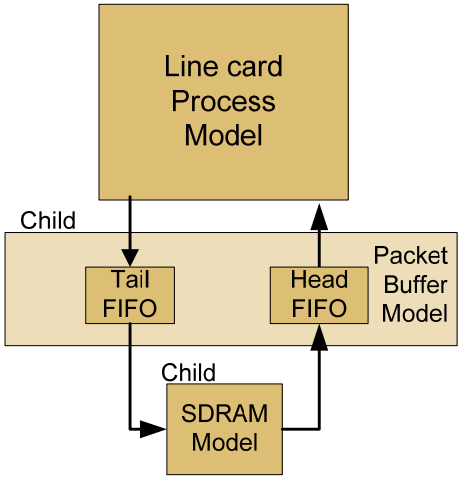

Figure 4 – Line card Model

Communication with the SDRAM is done using a custom packet format, as depicted in Figure 5. These packets contain a command field specifying the type of operation (e.g. activate, precharge, read, write) as well as the bank, row and column number and an optional payload. The SDRAM model will then perform the operation with a latency corresponding to that of a physical SDRAM device and, in case of a read or write command, reply back to the parent process. The SDRAM model itself does not need to hold actual data packets (these could be stored elsewhere in the PBM), as long as the PBM generates the proper command sequence to the SDRAM and waits for a read/write reply to return before moving the actual packet.

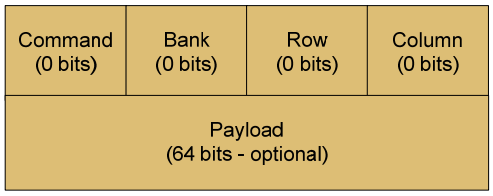

Figure 5 - SDRAM packet format

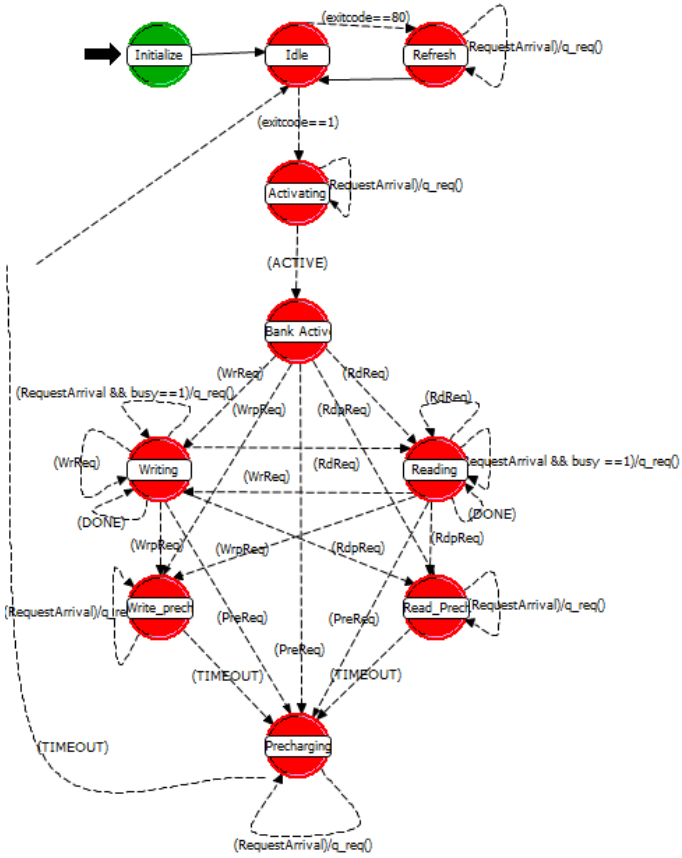

Figure 6 - Process Model of Simplified SDRAM Bank State Machine

The SDRAM model is implemented as a root process called the "dispatcher" with eight child processes – one for each bank. The dispatcher forwards the requests to the relevant banks based on the bank number found in the request packet header. The shared I/O data bus, which limits the active read/write operation to one bank at the time is currently not implemented, but can be handled by either the dispatcher or its parent process.

The state diagram for the bank processes model is depicted in Figure 6. Aside from some extra logic for queuing arriving requests, which cannot be handled right away it is virtually identical to the state diagram from the SDRAM documentation (Figure 2). The process model stays in the idle state after initialization. Once it receives a request packet containing an "activate" command, it activates the specified row by moving on to the "Activating" state. It stays there for a number of nanoseconds determined by the specified activation delay (tRCD) before moving to the "Bank Active" state. The process model is now ready to manipulate the activated row according to the subsequent requests received from the parent process by moving between the five other states. When the required operations have been performed, the row can be closed again by issuing a "precharge" request (PreReq). This puts the process model in the "Precharge" state, which automatically moves back to the "Idle" state after the precharge delay (tRP) has expired. Precharging can also be performed by issuing a Write-Precharge (WrpReq) or a Read-Precharge (RdpReq) request, which is simply a write or read command, which automatically precharges the bank once the write/read operation is completed.

When ever the system changes its state (e.g. from Reading to Writing), it will delay further action until the operation is completed. This delay is implemented as simple self-interrupts and is completely dependent on the current and the next state of the system. Hence, moving from "Bank Active" to "Writing" has a different latency than moving from "Reading" to "Writing" as is also the case for the physical SDRAM chip. The different delays when moving between states is listed in Table 2 in terms of the chip dependent delay constants listed in Table 1.

Shorthand	Name	Description
tRCD	Activation delay	Time it takes to activate a row from idle
tRP	Precharge delay	Time it takes to deactivate a row and return to idle
CL	CAS Latency	Time from read/write request until first byte is ready on the input/output
tTurnRD	Turnaround delay (rd-to-write)	Delay penalty when changing I/O direction
tTurnWR	Turnaround delay (write-to-rd)	Delay penalty when changing I/O direction
tRDWR	Read/Write delay	Time it takes to actively read/write 8B of data
tRTP	Read-to-Precharge delay	Time from a read command has been issued to a precharge command can be issued.
tWR	Write recovery delay	Delay from last byte has been written until precharging can start

Table 1 - List of delay types

From	To	Delay
Idle	Bank Active	tRCD
Bank Active	Idle	tRP
Bank Active	Read/Write	CL
Read	Read/ReadPre	tRDWR
Write	Write/WritePre	tRDWR
Read	Write	tTurnRD
Write	Read	tTurnWR
Write/WritePre	Idle	tRDWR+tWR+tRP
Read/ReadPre	Idle	tRTP+tRP

Table 2 - Path delays

Results and Verification

In order to verify and validate the SDRAM model, a simple client model has been developed, which consists of a request source and a reply sink as depicted in Figure 7. The request source transmits a repetitive pattern of requests to the memory module (e.g. activate, write, read, precharge). The corresponding replies from the memory is then received by the reply sink and validated based on the bank/row/column address and the payload, which is a function of the address fields.

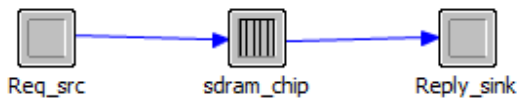

Figure 7 - Verification node model

By changing the traffic pattern generated by the request source, the functionality of the model i.e. whether the system jumps correctly between states and performs the correct read/write operations is easily verified. Furthermore, the time accuracy of the model can be verified by comparing the measured maximum throughput with the expected maximum throughput for various traffic patterns. Since the sharing of the I/O data bus used for reading and writing is not yet implemented, the verification and results described below are based on accessing a single memory bank.

Verification

Figure 8, **Figure 9** and **Figure 10** shows the throughput of one memory bank for two different access patterns emulating random access (red line) and bursty access (blue line) respectively as the request rate (packets/s) increases. As expected, the bursty access pattern yields a significantly higher maximum throughput compared to the random access pattern. The reason becomes clear when we look at the request patterns and their respective delay listed in **Table 3** and **Table 4**.

Request	Delay	Delay (additive)
Activate	tRCD=15 ns	15ns
2x Write	CL(5ns) + 2x tRDWR(4ns) = 13ns	28ns
Read	tTurnWR(4ns) + tRDWR(4ns) = 8ns	36ns
Read-Pre	tRTP(8ns) + tRP (10ns) = 18 ns	54ns
Activate	tRCD = 15 ns	69ns
Write-Pre	CL(5ns) + tRDWR(4ns) + tWR(8ns) + tRP(10ns) = 27ns	96ns
Activate	tRCD = 15ns	111ns
Read	CL(5ns) + tRDWR(4ns) = 9ns	120ns
Precharge	tRTP(8) + tRP (10ns) - tRDWR(4ns) = 14ns	**134ns**

Table 3 - Random Access Pattern

Request	Delay	Delay (additive)
Activate	tRCD=15 ns	15ns
49x Write	1x CL(5ns) + 49x tRDWR(4ns) = 201ns	216ns
49x Read	tTurnWR(4ns) + 49x tRDWR(4ns) = 196ns	416ns
Read-Pre	tRTP(8) - tRDWR(4ns) + tRP (10ns) = 14ns	**430ns**

Table 4 - Bursty Access Pattern

Based on the cycle time of the access patterns and the amount of read (or write) operations performed in each cycle, we can calculate the theoretical maximum throughput of the two patterns:

Random Access Pattern

Cycle time:		=	134ns
Cycles per second:	1s/134ns	=	7,462,687
Read bits/cycle:	3x64bit	=	192 bits
Throughput (bps):	192bits x 7,462,687	=	**1,433 Mbps**

Bursty Access Pattern

Cycle time:		=	430ns
Cycles per second:	1s/430ns	=	2,325,581
Read bits/cycle:	50x64bit	=	192 bits
Throughput (bps):	3.200bits x 2,325,581	=	**7,442 Mbps**

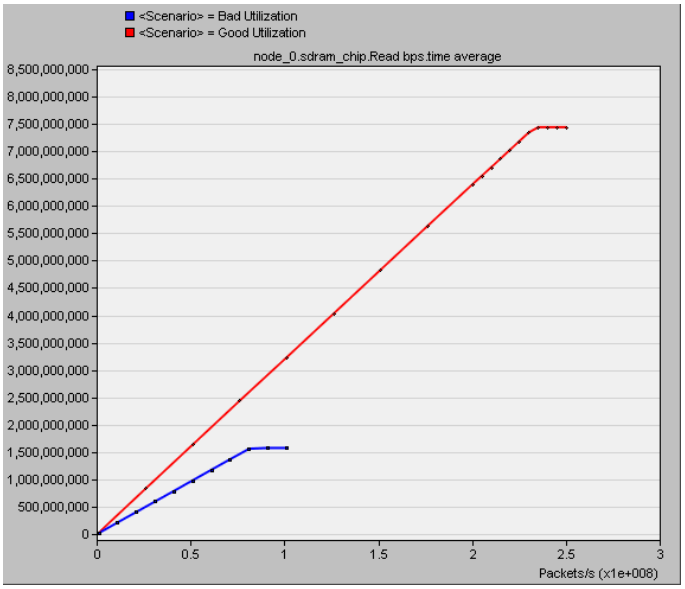

Figure 8 - Throughput vs. Request rate with two different access patterns

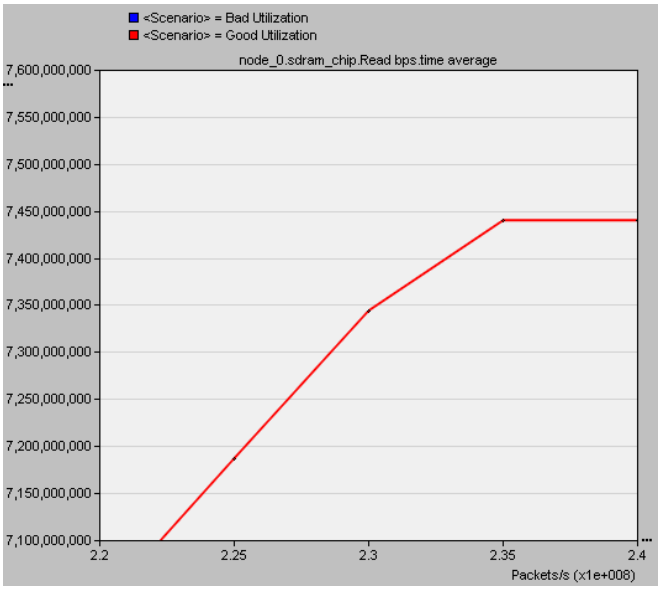

Figure 9 - Zoom of "Bursty Access Pattern"

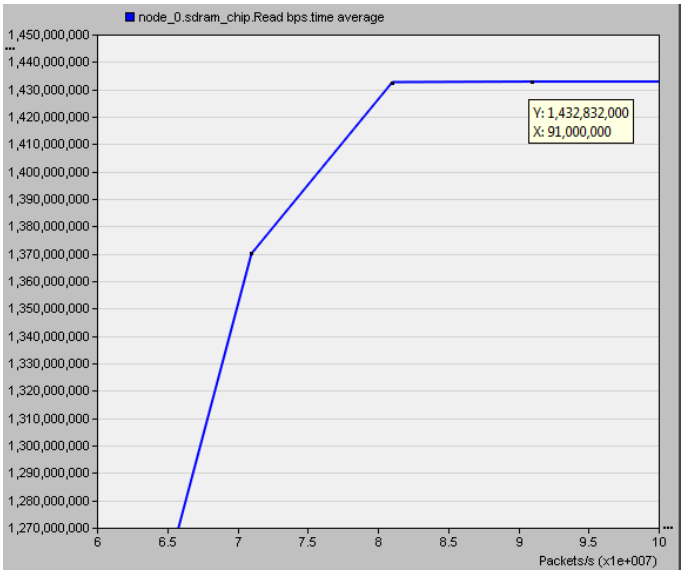

Figure 10 - Zoom of " Random Access Pattern"

By visual inspection of the throughput statistics on the previous page, it can be verified that the model accurately reproduces the delay found in a physical memory bank, which limits the maximum obtainable throughput.

Conclusion

In this paper, we have shown how a SRAM/DRAM buffering system can be modelled using OPNET with special emphasis on modelling the access pattern dependent performance of the DRAM. To this end, we have successfully implemented and verified a DRAM model based on a DDR3-SDRAM chip from Micron, which is easily modified to simulate other DRAM memory chips if needed. The emphasis has been on accurately reproducing the access pattern dependent delay inside the memory chip and hence, the penalty in terms of lower throughput and higher latency which is encountered for non-optimal access patterns. The model is designed to emulate a memory module with eight memory banks, but since the sharing of the I/O data bus between the banks is yet to be accurately implemented, the verification and results presented in this report is based on access to a single memory bank. The performance profile for this bank has been tested using two different access patterns and it has been verified, that the throughput the matches the expected values derived by manual calculation from the datasheet. The next step is to include the I/O sharing and perform the same verification using multiple banks. Once the complete model is verified, the SDRAM process model can be used as part of larger packet buffer models aimed at measuring the performance of different SDRAM-based buffering schemes for network nodes operating at 100Gbps and beyond.

Acknowledgment

This work has been partially supported by the Danish Advanced Technology Foundation (Højteknologifonden) through the research project ”The Road to 100 Gigabit Ethernet”.

References

[1] Micron MT41J256M8 DDR3 SDRAM datasheet: http://download.micron.com/pdf/datasheets/dram/ddr3/2Gb_DDR3_SDRAM.pdf, 2006.

[2] F. Wang, M. Hamdi, “Memory Subsystems In High End Routers”, IEEE Micro, vol. 29, pp. 52-63, 2009.

[3] S. Iyer, R. Kompella, and N. McKeown,” Designing Packet Buffers for Router Line Cards IEEE/ACM Transactions on Networking, Volume 16, Issue 3, pp. 705-717, 2008

[4] J. Garcìa-Vidal, Llorenç¸ Cerdà , J. Corbal, M. Valero, ‘‘A DRAM/SRAM Memory Scheme for Fast Packet Buffers,’’ IEEE Trans. Computers, vol. 55, no. 5, pp. 588-602, May 2006.

[5] C. Hermsmeyer, H. Song, R. Schlenk, R. Gemelli,,S. Bunse, ”Towards 100G Packet Processing: Chalenges and Technologies”, Bell Labs Technical Journal vol. 14, issue. 2, pp. 57–80, 2009.

[6] LAN/MAN Standards Committee of the IEEE Computer Society, “IEEE p802.3ba D2.1 - Amendment: Media Access Control parameters, physical layers and management parameters for 40 gb/s and 100 gb/s operation,”, 2009.

[7] ITU-T Recommendation G.709/Y.1331 (12/09), Interfaces for the optical transport network (OTN), ITU-T, Geneva, December, 2009.

[8] D. A. Patterson, J. L. Hennessy, “Computer Organization and Design – The Hardware/Software Interface”, Third Edition, ISBN: 1-55860-604-1, 2005.