UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE ENGENHARIA DE COMPUTAÇÃO

ÉDER FERREIRA ZULIAN

Explorations on Approximate DRAM

Work presented in partial fulfillment
of the requirements for the degree of
Computer Engeneering

Advisor: Prof. Dr. Marcelo de Oliveira Johann

Porto Alegre
July 2016

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
Reitor: Prof. Carlos Alexandre Netto
Vice-Reitor: Prof. Rui Vicente Oppermann
Pró-Reitor de Graduação: Prof. Sérgio Roberto Kieling Franco
Diretor do Instituto de Informática: Prof. Luis da Cunha Lamb
Coordenador do Curso de Engenharia de Computação: Prof. Raul Fernando Weber
Bibliotecária-chefe do Instituto de Informática: Beatriz Regina Bastos Haro

“You never get free lunch!”

ACKNOWLEDGEMENTS

I would like to thank Prof. Dr.-Ing. Norbert Wehn and all the staff of the Microelectronic Systems Design Research Group from the TU Kaiserslautern for the exceptional work environment and for the hospitality during my stay in Germany.

I would like to express my gratitude to Dipl.-Ing. Matthias Jung for guidance and lots of useful insights provided during this work and to Dr.-Ing. Christian Weis for sharing his vast knowledge.

I would also like to thank Prof. Dr. Marcelo de Oliveira Johann for being my advisor on this work in Brazil and Prof. Dr. Taisy Silva Weber for coordinating the Brazilian side of the cooperation between UFRGS and TU Kaiserslautern.

Finally, I thank my wife Sissi for her love and for inspiring me.

ABSTRACT

DRAMs have a considerable impact on performance and contribute significantly to the total power consumption in computer systems. They are based on dynamic memory cells that require periodic refreshes in order to ensure data integrity. Recent research shows the increasingly impact of DRAM refresh on power consumption for future high density DRAM devices (LIU et al., 2012),(BHATI et al., 2015).

The employment of new DRAM refresh techniques with focus on energy savings and enhanced memory throughput is an active research topic in the DRAM community. Many strategies renounce the reliability provided by a worst-case driven refresh interval present on the standards. They consciously break some rules in order to achieve certain gains.

Evidently, increases on the refresh interval or even the total suppression of refreshes may cause total or partial corruption of the stored data within its life time. Therefore, approximate DRAM storage is specially interesting for error resilient applications. Fortunately, error resilience is a characteristic that can be found on a broad range of applications such as signal processing, image, audio, and video processing, graphics, wireless communications, web search, and data analytics (CHIPPA et al., 2013).

The main purpose of this work is to evaluate the impact of disabling the DRAM refresh in terms of energy savings and occurrence of retention errors. Furthermore, this work presents some possible applications that could derive benefit from the suppression of DRAM refreshes.

Keywords: DRAM, refresh, retention time, accuracy, approximate computing, energy savings.

Explorando Armazenamento Aproximado em DRAMs

RESUMO

DRAMs impactam de maneira considerável na performance e contribuem significativamente no consumo energético e potência dissipada em sistemas computacionais. DRAMs têm como base células dinâmicas de armazenamento que demandam atualização periódica de seu conteúdo para que se mantenha íntegro. Pesquisas recentes indicam um crescente impacto na potência consumida devido à atualização periódica das células de memória para futuros dispositivos de alta densidade de células. (LIU et al., 2012),(BHATI et al., 2015).

Pesquisadores da área procuram novas técnicas para a atualização das células de memória visando menor consumo energético e aumento da taxa de transferência de dados da memória. Muitas estratégias abrem mão das garantias providas pelo estrito seguimento às especificações que forçosamente devem contemplar o pior caso.

Evidentemente, atualizar as células de memória menos frequentemente ou simplesmente deixar de atualizá-las pode causar perda total ou parcial dos dados armazenados caso o tempo de vida dos dados seja superior ao tempo de retenção das células de memória que o armazenam. Sendo assim, a utilização de armazenamento aproximado em DRAMs é especialmente interessante para aplicações que apresentem tolerância à erros. Felizmente, essa é uma característica presente em diversas aplicações tais como processamento de sinais, processamento de imagem, áudio e vídeo, comunicação sem fio, buscas em bancos de dados na rede mundial de computadores e análise de enormes quantidades de dados (CHIPPA et al., 2013).

O principal propósito deste trabalho é avaliar o impacto da supressão da atualização periódica da memória em termos de economia energética e ocorrência de erros de retenção. Além disso, são apresentadas sugestões de aplicações que podem colher algum benefício do uso desta técnica.

Palavras-chave: DRAM, atualização periódica da memória, tempo de retenção, precisão, computação aproximada, economia de energia.

LIST OF ABBREVIATIONS AND ACRONYMS

1T1C	One-Transistor One-Capacitor
AR	Auto Refresh
AT	Approximately Timed
BIOS	Basic Input/Output System
DDR	Double Data Rate
DIMM	Dual In-line Memory Module
DMC	DRAM memory controller
DRAM	Dynamic Random Access Memory
ECC	Error-correcting Code
GR	electron-hole generation-recombination center
HMC	Hybrid Memory Cube
I^2C	Inter-Integrated Circuit
JEDEC	Joint Electron Device Engineering Council
PCB	Printed Circuit Board
SMBus	System Management Bus
SoC	System-on-Chip
SPD	Serial Presence Detect
TLM	Transaction Level Modelling
tRFC	refresh cycle time
UDP	User Datagram Protocol
VRT	Variable Retention Time

LIST OF FIGURES

Figure 2.1 1T1C DRAM Cell Structure 14
Figure 2.2 Differential Sense Amplifiers (Open Bitline Structure) 15
Figure 2.3 Common DRAM Organization 16
Figure 2.4 DRAM DIMM 17
Figure 2.5 DRAM Memory Controller 17
Figure 2.6 Leakage Current and Refresh Interval vs. Temperature. 20
Figure 2.7 Wide I/O 3D memory stack 22
Figure 2.8 A representation of the HMC scheme in respect to memory accesses 22
Figure 2.9 Power Breakdown Samsung Galaxy S III (suspended state, 3G enabled) 23
Figure 2.10 Power Breakdown Google's Datacenter (2007) 24
Figure 2.11 Power Breakdown Modern Datacenter (2012) 24
Figure 2.12 Power Breakdown Energy-Efficient Seismic Simulation Platform 25
Figure 2.13 Power Breakdown eBrain 25

Figure 3.1 DRAMSys Base Architecture 27

Figure 4.1 DRAMSys and 3D-ICE Integration 29
Figure 4.2 Graphical representation of some aspects of the 3D-ICE project 30
Figure 4.3 Retention Errors for different Refresh Intervals and Temperatures 31
Figure 4.4 Retention Errors vs. Temperature (350 ms without refresh) 31
Figure 4.5 Image Rotation in FPGA 34
Figure 4.6 Image Rotation Energy and Time Results 35

LIST OF TABLES

Table 2.1 Impact of Refresh on DRAM Devices 19

Table 4.1 Retention Errors (350 ms without refresh).. 32
Table 4.2 Energy and Power Comparison (350 ms without refresh) 32

CONTENTS

1 INTRODUCTION.......................................11
1.1 Context and Motivation.......................................11
1.2 Relevance and Contribution 11
2 DRAM.......................................13
2.1 Basic Circuits.......................................13
2.2 Organization.......................................16
2.3 DRAM Peculiarities.......................................18
2.4 Refresh 19
2.5 Error Correction Mechanisms.......................................20
2.6 Technological Trends 21
2.6.1 Wide I/O.......................................21
2.6.2 Hybrid Memory Cube.......................................22
2.7 Impact on Power 23
3 DESIGN SPACE EXPLORATION TOOLS 26
3.1 SystemC and TLM.......................................26
3.2 DRAMSys 27
3.2.1 Error Model.......................................27
3.3 3D-ICE 28
3.4 DRAMPower 28
4 IMPLEMENTATION AND RESULTS 29
4.1 Simulation of WIDE I/O Memory.......................................30
4.2 Applicability of Approximate DRAM Storage.......................................32
4.3 Hardware Experiment with DDR3-SDRAM and FPGA 34
5 CONCLUSION 37
REFERENCES.......................................38
APPENDICES.......................................41
APPENDIXA C++ FILES.......................................42
APPENDIXB 3D-ICE PROJECT FILES.......................................48
B.1 Stack File.......................................48
B.2 Floorplan Files.......................................50
APPENDIXC PUBLICATION 53
C.1 Omitting Refresh: A Case Study for Commodity and Wide I/O DRAMs........53
APPENDIXD PARTIAL REPORT.......................................61

1 INTRODUCTION

1.1 Context and Motivation

Computer systems permeate today's life. Memory is an essential part of computers and is present in cheap electronic gadgets, mobile computers (phones, tablets, wearables), high availability data communication equipments and data servers to cite a few. DRAM is the primary memory of most computer systems since many decades. The simple hardware structure, the relative high access speed, the relative lower costs of production and the scalability provided by this memory technology corroborate this scenario.

Despite technological advances, there is a growing disparity between processor and memory speeds. This issue was formally announced two decades ago in a research paper as the *memory wall* (WULF; MCKEE, 1995). Furthermore, the process scaling of current semiconductors is approaching its limits. Hence, new solutions are necessary in order to deliver increased storage capacity and bandwidth. Stacked three-dimensional structures, in which each layer consists in a two-dimensional die, are being explored to overcome the current demands. However, this new approach comes with new challenges related to power density and removal of heat (WEIS et al., 2015).

Along with storage capacity and bandwidth, power and energy requirements are major concerns in current computer system designs (CAMERON; GE; FENG, 2005), (BENINI; BOGLIOLO; MICHELI, 2000). Trade-offs involving power and performance are recurrent among system-level architects. *Approximate computing* breaks with the all-or-nothing correctness philosophy adopted so far by computer systems. This concept applied to DRAMs, which is often referred to as *approximate DRAM storage* (TEMAN et al., 2015), (JUNG et al., 2016), explores trade-offs involving energy savings and performance improvements in the memory subsystem against acceptable inaccuracy in the computation caused by non-critical data corruption.

1.2 Relevance and Contribution

The development of new DRAM refresh techniques with focus on energy savings and enhanced memory throughput is an active research topic in the DRAM community. Many research papers, some of them below mentioned, suggest the relevance of the topic.

The idea of postponing DRAM refresh is presented in (BHATI; CHISHTI; JA-

COB, 2013) and a temperature variation aware bankwise refresh for 3D-DRAMs is presented in (SADRI et al., 2014).

Some techniques allow parts of the memory to be corrupted such as the *REVA* scheme (ADVANI et al., 2014) that refreshes only a region of interest and *Flikker* (LIU et al., 2011) that reduces the number of refreshes by partitioning the DRAM in a critical and non-critical regions.

Selective refresh techniques are explored by PARIS (BAEK; CHO; MELHEM, 2014) that only refreshes rows that contain useful data. CREAM (ZHANG et al., 2014) shows per-bank and per-subarray refresh techniques.

Retention time aware refresh techniques are proposed in *RAIDR* (LIU et al., 2012) that group rows in retention time bins refreshed with different rates and *AVATAR* (QURESHI et al., 2015) that tries to overcome VRT issues by combining an on-line ECC detection with row selective refresh.

In this work *DRAMSys* (JUNG; WEIS; WEHN, 2015), a flexible memory subsystem design space exploration framework, is adapted and then combined with *3D-ICE* (SRIDHAR et al., 2014), a tool which can perform transient thermal analysis of integrated circuits, in order to study and better understand the impact on energy savings and retention error occurrence of a simple but audacious new approach to DRAM cells refresh - *omitting refresh*.

Furthermore, as additional contributions to the scientific community a SystemC wrapper was ported to inside the 3D-ICE project in order to easily provide integration of 3D-ICE with any SystemC/TLM2.0 based simulation environment and part of the effort made also contributed to a conference paper (JUNG et al., 2015).

2 DRAM

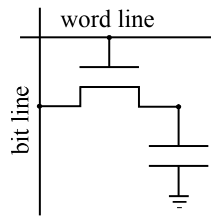
DRAM is a volatile memory, i.e., its content fades when power is turned off, and it is also dynamic, i.e., during normal operation the binary information written to a cell will leak off due to leakage currents. The time that a cell can hold the data is called *retention time*. Therefore, DRAM cells have to be periodically refreshed to maintain the data integrity. While a DRAM bit cell requires only a single transistor and capacitor pair an SRAM cell uses six transistors. Consequently, DRAM offers much higher densities and lower cost per bit. However, DRAM performance is worse than SRAM's mostly because the requirement of a specialized circuitry to sense and amplify slight voltage differences in the bitlines caused by a passive storage cell.

Commodity DRAM devices are designed to be simple and inexpensive to manufacture, so they do not bring much intelligence embedded on them. Thus the memory controller is responsible for refreshing all banks, ensuring that all timing constraints are met, avoiding timing violations, avoiding collisions on the data bus, respecting the turnaround time when the directions of the data bus changes, etc. Modern computers use improved versions of the original DRAM architecture. Along with process scaling, several techniques have been applied in order to extend the lifespan of the DRAM architecture up to these days. For example, the bandwidth was increased by doubling the data clock rate (data is transfered in both rising and falling edge of the clock signal), with on-die termination used to match line impedance increasing the quality of signals allowing higher frequency and the inclusion of delay-locked loop circuits used to compensate signal skew.

2.1 Basic Circuits

Modern DRAM devices use 1T1C DRAM cells to store a single bit of data. 1T1C DRAM cells consist of an access transistor controlled by a wordline, which selectively connects a bit storage capacitor to a bitline. This storage cell is a passive circuit that must be carefully sensed in order to read the logic level stored in it. The potentials V_{DD} or V_{SS} on the storage capacitor correspond to logical 1 or 0.

Figure 2.1: 1T1C DRAM Cell Structure

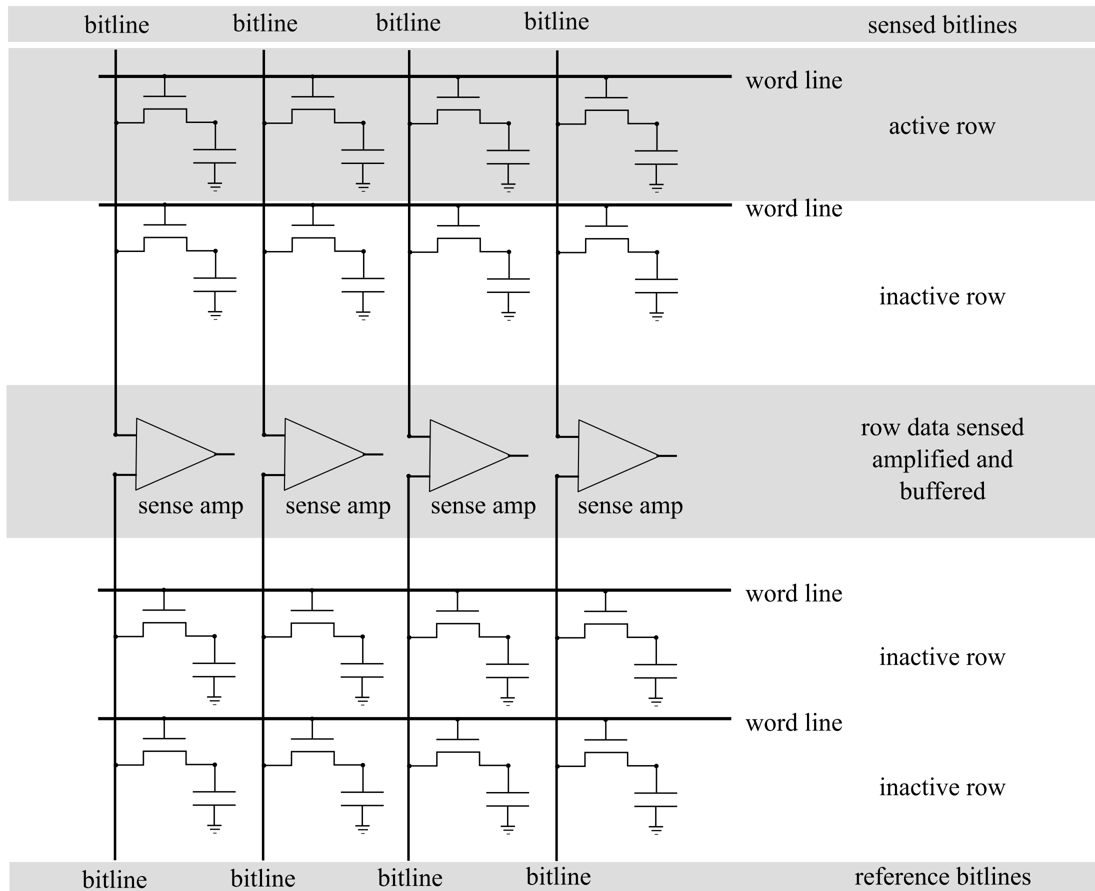


Multiple cells are connected to the same bitline but only one cell per bitline will have its access transistor open at a given time. When inactive, the bitlines are held at mid-rail potential $V_{DD}/2$. At the beginning of a memory access one of the wordlines is enabled causing a balance of charges between the bitline and the cell capacitor. Since the cell capacitance is roughly an order of magnitude smaller than the bitline capacitance the bitline voltage changes only a few hundred millivolts above or below $V_{DD}/2$. For this reason a *bitline sense amplifier* is needed to sense this small change in the bitline voltage and amplify it to the appropriate voltage.

The bitline sense amplifier is a regenerative latch that amplifies the bitline voltage to full-rail. The sense amplifiers will keep the resultant levels until the DRAM array is precharged for another access. Thus they act as a *row buffer* that caches an entire row of data. Subsequent reads to the same row of cells can be done directly from the sense amplifier without accessing the cells themselves, this is known as a *row hit*.

To sense very small changes in the bitline voltage modern DRAM devices use differential sense amplifiers which connect to a pair of bitlines. While the slight voltage variation of one line is sensed the other bitline works as a reference voltage. This, however, makes possible to sense only one bitline of the pair and consequently limits the number of memory cells that can be accessed at a given time.

Figure 2.2: Differential Sense Amplifiers (Open Bitline Structure)



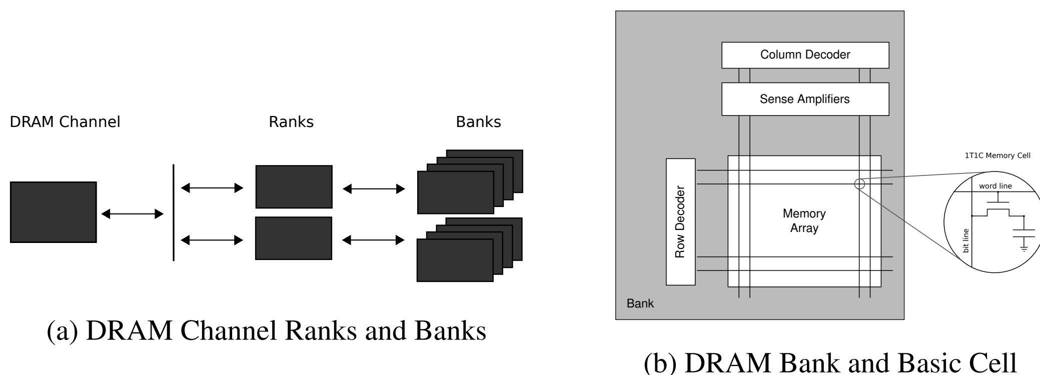
Differential sense amplifiers require bitlines that are very similar in terms of capacitance, voltage, path length and number of cells connected to them. The two main array structures of DRAM devices are open bitline and folded bitline. In the open bitline structure the array is divided in segments and differential sense amplifiers are connected to bitlines of different segments while in a folded bitline structures the pair of bitlines comes from the same array segment. Basically, the open bitline structure requires less area but, when compared to folded bitline structure, is more susceptible to electronic noise.

Since data reads to 1T1C DRAM cells are destructive (due to the balance of charges between the bit line and the storage cell), to complete the DRAM read cycle the data must be written back into the memory cell. The n-channel access transistor must be fully turned on. To store V_{DD} level in the memory cell, the wordline must be raised to $V_{DD} + V_T$ where V_T is the access transistor threshold voltage. Extra hardware may be required to generate this potential above V_{DD}.

2.2 Organization

DRAM memory cells are arranged together forming a bi-dimensional *array* of cells and each cell within such array can be accessed by a memory controller by specifying the row address and the column address to the DRAM. There is one or more arrays of cells inside a DRAM chip. Typically the memory arrays are designed to operate as a unity and as consequence the width in bits of each access to the memory chip is equal to the number of arrays contained in it. The number of arrays inside a chip can be used to characterize the DRAM. In a simple organization, for example a x8 DRAM chip (pronounced "by eight") has at least eight arrays of cells inside it and each column access (read or write) corresponds to an eight bits transfer (column width is 8 bits). Thus an 8-bit-wide-column is the basic addressable chunk of data for this device. A set of memory arrays that works independently of other sets is known as a *bank*. The bank is the granularity for memory operations, then commands targeting different banks can execute in parallel with respect to each other. DRAM devices can implement multiple independent banks internally.

Figure 2.3: Common DRAM Organization



(a) DRAM Channel Ranks and Banks

(b) DRAM Bank and Basic Cell

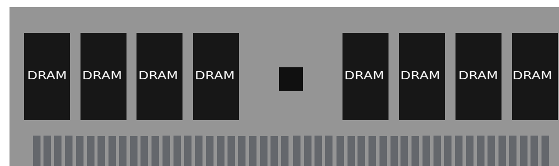
Source: (JACOB; NG; WANG, 2007)

Commercial DRAM modules are made of a circuit board with some DRAM chips plus associated circuitry attached to it and are typically sold as DIMMs. A system can have multiple DIMMs or one DIMM that implements separate groups of banks that can be accessed independently. The term *rank* is used to describe this level of independence. All ranks share the same address and data busses. Therefore individual chip select signals for each rank are required to put them in active or inactive state. Ranks whose chip select pin is held low are "deaf" and pay no heed to changes in the state of the bus, they hold their outputs in high impedance state allowing other devices to drive the lines. The selected rank whose chip select pin is held high is "listening" for changes and responds as if it was

the only chip on the bus.

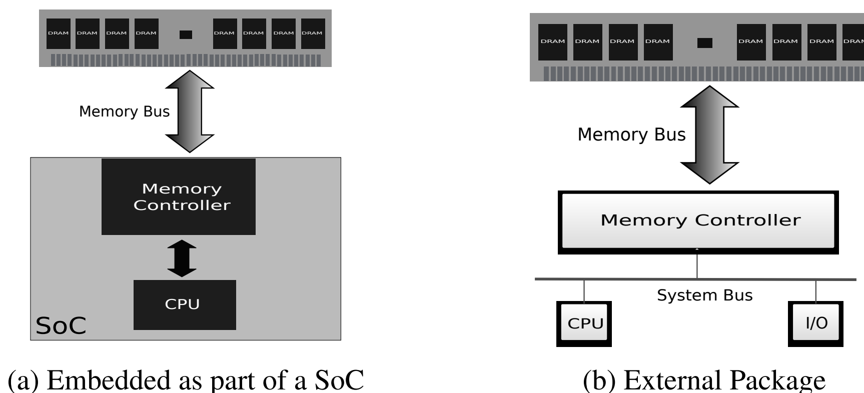
Typically, the bus width of a non-ECC memory is 64 bits (64 pins) while DRAM modules that provide ECC require 72 pins due to an extra chip required. In DDR memories the address is split into row and column addresses in order to save pins. Every bank is able to decode the address information and retrieve the data from the addressed cells.

Figure 2.4: DRAM DIMM



In traditional computer systems DRAM devices are arranged together to form a memory system that is managed by a single DMC. The DRAM memory controller may be in an external package connected to a processor or integrated in a SoC together with a processor core. DRAM controller and DRAM devices communicate via separate busses for data, address and control plus chip select signals.

Figure 2.5: DRAM Memory Controller



(a) Embedded as part of a SoC

(b) External Package

All memory transactions (e.g., read and write) are managed by the DMC. The DMC translates transactions into commands. Then it generates signal patterns in the memory bus in order to send the commands to the DRAM in the proper sequence and proper timing. The DMC is responsible to retrieve or store data on behalf of other devices and it is also responsible to issue refresh commands during normal operation. It can perform other tasks, e.g., ECC.

Additionally, modern DIMMs have a hardware feature called SPD that provides vital information to the system BIOS to keep the system working in optimal condition with the memory DIMM. JEDEC standards require that certain parameters be in the lower

128 bytes of an EEPROM located on the memory module. These bytes contain timing parameters, manufacturer, serial number among other useful information about the module. The SPD EEPROM is accessed using SMBus, a variant of the I^2C protocol. The communication is done via a simple two-wire bus that consists of a clock line and a data line.

2.3 DRAM Peculiarities

Due to process variation when integrated circuits are fabricated the attributes of nodes (length, width, oxide thickness) vary. Since the beginning of DRAM production different retention times among cells have been noticed, typically ranging from hundreds of milliseconds to tens of seconds (JACOB; NG; WANG, 2007, p. 356). Cells that are marked by shorter retention time are known as *weak-cells*.

Advances in fabrication process allow high-density DRAM devices, with smaller cells whose dimensions are approaching fundamental dimensions for current semiconductor technologies. For a given fabrication process, considering changes only in dimensions, not in materials (e.g., dopant elements, insulators), the smaller the number of atoms separating gate and drain the easier the leakage current flows and the higher the influence of the environment is.

Moreover, a not so often emphasized or sometimes forgotten phenomenon is the existence of DRAM cells with VRT. VRT is caused by microdefects in the p-n junction which are very difficult to detect. A microdefect of the right type and at the right place serve as an GR center. The GR center may conduct current through the p-n junction acting as an additional and significant leakage component (JACOB; NG; WANG, 2007, p. 826). Therefore there is a non-zero probability that a given cell changes its retention time for a while. The bi-modal retention time behaviour of some DRAM cells makes the estimation of a 100% accurate map of weak-cells infeasible. Since VRT is rare and difficult to detect, DRAM manufacturers are not interested in finding and eliminating it as long as the proposed refresh time is enough to make sure that even cells with VRT can pass all tests and work properly (JACOB; NG; WANG, 2007, p. 826).

2.4 Refresh

By the time of DRAM invention, as stated in its patent (DENNARD, 1968), the regeneration of the binary information stored in its cells (capacitors whose charge weakens over time) should take 10% to 20% of the device's operation time and conventional operations the remaining 80%. This means that since the very beginning of DRAM technology it is a well known fact that increases on DRAM refresh rates come with penalties: performance and energy efficiency degradation.

During normal operation the memory controller is responsible to send the right amount of refresh commands within a certain time window to ensure the integrity of the data in all memory cells. The duration of the time window is temperature dependent.

The refresh granularity vary among memory technologies. Nevertheless, a common fact is that the refresh of a memory segment takes tRFC and no accesses to that segment can be performed during this time. Additionally, right after the refresh operation all the rows within the refreshed segment are closed. Therefore the memory controller has to issue *Activate* commands to open rows and re-populate the row buffers. These inevitable row misses contribute to decrease the memory throughput as well.

Recent research on the impact of refresh for DRAM devices are presented in the table 2.1.

Table 2.1: Impact of Refresh on DRAM Devices

	Device Capacity (Gb)					
	2	4	8	16	32	64
Average Power	10%	15%	17%	24%	34%	47%
Energy	10%	14%	16%	18%	25%	n/a

Source: (LIU et al., 2012; BHATI et al., 2015)

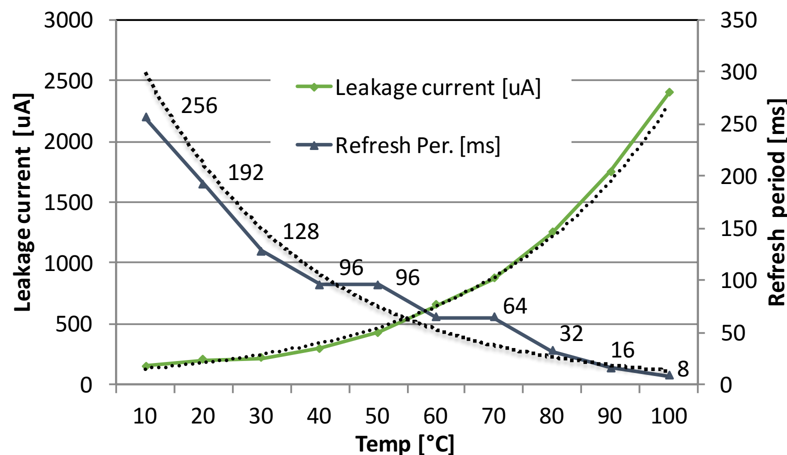
The JEDEC is an independent organization that is responsible for many standards related to semiconductor engineering. The maximum time interval between refresh operations is standardized by JEDEC for each DRAM technology and it is specified by DRAM manufacturers within the device's datasheet.

Typically a wide range of temperatures is considered as normal operation temperature. For example, the DDR4-SDRAM standard (JEDEC, 2012) specifies that temperatures ranging from 0 °C to 85 °C are the normal operation temperature for that technology. Nevertheless, for the same technology, a not so wide interval defines the extended temperature range: 85 °C to 95 °C.

The leakage current increases with the temperature and consequently the retention

time of the DRAM cells decreases. Thus the refresh commands have to be issued more often for higher temperatures in order to avoid retention errors (SADRI et al., 2014). Figure 2.6 presents the leakage current behaviour against temperature and the necessary adjustment to the refresh interval to completely avoid retention errors.

Figure 2.6: Leakage Current and Refresh Interval vs. Temperature.



Source: (LIU et al., 2013), (Micron Technology Inc., 2013), (SADRI et al., 2014) reviewed by (JUNG et al., 2015)

Though it is known that many cells can retain data for times much longer than the standard refresh interval, refresh requirements are based on a worst-case data decay time in order to ensure data integrity for all cells. The rareness of VRT allied to the capability of some applications to tolerate certain amount of errors make interesting the investigation of the impact on energy savings when disabling DRAM refreshes.

2.5 Error Correction Mechanisms

Along with temperature several aspects of the device's operation environment such as ambient radiation (from inside the computer system), electromagnetic interference, interactions with high-energy particles (from outside the computer system) have influence on chip-level soft errors.

Since errors in DRAM happen even though refresh requirements are met, some DRAM devices provide mechanisms for detection and possible correction of errors. Such mechanisms act decreasing the effective error rate and as consequence may be used to minimize some drawbacks of the approximate DRAM storage. In the other hand extra storage and logic are required by ECC memory with impact on power consumption and

final costs. Thus the decision of combining ECC memory with approximate DRAM is non-obvious but is beyond the scope of this work.

2.6 Technological Trends

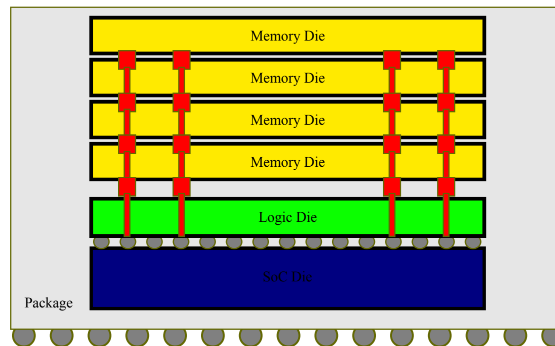
It is uncertain how long process scaling and improvements will be sufficient to maintain today's DRAM architecture. Efficiency and performance gains using the traditional DRAM have become smaller and more difficult to achieve. New solutions are necessary to attend the ever growing demand for memory performance. Therefore DRAM manufactures search for innovations to DRAM memory architecture that lead to higher performance and lower power consumption.

2.6.1 Wide I/O

The Wide I/O memory architecture is designed to consume low power and provide a high bandwidth with focus on mobile devices and battery powered embedded systems. Wide I/O can be stacked on top of system on chip (SoC) devices by means of vertical through-silicon via (TSV) interconnections. This accounts in favor of an optimized package size. On the other hand, the heat radiated from the SoC will pass through the memory die worsening the thermal scenario. Wide I/O high bandwidths are due to a wide memory bus that is up to 1024 bits wide.

The current Wide I/O standard supports memory stacks of up to four memory chips. Each memory chip in the stack is called a *slice* and a single memory chip is divided into four *quadrants*. The electrical connection between two stacked dies is know as *Micropillar*. A *channel* is a set of physically discrete connections within the Wide I/O interface which contains all the control, data and clock signals necessary to independently control a partition of the Wide I/O device. A *rank* is a portion of memory connected to a channel. Multiple slices may be connected to form a rank.

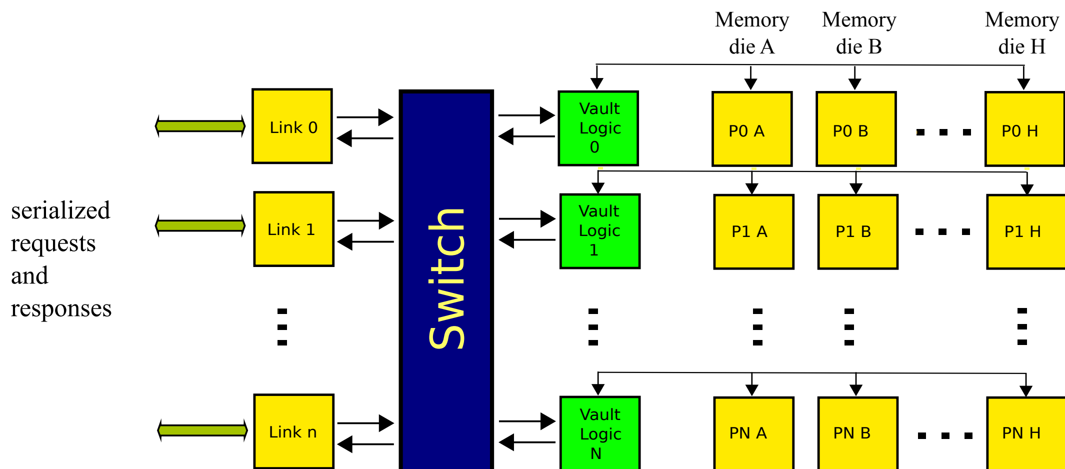
Figure 2.7: Wide I/O 3D memory stack



2.6.2 Hybrid Memory Cube

A HMC is a single package containing multiple DRAM dies (four or eight are possible according to the current specification) and one logic die, all stacked together using through-silicon via technology. Within each cube, memory is organized vertically. Each memory die is divided into partitions. The combination of vertically aligned partitions (one of each memory die in the stack) with a memory controller within the logic die is called a vault and is comparable to a channel in Wide I/O architecture.

Figure 2.8: A representation of the HMC scheme in respect to memory accesses



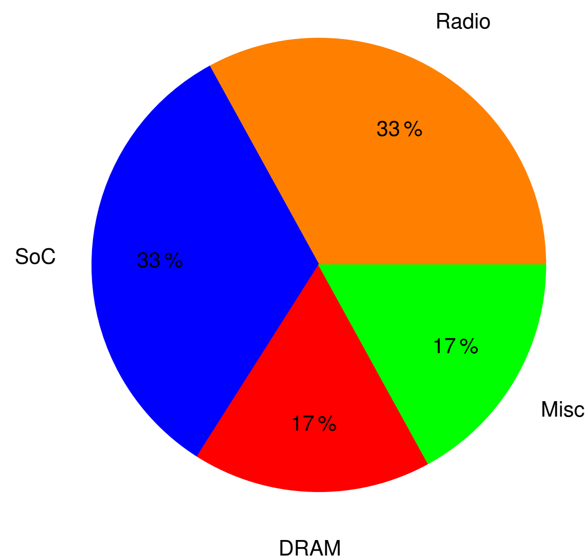
The vault controller manages all memory reference operations within the vault. Operations may be buffered and executed out of order in respect to the order of arrival. Responses from vault operations back to the external serial links will be out of order. However, requests from a single serial link to the same vault/bank address are executed in order. Requests from different external serial links to the same vault/bank address are not guaranteed to be executed in a specific order and must be managed by the host controller.

Each vault controller determines its own timing requirements and controls its refresh rate.

2.7 Impact on Power

Energy and power usage are important concerns in the design of computer systems. This section presents power breakdowns, extracted from publications, for different computer systems. Figure 2.9 shows the power breakdown for a smartphone from 2013 operating in suspend state and 3G radio enabled.

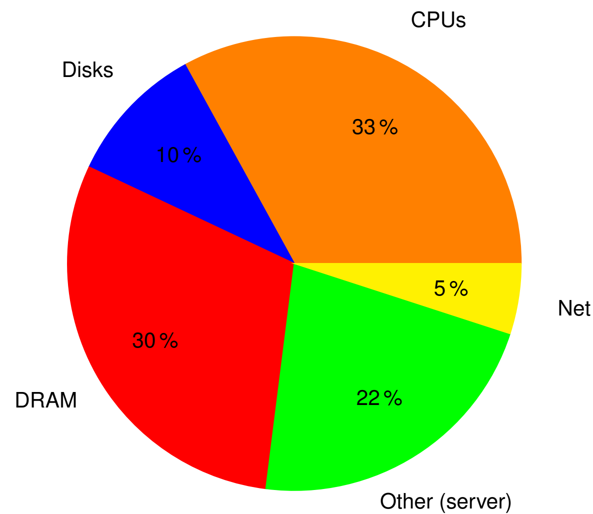
Figure 2.9: Power Breakdown Samsung Galaxy S III (suspended state, 3G enabled)



Source: (CARROLL; HEISER, 2013)

Figure 2.10 and figure 2.11 provides a power breakdown for a datacenter equipment. The authors of (BARROSO; HÖLZLE, 2009) showed that in 2007 the relative power consumption of the memory system was very close to the CPU consumption.

Figure 2.10: Power Breakdown Google's Datacenter (2007)

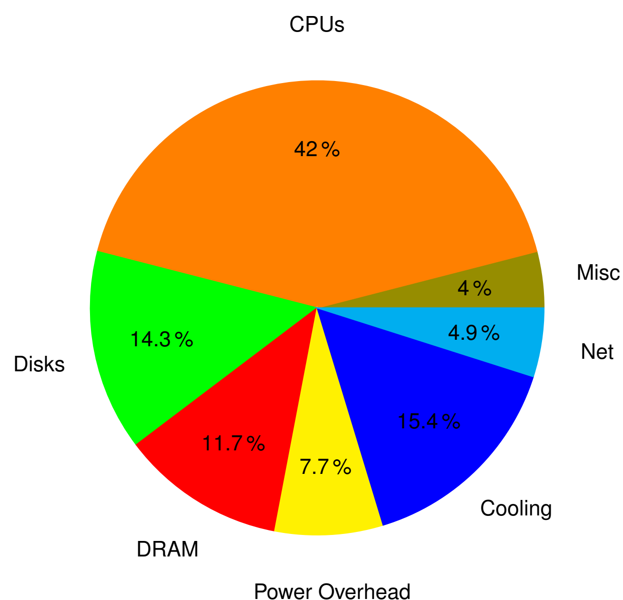


Source: (BARROSO; HÖLZLE, 2009)

In its second edition (BARROSO; CLIDARAS; HÖLZLE, 2013) presents the results for the year 2012. According to the authors the expressive decrease in the DRAM portion of the total power has more than one cause:

- Improved thermal management have allowed CPUs to run closer to their maximum power envelope resulting in higher energy consumption per CPU.
- Power hungry FB-DIMMs were replaced by DDR3-SDRAMs which consume less.
- The DRAM's voltage has dropped from 1.8 V down to 1.5 V and below.

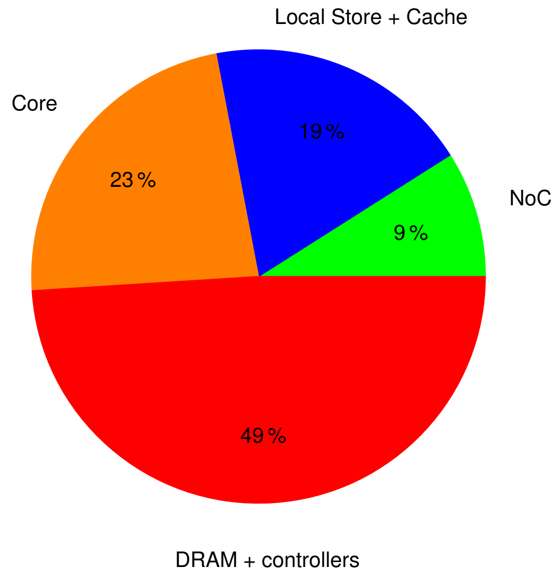
Figure 2.11: Power Breakdown Modern Datacenter (2012)



Source: (BARROSO; CLIDARAS; HÖLZLE, 2013)

Fiugure 2.12 presents a power breakdown for an energy-efficient seismic simulation platform.

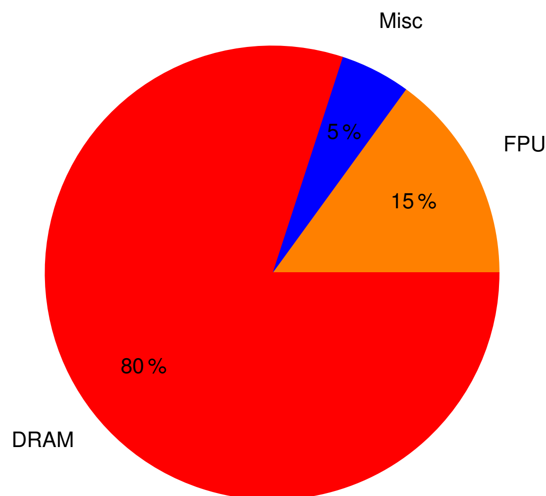
Figure 2.12: Power Breakdown Energy-Efficient Seismic Simulation Platform



Source: (KRUEGER et al., 2011)

Figure 2.13 for a multi-chip custom digital super-computer called eBrain.

Figure 2.13: Power Breakdown eBrain



Source: (FARAHINI et al., 2014)

The information presented on this section is intended to provide some insights about the impact of DRAMs on the power and energy consumption for different computer systems. It shows that the portion related to DRAM is not negligible for any of the applications described and also shows that for specific applications the DRAM impact can be very high.

3 DESIGN SPACE EXPLORATION TOOLS

Scalability, speed and low costs of production allied to a successful technological evolution sustain DRAM hegemony as primary memory of computer systems. Although current technology scaling is reaching its limits, demands for memory capacity, high-performance and energy-efficiency are increasing in what looks like an unstoppable trend. Also, technological advances come with new challenging engineering problems that need to be fully understood and solved.

The design space of DRAM is huge. In order to face the new challenges that arise designers must have an in-depth understanding of the memory system and its subsystems. This combined with a detailed and accurate model of the memory system may provide a way of increasing the intuition and knowledge on how the system behaves when its parameters vary. Hence simulation provides substantial help for design space exploration. It plays an important role saving resources and anticipating the release of new solutions. DRAM design space exploration tools empower researchers and engineers with flexibility and agility to find out how to enhance current devices, to foresee how systems will behave in extreme conditions, to elaborate new refresh strategies aiming optimization, and even to create totally new models.

3.1 SystemC and TLM

SystemC is a set of C++ classes which can be used to develop event-driven applications. SystemC is described and documented in the IEEE Standard for Standard SystemC Language Reference Manual (IEEE Computer Society, 2012).

TLM is an approach to modeling digital systems focused in the abstraction of communication between the primary functional blocks within a system. TLM2.0 is described and documented in a reference manual released by the Open SystemC Initiative group (AYNSLEY, 2009).

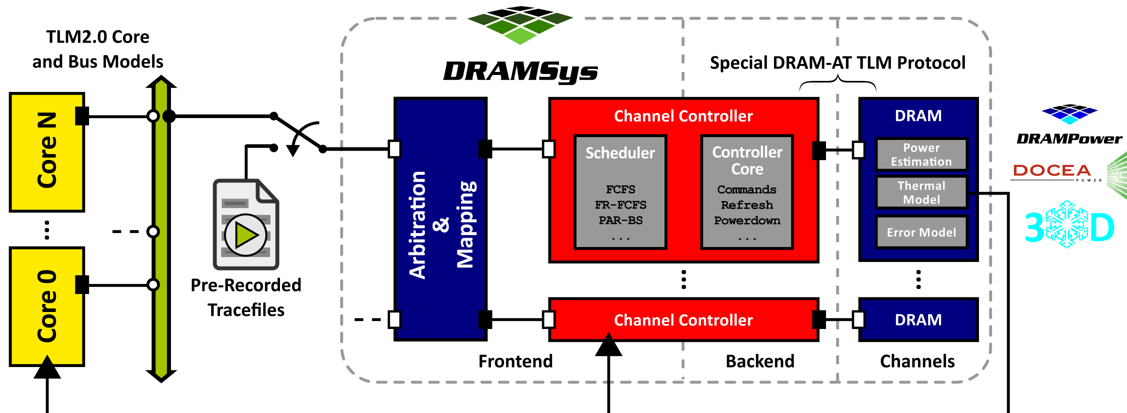
Combined they represent a powerful alternative to create fast and still accurate virtual platforms typically used for performance analysis and architecture exploration.

3.2 DRAMSys

DRAMSys is a flexible DRAM subsystem design space exploration framework that consists of models reflecting the DRAM functionality, power consumption and retention time errors. The exploration framework models a wide range of standard and emerging DRAM subsystems such as DDR3, DDR4, LPDDR3, Wide I/O and HMC. Implemented in C++ and SystemC this framework offers interoperability with third party tools.

A key concept behind this tool is the use of TLM2.0 to implement AT simulation based on function calls instead of a pin accurate register transfer level simulation where all events are simulated. This approach ensures faster simulation with negligible losses in accuracy (JUNG et al., 2013; JUNG; WEIS; WEHN, 2015).

Figure 3.1: DRAMSys Base Architecture



Source: (JUNG; WEIS; WEHN, 2015)

DRAMSys is able to process pre-recorded trace files containing memory transactions. The framework supports trace files from other simulators like Gem5 (BINKERT et al., 2011) or Simplescalar (BURGER; AUSTIN, 1997). This feature facilitates efficient analysis and explorations.

3.2.1 Error Model

DRAMSys implements a DRAM bit error model (WEIS et al., 2015) that enables early investigations on retention time behaviour against temperature. The DRAM bit error model takes into consideration the possibility of DRAM cells with variable retention time behaviour. The current version supports Wide I/O and DDR3-SDRAM memories.

3.3 3D-ICE

3D-ICE stands for 3D Interlayer Cooling Emulator. It is a Linux based Thermal Emulator Library written in C, which can perform transient analysis of 2D or 3D integrated circuits. The tool offers a client-server mode which is the operation mode relevant to this work. This feature is very useful when power traces are dynamically generated and to simulate the temperature behaviour in run-time. Client and server exchange messages through the network, the client provides power information to the server and requests the server to perform thermal simulation steps based on it. Thermal maps of individual dies are available as thermal data output of the simulator.

A 3D-ICE project is composed of a stack description file and one or more floorplan files. The stack description file describes structure, material properties of the 3D stack, possible heat sinks in the system, the discretization parameters, analysis parameters, and 3D-ICE commands specifying the desired outputs for the simulation. The floor plan file contains the coordinates and dimensions of the logic blocks relevant to the thermal simulation. Power information for every block specified in this file must be provided to the simulator in order to execute a thermal simulation step.

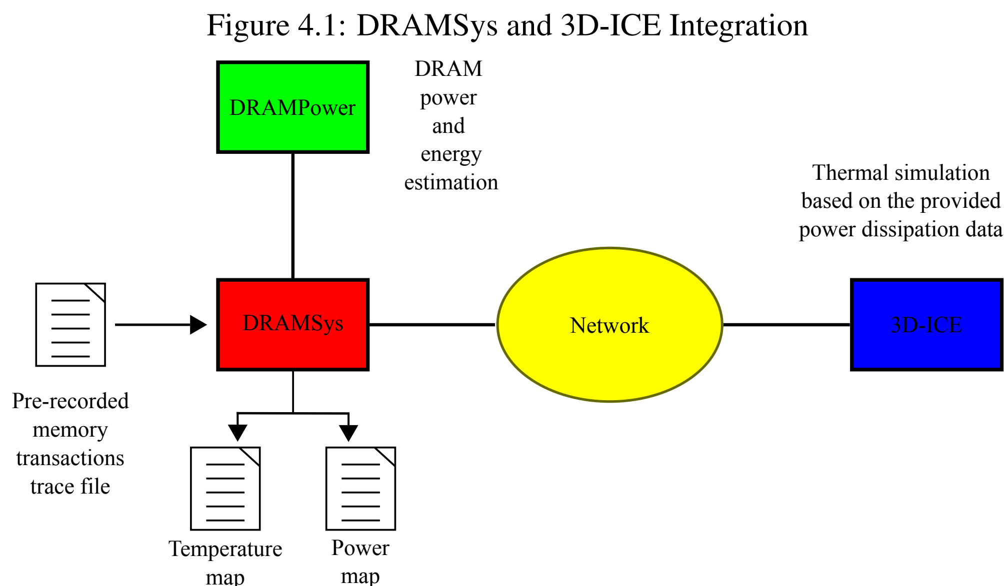
3.4 DRAMPower

DRAMPower (CHANDRASEKAR et al., 2011) is an open source tool for fast and accurate DRAM power and energy estimation. Its current version supports DDR2, DDR3, DDR4, LPDDR, LPDDR2, LPDDR3 and Wide I/O DRAM memories based on JEDEC standards. DRAMPower can be compiled as a library and called directly from a simulator using a provided API. This is the approach adopted by DRAMSys that simulates the DRAM memory controller sends commands to the library. Also, part of the library can be found inside Gem5 where it is employed for power and energy estimation.

4 IMPLEMENTATION AND RESULTS

In this work a simulation environment which consists of the tools presented in the previous chapter was adapted in order to investigate how disabling refreshes affects the energy consumption and the occurrence of retention errors. Since temperature plays an important role in the occurrence of retention errors, 3D-ICE was used for thermal simulation in some experiments providing dynamic temperature values during the simulation. DRAMSys was adapted to act as 3D-ICE client. The client obtains power information the DRAMPower library and forwards it to the server during the request of a thermal simulation step. The server then generates temperature and power maps.

Figure 4.1: DRAMSys and 3D-ICE Integration



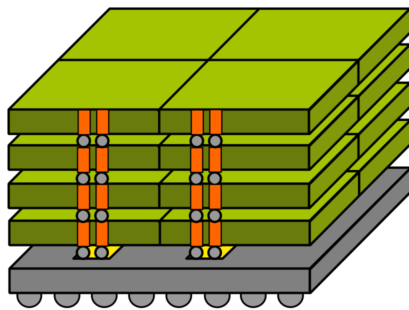
A new SystemC module was created in DRAMSys to connect it to the 3D-ICE server. This module contains a thread process that is initialized at the beginning of the simulation and is responsible for the requests to the thermal simulator. Since thermal simulation adds an extra time overhead to the total simulation time, the period for the execution of thermal simulation steps can be chosen by the user via DRAMSys' configuration. The code for the module created can be found in the appendix A.

An additional piece of software *IceWrapper* was used to implement the interface between DRAMSys and 3D-ICE. It was incorporated to DRAMSys' repository as a submodule. The IceWrapper was extended to provide ability to request a power map from the 3D-ICE server. Later the IceWrapper itself was ported to inside the 3D-ICE project in order to easily provide integration of 3D-ICE with any SystemC/TLM2.0 based simulation environment. Changes are currently being reviewed by the 3D-ICE team and should

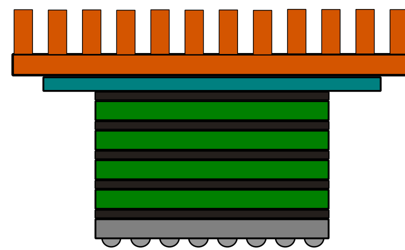
be incorporated to the project in the near future.

A 3D-ICE project that consist of three input files describing the test setup was created and can be found in the appendix B. The files describe a 3D-DRAM chip mounted on top of a SoC. They also define a heat sink on top of the DRAM die.

Figure 4.2: Graphical representation of some aspects of the 3D-ICE project



(a) 3D Memory on top of a SoC



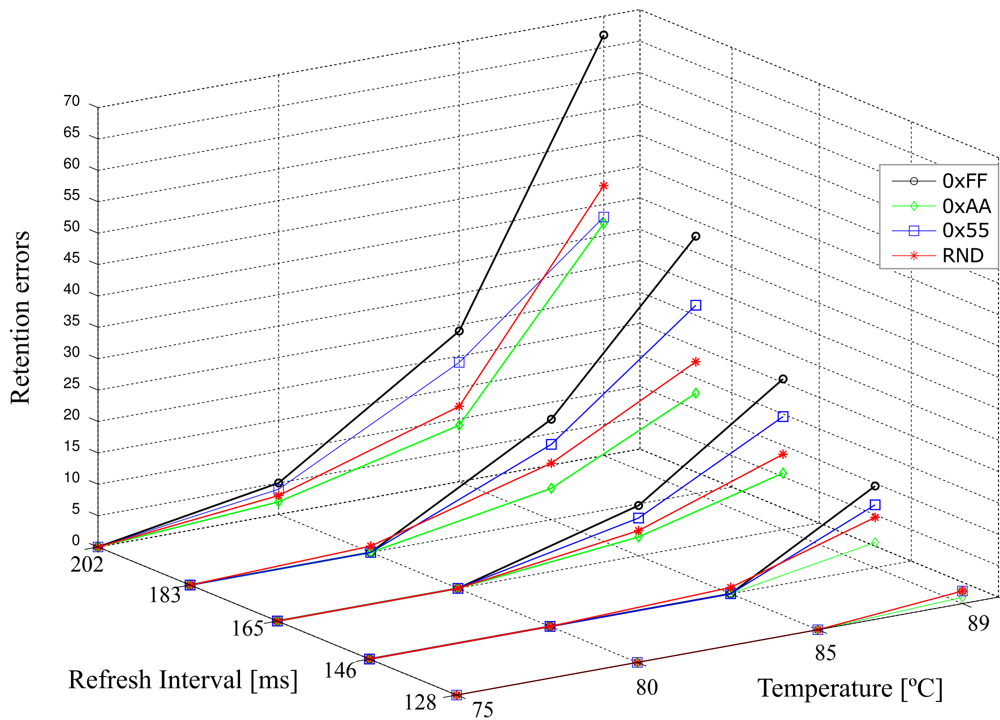
(b) A heat sink mounted on top of the stack

4.1 Simulation of WIDE I/O Memory

Faster simulations can be achieved by replaying prerecorded transaction trace files (KOGEL, 2010). This convenient approach is adopted in this work for the sake of time saving.

In order to investigate the behaviour of the retention time errors input trace files were generated in a way to fill up the memory with a certain pattern and after some time read it back. Several simulations with a static temperature value ranging from 75 °C to 89 °C were executed for different refresh intervals, all of them greater than the standard refresh interval, so errors could happen. Figure 4.3 shows the results.

Figure 4.3: Retention Errors for different Refresh Intervals and Temperatures



In a further experiment a region of 128 MiB of the memory was filled with data patterns (similar to the ones used in the previous experiment) and after some time its content was read back. The refresh interval chosen was 350 milliseconds that is more than five times the interval in which all memory cell are usually refreshed. For this experiment the temperature was dynamically simulated. Figure 4.4 presents the trend for retention errors against the temperature.

Figure 4.4: Retention Errors vs. Temperature (350 ms without refresh)

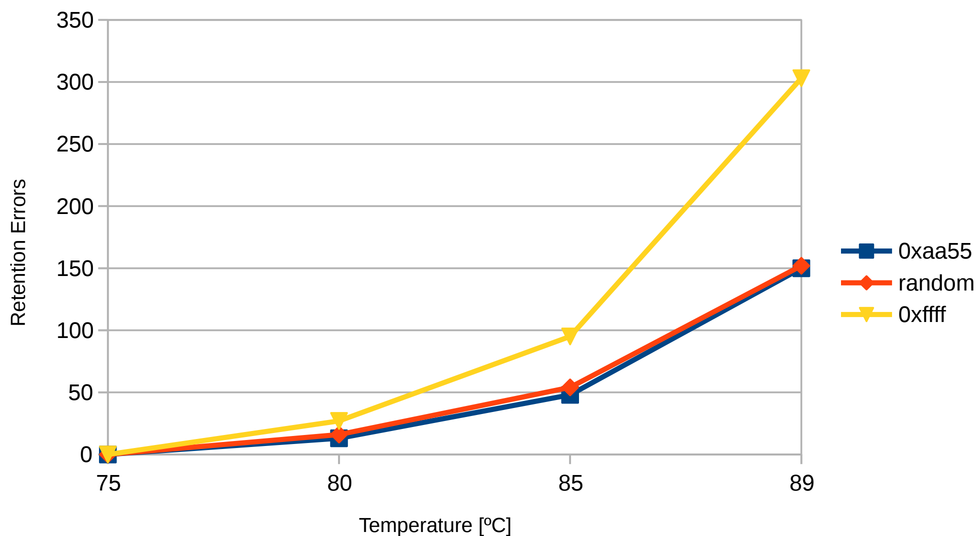


Table 4.1 shows the number of bit-flips observed for each pattern with refresh

enabled and refresh disabled.

Table 4.1: Retention Errors (350 ms without refresh)

Retention Errors		
DataPattern	Normal Refresh	Omitted Refresh
0xFFFF	0	294
0xAA55	0	153
RND	0	149

Table 4.2 presents a comparison of the energy consumption and the average power. The savings are around 24%.

Table 4.2: Energy and Power Comparison (350 ms without refresh)

	Normal Refresh	Omitted Refresh
TotalEnergy[mJ]	10.03	7.62
AveragePower[mW]	26.73	20.33

From the results just presented it is possible to conclude that the energy consumption and average power can be reduced by about 24% by omitting refreshes at the price of tolerating 294 bit flips.

Nevertheless, from the several simulations with different retention times and temperatures it was possible to realize that the gains are very application dependent. For generalization the following applies:

- The number of retention errors depends on the data lifetime (how long the data must remain in the memory) and also on the operation temperature.
- Refresh can be omitted only for resilient applications, i.e., applications that tolerate a certain amount of errors.
- The total energy saving is due to refresh omission during the application run time. Therefore it is application dependent.

4.2 Applicability of Approximate DRAM Storage

The simulations showed (figure 4.3) that the occurrence of retention errors when exceeding the standard refresh interval is low even for high operation temperatures. Nevertheless, since errors may happen approximate DRAM storage is interesting for error re-

silient applications only. Fortunately error resilience is a characteristic that can be found on a broad range of applications such as signal processing, image, audio, and video processing, graphics, wireless communications, web search, and data analytics (CHIPPA et al., 2013). Examples of applications that could derive benefit on terms of energy savings, reduced power peaks and performance gains from the suppression of DRAM refresh are presented below.

- All sorts of electronic gadgets that can tolerate occasional inaccuracy in the output or even failures that humans are not able to perceive.
- Smart TV devices operating in energy-saving mode could use unreliable DRAM for buffering the video frames. Since the image refresh rate is relatively high the human eye would hardly detect a few failures caused by bit flips in the DRAM.
- End-user telecommunication devices in which DRAM is used as temporary storage for the network packets when using unreliable protocols with respect to data delivery. In general, non-critical voice and video traffic is transmitted using UDP. From the application point of view a slight degradation in quality due to lost of datagrams is better than large delays due to retransmission. Therefore those applications show an inherent resilience to errors what accounts in favor of DRAM refresh suppression.
- An algorithm to adaptive control of screen brightness may tolerate some imprecision on the brightness level of a few pixels since it is something hard to notice and most of the time acceptable.
- The lifetime of the streaming data in embedded systems may be less than the DRAM refresh period. As suggested in (ADVANI et al., 2014) wearable video systems such as smart glasses require capabilities for real-time video analytics and prolonged battery lifetimes.
- As stated in (JUNG et al., 2016), channel decoding techniques used in digital communication systems such as WiGig and WiFi are error resilient and can tolerate a certain amount of bit errors due to the lack of DRAM refreshes with negligible impact on the application performance.

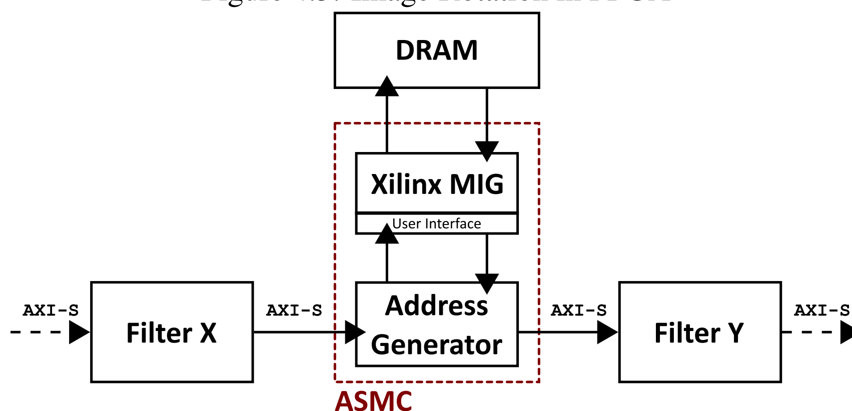
It is important to note that not all data can be stored into unreliable memory. Thus the use of approximate DRAM is a design decision since extra hardware may be required (e.g., the addition of an extra memory rank to the system to be used as unreliable memory or make use of DRAM that provides the necessary features to create reliable and unreli-

able sectors). Furthermore, the software needs to match up the hardware and implement mechanisms to explore its capabilities.

4.3 Hardware Experiment with DDR3-SDRAM and FPGA

An experiment using real hardware was carried out to show the feasibility of the approach for a real image processing application. In this experiment, an image stream from a camera is rotated by 90 degrees using the DDR3-SDRAM (1 GiB DDR3 SO-DIMM 64 bit data bus, 4 devices x16) as temporary data storage. Four chips together result in 1 GiB (or 8 Gb) storage thus each device has a density of 2 Gb and provide 16 bits of data each (by 16). Each image frame has 4,525,056 pixels with 48 bits per pixel (approximately 26 MiB of data). The image is stored into the DRAM at consecutive addresses and it is retrieved back from specific locations in a way such that the output image is rotated 90 degrees. It is important to mention that the results of this experiment were published in (JUNG et al., 2015).

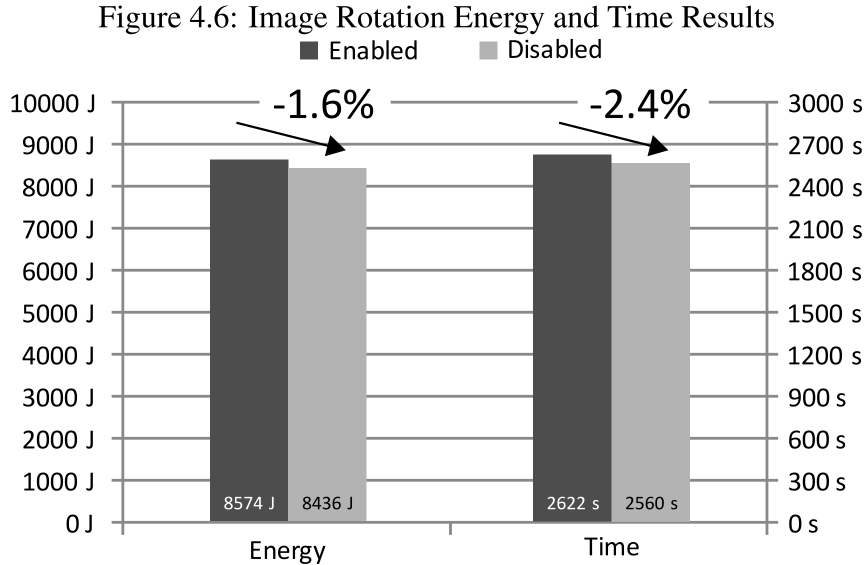
Figure 4.5: Image Rotation in FPGA



The application has a real-time constraint of 8.6 ms on the frame rate. Therefore, no frame stays in the memory longer than 8.6 ms and being this time much less than the typical refresh window time of 64 ms (in which all rows and consequently all cells must be refreshed) no errors are expected to happen when refresh is omitted. Indeed, all frames were checked for data corruption and no bit errors occurred.

Figure 4.6 presents the results in terms of energy and time consumption for the same task with refresh enable and refresh disabled.

Figure 4.6: Image Rotation Energy and Time Results



The normal runtime to process 611,620 frames is 43.7 minutes. This time was reduced by 62 seconds and the total energy was reduced by 1.6% with refreshes disabled. Even though the savings in time and energy are modest, this experiment shows the feasibility of the approach for a real application.

The modest gains observed are due to the low refresh overhead for the memory used. Since the memory's density is relatively low the refresh overhead is low. Higher densities imply more cells to be refreshed within the same refresh time window. Therefore the memory requires more time to conclude the refresh operation internally after receiving an auto refresh command. The refresh overhead can be calculated with equation 4.1:

$$refresh\ overhead = \frac{time\ required\ for\ refresh}{refresh\ window\ time} \tag{4.1}$$

The time required for refreshing all the rows (then all the memory cells) can be obtained by multiplying refresh cycle time by the number of auto refresh commands that must be issued within the refresh window interval. Equation 4.1 can be rewritten in terms of information from the memory specification as shown in equation 4.2:

$$refresh\ overhead = \frac{t_{RFC} \times number\ of\ AR\ commands\ in\ a\ window}{t_{REFW}} \tag{4.2}$$

According to the DDR3-SDRAM standard (JEDEC, 2010) the refresh cycle time $t_{RFC} = 160\ ns$ for the device density of 2 Gb and it requires refresh cycles at an average periodic interval of $t_{REFI} = 7.8\ \mu s$ for the test conditions (normal temperature range $0\,°C$ to $85\,°C$). Also, for the test scenario $t_{REFW} = 64\ ms$ what means that 8192 AR commands are issued within a window. By applying equation 4.2 it is possible to obtain

the refresh overhead.

$$refresh\,overhead = \frac{160 \times 10^{-9} \times 8192}{64 \times 10^{-3}} = 2.048\% \tag{4.3}$$

Additionally to the refresh overhead calculated above there are some extra penalties that explain the observed time savings that are greater than the refresh overhead:

- All banks of the SDRAM must be precharged and idle for a minimum of the precharge time $t_{RP}(min) = 12.5\ ns$ before the refresh command can be applied. Therefore, the memory controller must issue an precharge-all command before issuing the refresh command when necessary.
- When the refresh cycle has completed, all banks of the SDRAM will be in the precharged (idle) state. The data previously loaded in the row buffers is lost and a miss will inevitably occur in case a read is performed (the data can not be directly retrieved from the sense amplifiers).

5 CONCLUSION

The use of an unreliable DRAM (or unreliable sectors within the DRAM) means that some bits may flip resulting in some imprecision in the final result. Nonetheless, the imprecision can be reduced, for example, by storing only the least significant bits of a word in a lesser reliable area of the memory. This can be achieved if the software layers of a system implement a finer level of control suitable to the hardware features. Operating systems must be aware of the hardware capabilities and implement proper device drivers to explore the features provided.

An operating system aware of the accuracy level required by a given task has to be able to identify different kinds of data for that task and choose the proper storage location in the DRAM that satisfies the task's requirements. Alternatively, the operating system could first store the data then dynamically change the reliability level (by disabling or re-enabling refresh) to fit the task with possible energy savings.

The software should be able to allocate precise storage for precision sensitive parts of an application and also allocate imprecise storage to imprecision tolerant data. Programming languages and compilers could be precision aware and define mechanisms e.g., variable qualifiers, to allow programmers to choose among reliability levels.

Much has to be done in the hardware software interface in order to have general-purpose computers that dynamically achieve a desirable trade-off between efficiency (in terms of performance gains or energy savings) and acceptable inaccuracy of the output.

The audacious approach presented in this work minimizes the demands regarding hardware and software changes and so facilitates design explorations. Evidently the applicability of this radical strategy has a limited scope, i.e., it is not suitable for all general-purpose computation. Nevertheless researchers and designers can easily apply this approach for testing and eventually make use of it in some of today's special-purpose applications.

The omission of DRAM refreshes is beneficial to applications that exhibit some resilience to errors and require high memory throughput. Moreover, it can extend the battery life in mobile devices that most of the time run non-critical applications creating an interesting scenario in which loss of accuracy of the output is acceptable and energy savings are very welcome.

REFERENCES

ADVANI, S. et al. Refresh Enabled Video Analytics (REVA): Implications on power and performance of DRAM supported embedded visual systems. In: **Computer Design (ICCD), 2014 32nd IEEE International Conference on**. [S.l.: s.n.], 2014. p. 501–504.

AYNSLEY, J. **OSCI TLM-2.0 Language Reference Manual**. Open SystemC Initiative (OSCI), 2009. Disponível em: <http://www.accellera.org/downloads/standards/systemc>.

BAEK, S.; CHO, S.; MELHEM, R. Refresh now and then. **Computers, IEEE Transactions on**, IEEE, v. 63, n. 12, p. 3114–3126, 2014.

BARROSO, L. A.; CLIDARAS, J.; HÖLZLE, U. **The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines, Second Edition**. Morgan & Claypool Publishers, 2013. (Synthesis Lectures on Computer Architecture). ISBN 9781627050098. Disponível em: <http://dx.doi.org/10.2200/S00516ED2V01Y201306CAC024>.

BARROSO, L. A.; HÖLZLE, U. **The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines**. Morgan & Claypool Publishers, 2009. (Synthesis Lectures on Computer Architecture). Disponível em: <http://dx.doi.org/10.2200/S00193ED1V01Y200905CAC006>.

BENINI, L.; BOGLIOLO, A.; MICHELI, G. D. A survey of design techniques for system-level dynamic power management. **Very Large Scale Integration (VLSI) Systems, IEEE Transactions on**, v. 8, n. 3, p. 299–316, June 2000. ISSN 1063-8210.

BHATI, I. et al. DRAM Refresh Mechanisms, Trade-offs, and Penalties. **Computers, IEEE Transactions on**, PP, n. 99, p. 1–1, 2015. ISSN 0018-9340.

BHATI, I.; CHISHTI, Z.; JACOB, B. Coordinated Refresh: Energy Efficient Techniques for DRAM Refresh Scheduling. In: **Proceedings of the 2013 International Symposium on Low Power Electronics and Design**. Piscataway, NJ, USA: IEEE Press, 2013. (ISLPED ’13), p. 205–210. ISBN 978-1-4799-1235-3. Disponível em: <http://dl.acm.org/citation.cfm?id=2648668.2648720>.

BINKERT, N. et al. The Gem5 Simulator. **SIGARCH Comput. Archit. News**, ACM, New York, NY, USA, v. 39, n. 2, p. 1–7, ago. 2011. ISSN 0163-5964. Disponível em: <http://doi.acm.org/10.1145/2024716.2024718>.

BURGER, D.; AUSTIN, T. M. The SimpleScalar Tool Set, Version 2.0. **SIGARCH Comput. Archit. News**, ACM, New York, NY, USA, v. 25, n. 3, p. 13–25, jun. 1997. ISSN 0163-5964. Disponível em: <http://doi.acm.org/10.1145/268806.268810>.

CAMERON, K. W.; GE, R.; FENG, X. High-performance, power-aware distributed computing for scientific applications. **Computer**, v. 38, n. 11, p. 40–47, Nov 2005. ISSN 0018-9162.

CARROLL, A.; HEISER, G. The systems hacker’s guide to the galaxy energy usage in a modern smartphone. In: **Proceedings of the 4th Asia-Pacific Workshop on Systems**. New York, NY, USA: ACM, 2013. (APSys ’13), p. 5:1–5:7. ISBN 978-1-4503-2316-1. Disponível em: <http://doi.acm.org/10.1145/2500727.2500734>.

CHANDRASEKAR, K. et al. **DRAMPower: Open-source DRAM Power & Energy Estimation Tool**. 2011. <http://www.drampower.info>. [Online, last accessed 2016-04-13].

CHIPPA, V. K. et al. Analysis and Characterization of Inherent Application Resilience for Approximate Computing. In: **Proceedings of the 50th Annual Design Automation Conference**. New York, NY, USA: ACM, 2013. (DAC '13), p. 113:1–113:9. ISBN 978-1-4503-2071-9. Disponível em: <http://doi.acm.org/10.1145/2463209.2488873>.

DENNARD, R. H. **Field-effect transistor memory**. [S.l.]: Google Patents, 1968. <http://www.google.com/patents/US3387286>. US Patent 3,387,286.

FARAHINI, N. et al. A scalable custom simulation machine for the bayesian confidence propagation neural network model of the brain. In: **2014 19th Asia and South Pacific Design Automation Conference (ASP-DAC)**. [S.l.: s.n.], 2014. p. 578–585. ISSN 2153-6961.

IEEE Computer Society. **IEEE Standard for Standard SystemC Language Reference Manual**. 2012.

JACOB, B.; NG, S.; WANG, D. **Memory Systems: Cache, DRAM, Disk**. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2007. ISBN 0123797519, 9780123797513.

JEDEC. **JESD79-3E**. 2010.

JEDEC. **JESD79-4A**. 2012.

JUNG, M. et al. Efficient reliability management in SoCs - an approximate DRAM perspective. In: **2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)**. [S.l.: s.n.], 2016. p. 390–394.

JUNG, M.; WEIS, C.; WEHN, N. DRAMSys: A Flexible DRAM Subsystem Design Space Exploration Framework. **IPSJ Transactions on System LSI Design Methodology**, v. 8, p. 63–74, 2015.

JUNG, M. et al. TLM Modelling of 3D Stacked Wide I/O DRAM Subsystems: A Virtual Platform for Memory Controller Design Space Exploration. In: **Proceedings of the 2013 Workshop on Rapid Simulation and Performance Evaluation: Methods and Tools**. New York, NY, USA: ACM, 2013. (RAPIDO '13), p. 5:1–5:6. ISBN 978-1-4503-1539-5. Disponível em: <http://doi.acm.org/10.1145/2432516.2432521>.

JUNG, M. et al. Omitting refresh: A case study for commodity and wide i/o drams. In: **Proceedings of the 2015 International Symposium on Memory Systems**. New York, NY, USA: ACM, 2015. (MEMSYS '15), p. 85–91. ISBN 978-1-4503-3604-8. Disponível em: <http://doi.acm.org/10.1145/2818950.2818964>.

KOGEL, T. **Generating Workload Models from TLM-2.0-based Virtual Prototypes for Efficient Architecture Performance Analysis**. 2010. <http://www.nascug.org/events/13th/tlm20_workload_models.pdf>.

KRUEGER, J. et al. Hardware/software co-design for energy-efficient seismic modeling. In: **2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC)**. [S.l.: s.n.], 2011. p. 1–12. ISSN 2167-4329.

LIU, J. et al. An Experimental Study of Data Retention Behavior in Modern DRAM Devices: Implications for Retention Time Profiling Mechanisms. **SIGARCH Comput. Archit. News**, ACM, New York, NY, USA, v. 41, n. 3, p. 60–71, jun. 2013. ISSN 0163-5964. Disponível em: <http://doi.acm.org/10.1145/2508148.2485928>.

LIU, J. et al. RAIDR: Retention-Aware Intelligent DRAM Refresh. In: **Proceedings of the 39th Annual International Symposium on Computer Architecture**. Washington, DC, USA: IEEE Computer Society, 2012. (ISCA '12), p. 1–12. ISBN 978-1-4503-1642-2. Disponível em: <http://dl.acm.org/citation.cfm?id=2337159.2337161>.

LIU, S. et al. Flikker: Saving DRAM Refresh-power Through Critical Data Partitioning. **SIGPLAN Not.**, ACM, New York, NY, USA, v. 46, n. 3, p. 213–224, mar. 2011. ISSN 0362-1340. Disponível em: <http://doi.acm.org/10.1145/1961296.1950391>.

Micron Technology Inc. 4Gb: x16, x32 Mobile LPDDR3 SDRAM. July 2013.

QURESHI, M. K. et al. AVATAR: A Variable-Retention-Time (VRT) Aware Refresh for DRAM Systems. **Memory**, v. 2, n. 4Gb, p. 20, 2015.

SADRI, M. et al. Energy Optimization in 3D MPSoCs with Wide-I/O DRAM Using Temperature Variation Aware Bank-Wise Refresh. In: **Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014**. [S.l.: s.n.], 2014. p. 1–4.

SRIDHAR, A. et al. 3D-ICE: A Compact Thermal Model for Early-Stage Design of Liquid-Cooled ICs. **Computers, IEEE Transactions on**, v. 63, n. 10, p. 2576–2589, Oct 2014. ISSN 0018-9340.

TEMAN, A. et al. Energy Versus Data Integrity Trade-offs in Embedded High-density Logic Compatible Dynamic Memories. In: **Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition**. San Jose, CA, USA: EDA Consortium, 2015. (DATE '15), p. 489–494. ISBN 978-3-9815370-4-8. Disponível em: <http://dl.acm.org/citation.cfm?id=2755753.2755864>.

WEIS, C. et al. Retention Time Measurements and Modelling of Bit Error Rates of WIDE I/O DRAM in MPSoCs. In: **Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition**. San Jose, CA, USA: EDA Consortium, 2015. (DATE '15), p. 495–500. ISBN 978-3-9815370-4-8. Disponível em: <http://dl.acm.org/citation.cfm?id=2755753.2755865>.

WULF, W. A.; MCKEE, S. A. Hitting the Memory Wall: Implications of the Obvious. **SIGARCH Comput. Archit. News**, ACM, New York, NY, USA, v. 23, n. 1, p. 20–24, mar. 1995. ISSN 0163-5964. Disponível em: <http://doi.acm.org/10.1145/216585.216588>.

ZHANG, T. et al. CREAM: a Concurrent-Refresh-Aware DRAM Memory Architecture. In: IEEE. **High Performance Computer Architecture (HPCA), 2014 IEEE 20th International Symposium on**. [S.l.], 2014. p. 368–379.

Appendices

AppendixA C++ FILES

```
#ifndef TEMPERATURE_CONTROLLER_H_
#define TEMPERATURE_CONTROLLER_H_

#include <systemc.h>
#include <iostream>
#include <string>
#include <fstream>

#include "../common/DebugManager.h"
#include "../common/Utils.h"
#include "../controller/core/configuration/Configuration.h"

#ifdef THERMALSIM
#include "IceWrapper.h"
#endif

SC_MODULE(TemperatureController) {
public:
    static inline TemperatureController &getInstance()
    {
        static TemperatureController temperaturectrl("TemperatureController");
        return temperaturectrl;
    }

    SC_CTOR(TemperatureController)
    {
        temperatureScale = Configuration::getInstance().temperatureSim.TemperatureScale;

        dynamicTempSimEnabled = Configuration::getInstance().ThermalSimulation;

        staticTemperature =
            Configuration::getInstance().temperatureSim.StaticTemperatureDefaultValue;

        if (dynamicTempSimEnabled == true) {
#ifdef THERMALSIM
            // Connect to the server
            std::string ip = Configuration::getInstance().temperatureSim.IceServerIp;
            unsigned int port =
                Configuration::getInstance().temperatureSim.IceServerPort;
            thermalSimulation = new IceWrapper(ip, port);
            printDebugMessage("Dynamic temperature simulation. Server @ " + ip + ":" +
                std::to_string(port));
#else
            SC_REPORT_FATAL(name(), "DRAMSys was build without support to dynamic
                temperature simulation. Check the README file for further details.");
#endif
            // Initial power dissipation values (got from config)
            currentPowerValues =
                Configuration::getInstance().temperatureSim.powerInitialValues;
```

```
            lastPowerValues = currentPowerValues;

            // Substantial changes in power will trigger adjustments in the simulaiton
                period. Get the thresholds from config.
            powerThresholds =
                Configuration::getInstance().temperatureSim.powerThresholds;
            decreaseSimPeriod = false;
            periodAdjustFactor =
                Configuration::getInstance().temperatureSim.SimPeriodAdjustFactor;
            nPowStableCyclesToIncreasePeriod =
                Configuration::getInstance().temperatureSim.NPowStableCyclesToIncreasePeriod;
            cyclesSinceLastPeriodAdjust = 0;

            // Get the target period for the thermal simulation from config.
            targetPeriod = Configuration::getInstance().temperatureSim.ThermalSimPeriod;
            period = targetPeriod;
            t_unit = Configuration::getInstance().temperatureSim.ThermalSimUnit;

            genTempMap =
                Configuration::getInstance().temperatureSim.GenerateTemperatureMap;
            temperatureMapFile = "temperature_map";
            std::system("rm -f temperature_map*");

            genPowerMap = Configuration::getInstance().temperatureSim.GeneratePowerMap;
            powerMapFile = "power_map";
            std::system("rm -f power_map*");

            SC_THREAD(temperatureThread);
        } else {
            printDebugMessage("Static temperature simulation. Temperature set to " +
                std::to_string(staticTemperature));
        }
    }

    double getTemperature(int deviceId, float currentPower);

private:
    std::string temperatureScale;
    double temperatureConvert(double tKelvin);

    double staticTemperature;

    bool dynamicTempSimEnabled;

#ifdef THERMALSIM
    IceWrapper *thermalSimulation;
#endif
    std::vector<float> temperaturesBuffer;
    std::vector<float> temperatureValues;

    std::vector<float> currentPowerValues;
    std::vector<float> lastPowerValues;
```

```
    std::vector<float> powerThresholds;

    double targetPeriod;
    double period;
    enum sc_time_unit t_unit;
    void temperatureThread();
    void updateTemperatures();
    double adjustThermalSimPeriod();
    void checkPowerThreshold(int deviceId);
    bool decreaseSimPeriod;
    unsigned int periodAdjustFactor;
    unsigned int cyclesSinceLastPeriodAdjust;
    unsigned int nPowStableCyclesToIncreasePeriod;

    bool genTempMap;
    std::string temperatureMapFile;
    bool genPowerMap;
    std::string powerMapFile;

    void printDebugMessage(std::string message);
};

#endif /* TEMPERATURE_CONTROLLER_H_ */
```

```
#include <cmath>

#include "TemperatureController.h"
#include "../controller/core/configuration/Configuration.h"

double TemperatureController::temperatureConvert(double tKelvin)
{
    if (temperatureScale == "Celsius") {
        return tKelvin - 273.15;
    } else if (temperatureScale == "Fahrenheit") {
        return (tKelvin - 273.15) * 1.8 + 32;
    }

    return tKelvin;
}

double TemperatureController::getTemperature(int deviceId, float currentPower)
{
    printDebugMessage("Temperature requested by device " + std::to_string(deviceId) + "
        current power is " + std::to_string(currentPower));

    if (dynamicTempSimEnabled == true) {
        currentPowerValues.at(deviceId) = currentPower;
        checkPowerThreshold(deviceId);

        if (temperatureValues.empty())
            return temperatureConvert(staticTemperature + 273.15);
```

```
        return temperatureConvert(temperatureValues.at(deviceId));
    } else {
        printDebugMessage("Temperature is " + std::to_string(staticTemperature));
        return staticTemperature;
    }
}

void TemperatureController::updateTemperatures()
{
#ifdef THERMALSIM
    thermalSimulation->sendPowerValues(&currentPowerValues);
    thermalSimulation->simulate();
    thermalSimulation->getTemperature(temperaturesBuffer, TDICE_OUTPUT_INSTANT_SLOT,
        TDICE_OUTPUT_TYPE_TFLPEL, TDICE_OUTPUT_QUANTITY_AVERAGE);

    std::string mapfile;
    sc_time ts = sc_time_stamp();
    if (genTempMap == true) {
        mapfile = temperatureMapFile + "_" + std::to_string(ts.to_default_time_units())
            + ".txt";
        thermalSimulation->getTemperatureMap(mapfile);
    }
    if (genPowerMap == true) {
        mapfile = powerMapFile + "_" + std::to_string(ts.to_default_time_units()) +
            ".txt";
        thermalSimulation->getPowerMap(mapfile);
    }
#endif
    // Save values just obtained for posterior use
    temperatureValues = temperaturesBuffer;
    // Clear the buffer, otherwise it will grow every request
    temperaturesBuffer.clear();
}

void TemperatureController::checkPowerThreshold(int deviceId)
{
    if (std::abs(lastPowerValues.at(deviceId) - currentPowerValues.at(deviceId)) >
        powerThresholds.at(deviceId)) {
        decreaseSimPeriod = true;
    }
    lastPowerValues.at(deviceId) = currentPowerValues.at(deviceId);
}

double TemperatureController::adjustThermalSimPeriod()
{
    // Temperature Simulation Period Dynamic Adjustment
    //
    // 1. Adjustment is requierd when:
    //
    // 1.1. The power dissipation of one or more devices change considerably
    // (exceeds the configured threshold for that device in any direction,
    // i.e. increases or decreases substantially) during the current
```

```
    // simulaiton period.
    //
    // 1.1.1. The simulation period will be reduced by a factor of 'n' so the
    // simulation occurs 'n' times more often.
    //
    // 1.1.2. The step 1.1.1 will be repeated until the point that there are
    // no sustantial changes in power dissipation between two consecutive
    // executions of the thermal simulation, i.e. all changes for all devices
    // are less than the configured threshold.
    //
    // 1.2. The current simulation period differs from the target period
    // defined in the configuration by the user.
    //
    // 1.2.1 Provided a scenario in which power dissipation changes do not
    // exceed the thresholds, the situation period will be kept for a number
    // of simulation cycles 'nc' and after 'nc' the period will be increased
    // again in steps of 'n/2' until it achieves the desired value given by
    // configuration or the described in 1.1 occurs.

    if (decreaseSimPeriod == true) {
        period = period / periodAdjustFactor;
        cyclesSinceLastPeriodAdjust = 0;
        decreaseSimPeriod = false;
        printDebugMessage("Thermal Simulation period reduced to " +
            std::to_string(period) + ". Target is " + std::to_string(targetPeriod));
    } else {
        if (period != targetPeriod) {
            cyclesSinceLastPeriodAdjust++;
            if (cyclesSinceLastPeriodAdjust >= nPowStableCyclesToIncreasePeriod) {
                cyclesSinceLastPeriodAdjust = 0;
                period = period * (periodAdjustFactor / 2);
                if (period > targetPeriod)
                    period = targetPeriod;
                printDebugMessage("Thermal Simulation period increased to " +
                    std::to_string(period) + ". Target is " +
                    std::to_string(targetPeriod));
            }
        }
    }

    return period;
}

void TemperatureController::temperatureThread()
{
    while (true) {
        updateTemperatures();
        double p = adjustThermalSimPeriod();

        int i = 0;
        for (auto t : temperatureValues) {
            printDebugMessage("Temperature[" + std::to_string(i++) + "] is " +
```

```
                std::to_string(t));
        }
        printDebugMessage("Thermal simulation period is " + std::to_string(p));

        wait(sc_time(p, t_unit));
    }
}

void TemperatureController::printDebugMessage(std::string message)
{
    DebugManager::getInstance().printDebugMessage(name(), message);
}
```

AppendixB 3D-ICE PROJECT FILES

A 3D-ICE project consists of writing a stack descriptor file and one or more floorplan files.

B.1 Stack File

The stack description file is a netlist that specifies all the physical and geometrical properties of the 3D-IC for the simulation.

3d_stack.stk

```
material SILICON :
    thermal conductivity 1.30e-4 ;
    volumetric heat capacity 1.628e-12 ;

material BEOL :
    thermal conductivity 2.25e-6 ;
    volumetric heat capacity 2.175e-12 ;

material SIO2 :
    thermal conductivity 1.38e-6 ;
    volumetric heat capacity 1.62e-12 ;

material COPPER :
    thermal conductivity 4.01e-04 ;
    volumetric heat capacity 3.37e-12 ;

heat sink :
    sink height 1e03, area 100e06, material COPPER ;
    spreader height 0.5e03, area 70e06, material SILICON ;
    heat transfer coefficient 1.3e-09 ;
    ambient temperature 318.15 ;

layer PCB :
    height 10 ;
    material BEOL ;

die DRAM_TOP :
    layer 58.5 SILICON ;
```

```
    source 2 SILICON ;
    layer 1.5 BEOL ;

die DRAM :
    layer 15 SIO2 ;
    layer 1.5 BEOL ;
    source 2 SILICON ;
    layer 58.5 SILICON ;

die MPSOC :
    layer 27 SIO2 ;
    layer 10 BEOL ;
    source 2 SILICON ;
    layer 50 SILICON ;

dimensions :
   chip length 6100, width 10600 ;
   cell length  100, width   100 ;

stack:
    die DRAM_DIE_4 DRAM_TOP floorplan "./mem.flp" ;
    die DRAM_DIE_3 DRAM     floorplan "./mem.flp" ;
    die DRAM_DIE_2 DRAM     floorplan "./mem.flp" ;
    die DRAM_DIE_1 DRAM     floorplan "./mem.flp" ;
    die MPSOC_DIE  MPSOC    floorplan "./core.flp" ;
    layer CONN_TO_PCB PCB ;

solver:
   transient step 0.01, slot 0.05 ;
   initial temperature 300.0 ;

output:
   TflpelDRAM_DIE_4.channel0 , "d4_ch0.txt" , average  , slot ;
   TflpelDRAM_DIE_4.channel1 , "d4_ch1.txt" , average  , slot ;
   TflpelDRAM_DIE_4.channel2 , "d4_ch2.txt" , average  , slot ;
   TflpelDRAM_DIE_4.channel3 , "d4_ch3.txt" , average  , slot ;
   Tmap  DRAM_DIE_4, "d4_tmap.txt", slot ;
   Pmap  DRAM_DIE_4, "d4_pmap.txt", slot ;

   TflpelDRAM_DIE_3.channel0 , "d3_ch0.txt" , average  , slot ;
   TflpelDRAM_DIE_3.channel1 , "d3_ch1.txt" , average  , slot ;
```

```
TflpelDRAM_DIE_3.channel2 , "d3_ch2.txt" , average  , slot ;
TflpelDRAM_DIE_3.channel3 , "d3_ch3.txt" , average  , slot ;
Tmap  DRAM_DIE_3, "d3_tmap.txt", slot ;
Pmap  DRAM_DIE_3, "d3_pmap.txt", slot ;

TflpelDRAM_DIE_2.channel0 , "d2_ch0.txt" , average  , slot ;
TflpelDRAM_DIE_2.channel1 , "d2_ch1.txt" , average  , slot ;
TflpelDRAM_DIE_2.channel2 , "d2_ch2.txt" , average  , slot ;
TflpelDRAM_DIE_2.channel3 , "d2_ch3.txt" , average  , slot ;
Tmap  DRAM_DIE_2, "d2_tmap.txt", slot ;
Pmap  DRAM_DIE_2, "d2_pmap.txt", slot ;

TflpelDRAM_DIE_1.channel0 , "d1_ch0.txt" , average  , slot ;
TflpelDRAM_DIE_1.channel1 , "d1_ch1.txt" , average  , slot ;
TflpelDRAM_DIE_1.channel2 , "d1_ch2.txt" , average  , slot ;
TflpelDRAM_DIE_1.channel3 , "d1_ch3.txt" , average  , slot ;
Tmap  DRAM_DIE_1, "d1_tmap.txt", slot ;
Pmap  DRAM_DIE_1, "d1_pmap.txt", slot ;
```

B.2 Floorplan Files

Every die in the stack must be related to a floorplan file, which provides the heat sources within the die for the simulation. Floorplan files provide information about functional blocks and their positions inside a die.

mem.flp

```
channel0:
    position   150,  100;
    dimension  2600, 5200;

channel1:
    position   3350,  100;
    dimension  2600, 5200;

channel2:
    position   150, 5300;
    dimension  2600, 5200;
```

```
channel3:
    position   3350, 5300;
    dimension  2600, 5200;
```

---

core.flp

```
CPUs :

  position         0,    0 ;
  dimension     2750, 4300 ;

GPU :

  position      3350,    0 ;
  dimension     2750, 4000 ;

BASEBAND1 :

  position      4250, 4000 ;
  dimension     1850, 3300 ;

BASEBAND2 :

  position      3350, 7300 ;
  dimension     2750, 3300 ;

LLCACHE :

  position         0, 4300 ;
  dimension     1900, 3000 ;

DRAMCTRL1 :

  position      1900, 4300 ;
  dimension      850, 3000 ;

DRAMCTRL2 :
```

```
  position     3350, 4000 ;
  dimension     900, 3300 ;

TSVS :

  position     2750, 2300 ;
  dimension     600, 6000 ;

ACELLERATORS :

  position        0, 7300 ;
  dimension    2750, 3300 ;
```

AppendixC PUBLICATION

C.1 Omitting Refresh: A Case Study for Commodity and Wide I/O DRAMs

Published in: MEMSYS ’15 Proceedings of the 2015 International Symposium on Memory Systems. Pages 85-91.

Omitting Refresh

A Case Study for Commodity and Wide I/O DRAMs

Matthias Jung, Éder Zulian, Deepak M. Mathew, Matthias Herrmann,
Christian Brugger, Christian Weis, Norbert Wehn
Microelectronic Systems Design Research Group
University of Kaiserslautern
67663 Kaiserslautern Germany
{jungma,zulian,deepak,herrmann,brugger,weis,wehn}@eit.uni-kl.de

ABSTRACT

Dynamic Random Access Memories (DRAM) have a big impact on performance and contribute significantly to the total power consumption in systems ranging from mobile devices to servers. Up to half of the power consumption of future high density DRAM devices will be caused by refresh commands. Moreover, not only the refresh rate does depend on the device capacity, but it strongly depends on the temperature as well. In case of 3D integration of MPSoCs with Wide I/O DRAMs the power density and thermal dissipation are increased dramatically. Hence, in 3D-DRAM even more DRAM refresh operations are required. To master these challenges, clever DRAM refresh strategies are mandatory either on hardware or on software level using new or already available infrastructures and implementations, such as *Partial Array Self Refresh* (PASR) or *Temperature Compensated Self Refresh* (TCSR).

In this paper, we show that for dedicated applications refresh can be disabled completely without or with negligible impact on the application performance. This is possible if it is assured that either the lifetime of the data is shorter than the currently required DRAM refresh period or if the application can tolerate bit errors to some degree in a given time window.

CCS Concepts

•Computer systems organization → Reliability; Processors and memory architectures;

Keywords

DRAM, Retention, Refresh, 3D-Integration, Reliability, Approximate Computing

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

MEMSYS '15 Washington DC, USA
© 2015 ACM. ISBN 123-4567-24-567/08/06…$15.00
DOI: 10.475/123_4

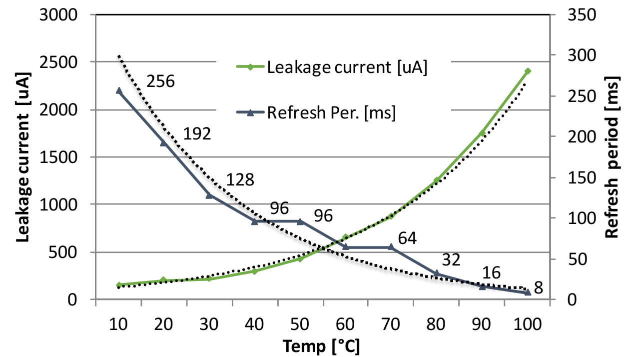


Figure 1: Leakage Current and Required Refresh Periods at Different Temperatures. [28, 31, 39]

1. INTRODUCTION

Memory energy consumption has become a significant concern in mobile computing, servers and high-performance computing platforms. There are applications, such as used in the GreenWave computing platform [25], in which 49% of the total power consumption has to be attributed to DRAMs.

DRAMs must be refreshed regularly due to their charge based bit storage property (capacitor). The retention time of a DRAM cell is defined as the amount of time that a DRAM cell can safely retain data without being refreshed [18]. This DRAM refresh operation must be issued periodically and causes both performance degradation and increased energy consumption. Liu et al. [29] predicted that 40% to 50% of the power consumption of future DRAM devices will be caused by refresh commands (compare Table 1).

Moreover, 3D integrated DRAMs like Wide I/O or HMC worsen the temperature behavior. Due to the much increased leakage at the cells the refresh frequency needs to be adjusted accordingly to avoid retention errors [39] as shown in Figure 1. Thus, the efficient utilization of the available DRAM bandwidth and the efficient usage of DRAM power-down modes and clever refresh techniques are major contributions to a high energy efficiency of DRAM subsystems and the computing system in which they are integrated [22].

Recently, a lot of research has been done (see Section 2) to minimize the number of refresh operations by means of different techniques. However, it was not considered that for some safety uncritical applications DRAM *Auto-Refresh* (AR) can be disabled completely without or with negligible

Ref.	Impact on	DRAM Device Size (Gb)					
		2	4	8	16	32	64
[29]	Avg. Power	10%	15%	17%	24%	34%	47%
[5]	Energy	10%	14%	16%	18%	25%	?
[29]	CPU Perf.	-5%	-7%	-9%	-14%	-26%	-46%
[5]	CPU IPC	-5%	-8%	-11%	-18%	-36%	?
[5]	CPU Latency	4%	6%	8%	13%	24%	?

Table 1: Impact of Refresh on Future DRAM Devices

impact on the application performance, called *Omit Refresh* (OR) strategy. This strategy is feasible, if it is assured that either the lifetime of the data is shorter than the currently required DRAM refresh period or the application can tolerate bit errors to some degree in a given time window.

Similar to Flikker [30] we propose a reliable and an unreliable memory region. However, we go one step further: Instead of lowering the refresh rate in the unreliable region we apply the OR-Strategy.

As shown in Figure 3 the lowest 3D-DRAM layer has the highest average temperature. Hence, this layer requires a higher refresh rate than the rest of the DRAM stack [39]. Consequently, the lowest layer is a perfect candidate for applying the OR-Strategy by tolerating an unreliable memory layer in order to save refresh power. While the upper reliable part of the stack is refreshed the unreliable region can be accessed exclusively, which is managed by the DRAM controller. In combination with a compiler that can handle reliable and unreliable data types [40, 30] OR can be used even on CPU based applications.

A retention error aware DRAM model is key to analyze the impact of lower refresh rates or even disabled refresh (OR) on executed applications. Especially for error resilient applications, this can be exploited to save energy [16]. We measured [46] the retention times of WIDE I/O [14, 24] and DDR3 DRAM devices using different data pattern. We observed *data pattern dependencies* (DPD), shown in Figure 2 and *variable retention times* (VRT), similar to the study in [28] for commodity DRAMs. With this data we created a DRAM retention time error model [46] that can be used to study energy vs. reliability trade-offs. Obviously, the aspect ratio of the DRAM cell capacitor is increasing by technology scaling, while the dimensions of the select transistor are decreasing. Thus, the leakage power and the reported soft error rates are higher for newer DRAM device technologies [34]. We considered this effect in our executed experiments.

In this paper, we apply the OR-Strategy on different applications that range from image processing to big data accelerators. This highlights the feasibility of this approach and shows the impact on each selected application.

2. RELATED WORK

DRAM refresh has become one of the most important research topics in the DRAM community. This section gives a survey of the important state-of-the-art techniques to reduce the number of refreshes. A more detailed recent survey can be found in [5].

Reliability vs. Refresh.

The *REVA* [1] refresh scheme can be used in dedicated video applications. It refreshes only the important *region of interest* (ROI) in a video frame. *Flikker* [30] reduces the number of refreshes by partitioning a DRAM bank in a

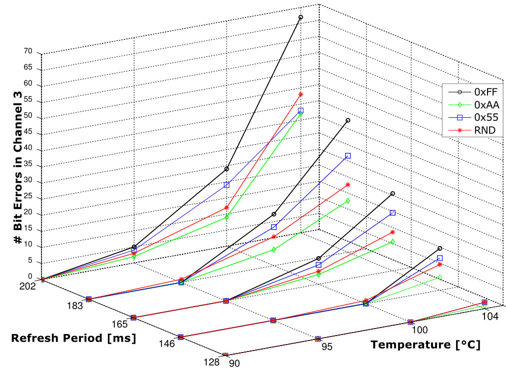


Figure 2: Measurement Results with Fixed Refresh Periods

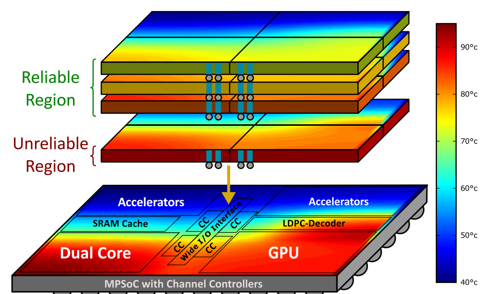


Figure 3: Wide I/O DRAM on an MPSoC, simulated with gem5 [8] DRAMSys [21], DRAMPower [11] and 3D-ICE [41]

critical and non-critical region. The non-critical region will be refreshed with a lower refresh rate.

Selective Refresh.

PARIS [2], *DTail* [13] and *ESKIMO* [20] exclude rows that do not store useful data from being refreshed. *Smart Refresh* [15] refreshes only rows that have not been accessed recently. *CREAM* [47], [39] and [12] show per-bank and per-subarray refresh techniques. The authors of [37] issue refresh commands according to the stored data values. *Flexible Auto-Refresh* [7] shows a realistic implementation of a flexible and row selective refresh.

Thermal aware Refresh.

A temperature variation aware bank-wise refresh for 3D-DRAMs is presented in [39]. [22] presents how DRAM self-refresh can be approached in a staggered way.

Refresh Scheduling.

The idea of postponing DRAM refresh into self-refresh phases is presented in [6]. The authors of [35] show how to use JEDEC's DDR4 fine granularity refresh (FGR) efficiently, [43] presents an enhancement (EFGR). In [42], *Elastic Refresh*, an approach that adapts the refresh behavior according to the current workload is shown. *Refresh Pausing* [36] is a technique that allows to pause a current refresh operation to serve a DRAM access. To make refresh predictable for real-time application it can be triggered from software level, as shown in [4].

Retention Aware Refresh.

RAPID [44] is a software approach that allocates longer-retention pages before shorter-retention pages. The refresh rate is adjusted according to the shortest-retention page used. *RAIDR* [29] forms rows into retention time bins that are refreshed with different rates. *RIO* [2], a software approach, excludes weak rows from usage. *SECRET* [27] and [48] store the content of weak cells in a separate region in the DRAM by using an error correction pointer mechanism. The authors of [17] lower the overall refresh rate and refresh weak rows selectively by issuing single `ACT-PRE` commands. In *ProactiveDRAM* [45] a list of weak rows exists in the DRAM device and the DRAM itself decides at each refresh command if the row should be refreshed. *AVATAR* [38] tries to overcome VRT issues by combining an online ECC detection with row selective refresh.

Overall, it is very difficult to create a valid list of weak cells for DRAMs, as they experience VRTs and DPDs for their retention times. Moreover, in [46] it is shown that the temperature has a strong effect on VRT. Hence, it is infeasible during startup of a system to determine an exact list of weak cells that considers all parameters, such as temperature, retention time and DPD. Thus, it is preferable to omit refresh in the unreliable regions and use standard guard banding in the reliable regions for commodity devices (see Figure 1) and advanced margins for 3D stacked devices, as it was analyzed in [26]. Moreover, in the reliable part of the stack the previously mentioned state-of-the-art techniques e.g. temperature aware bank-wise refresh [39] can still be applied.

3. EXPERIMENTS

To prove our assumptions and to demonstrate the feasibility of the OR-Strategy we conduct three experiments. The first study investigates the similarity in large complex graphs while considering data that need no refresh. In the second experiment we evaluate the input data resilience of an LDPC decoder for wireless baseband processing. Finally, in the last application we execute an image processing task on a XILINX FPGA to obtain real measurement data of the applied OR-Strategy.

3.1 Big Data

Complex graphs are at the heart of today's big data challenges like recommendation systems, customer behavior modeling, or incident detection systems. One reoccurring task in these fields is the extraction of network motifs, reoccurring and statistically significant subgraphs. In this example we show the influence of OR to a precisely tailored embedded architecture [10] for computing similarities based on one special network motif, the co-occurrence. It is based on efficient and scalable building blocks that exploit well-tuned algorithmic refinements and an optimized graph data representation approach.

In [10] the following method is considered for calculating similarities using the co-occurrence: In the example given in Figure 4, the similarity between *you* and *Liam* is based on the number of common friends, the so-called co-occurrence: $coocc(you, Liam) = 3$. The question arises whether the number three is significant. Assume *you* and *Liam* have thousands of friends, versus *you* have only three. For this random graphs are created based on the same degree sequence and the premise that nodes have no similarities.

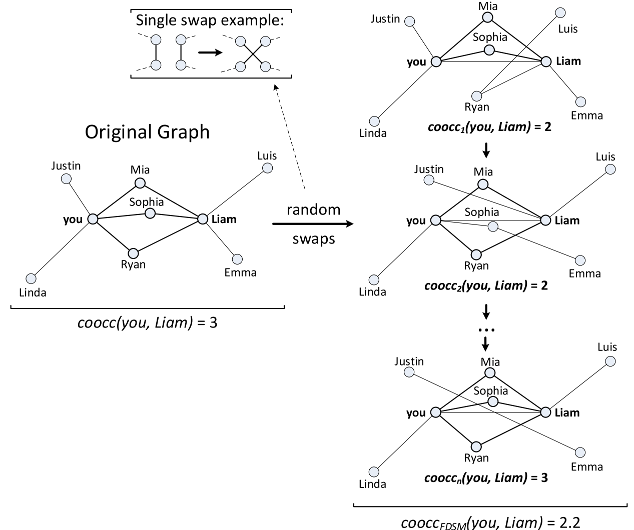


Figure 4: Co-Occurrence and Swapping in a Graph

To get the random graphs a sufficient number of pairs of edges are swapped, drawn uniformly at random, if and only if no multiple edges would arise due to the swap. This generates independent graphs with the same degree sequence, see Figure 4. Generating many of such graphs the expected co-occurrence can be calculated. That information can judge how significant the similarity in the original graph is.

However, for large graphs this is in general a very time and memory consuming job on standard computing clusters. Therefore, this application has been implemented as an ASIC [10], where the graph is stored in the DRAM as sparse adjacency matrix.

There is a reason to believe that a certain amount of errors in the large graph matrix will not influence the quality of the result. For instance, when this ASIC is used as accelerator in an MPSoC similar to Figure 3, the graph matrix can be stored in the unreliable region, whereas important control variables are stored in the reliable region of the memory.

To show the feasibility of this idea we simulated similarity measures for the Netflix dataset [19, 3] in combination with our retention time error model [46]. Netflix, a commercial video streaming service, has released 100,480,507 user ratings for all of their 17,700 movies from 480,189 users. For this simulation we use a data subset with 20,000 users.

The dataset is stored in both list and matrix form. The list allows to select edges uniformly at random, while the matrix allows to check for conflicting edges in constant time. The datasets for recommendation are typically sparse, as each user only rate a few movies. In our case, the matrix has a sparsity of 1% and a size of 337 Mbit. We store the edge in the matrix as zeros. In that case, retention errors (transitions from '1' to '0', as shown in [46] for a Wide I/O device) can only introduce new edges, but not delete existing edges. This is important for the algorithm to not introduce duplicate edges in the swap step. The list has a size of 99 Mbit and is stored in the reliable region.

In each co-occurrence calculation the complete dataset is touched at least one time and this is equivalent to a complete refresh. The required lifetime of the data in the DRAM

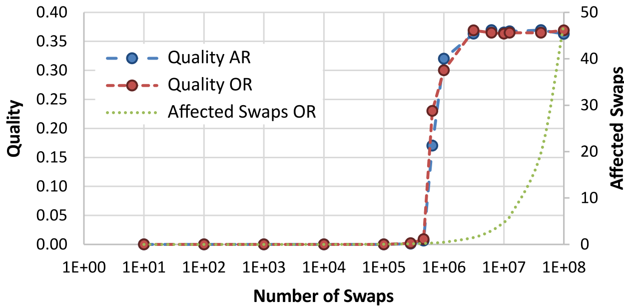


Figure 5: Netflix - 20k Users

between two co-occurrence calculations (swap-phase) is approximately $244ms$. In this time frame retention errors can occur. This is modeled with the model of [46] under a worst case temperature of $90°C$.

To check the impact of omitting refresh, we compare the quality of the output of the Link Assessment algorithm with normal *Auto-Refresh* (AR) and OR. The quality is assessed based on a ground-truth, a set of human selected movie pairs of highly similar movies, as described in [9]. In our case, it is assembled based on all permutations of movie sequels. The higher these pairs are ranked in the output, the higher the quality. In [9] so called phase-transitions have been observed for key variables like the number of swaps. Figure 5 shows this phase transition with AR and OR. Important features are the position of the transition and the final level, the higher and more left the better. Both curves for AR and OR lie on each other, apart from statistical variations, no difference in quality can therefore be observed. In green the number of affected swaps are shown also in Figure 5. Of 10^8 swaps only 48 swaps are affected by retention errors on average. This explains the negligible impact of OR on the resulting quality.

3.2 Channel Decoding

In digital communication systems channel coding can be considered as key-technology to achieve reliable communication. Thus, it plays an integral role in all important modern communication standards.

One of the currently most sophisticated concepts of channel-coding is *Low-Density Parity Check* (LDPC) coding. This circumstance is also reflected in its adoption to recent wireless communication standards like WiGig and WiFi. The strength of LDPC coding is its iterative decoding algorithm that allows the utilization of information on the reliability of each bit, so-called soft-information. As for most iterative decoders, the number of iterations is limited to a maximum value and can be adjusted on demand without deterioration of the result. As a consequence, the amount of iterations the decoder performs strongly depends on the degree of impairment of the currently received signal. This feature drastically reduces the energy consumption of the decoder.

In this experiment we determine the communications performance of an LDPC decoder that is used in an MPSoC as shown in Figure 3. The unreliable region of the DRAM is used for buffering the input soft-information. The communication system simulation consists of a source that generates random binary data, a WiGig LDPC channel encoder, a *Binary phase-shift keying* (BPSK) modulator that maps

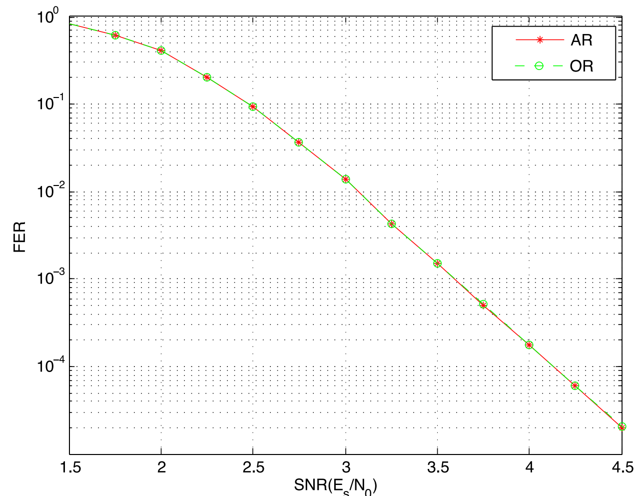


Figure 6: Communications performance of two WiGig decoders with and without OR

bits to the signal values $+1$ and -1 respectively, a channel that adds random gaussian-distributed noise to our signal, and the component's corresponding counterparts on receiver side. The channel code is a rate $13/16$ WiGig code of length $N = 672$. Communications performance is determined by Monte-Carlo simulation of 10^7 frames for a given *Signal-to-Noise-Ratio* (SNR) and keeping track of the corresponding *Frame-Error-Rate* (FER) after the channel decoding.

The results are shown in Figure 6. We can clearly see that although some of the bits in the input memory of the decoder have flipped (due to OR), the FER is the same as with enabled AR. This shows that the influence of the retention errors in the DRAM on the FER after decoding is much smaller than the influence of the errors induced by the channel. This is known as inherent error resilience of channel coding [16].

Figure 7 shows that for a worst case temperature of $100°C$ and a refresh rate for AR of $t_{REF} = 8ms$ about 12% of the total energy can be saved by disabling the refresh (OR).

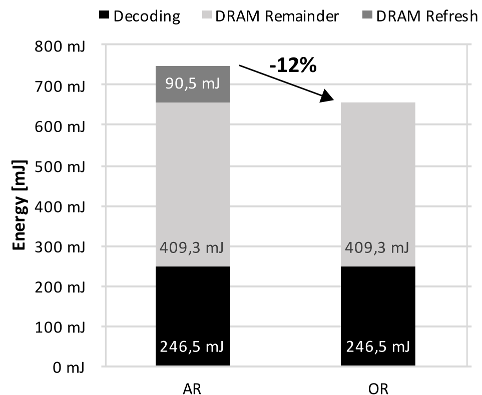


Figure 7: Energy Breakdown of WiGig Decoder and DRAM for 10^7 Frames at SNR=3.5dB

3.3 Image Processing

In this section we demonstrate that our OR-Strategy can be beneficial for even today's commodity DDR3 DRAMs by using an image processing FPGA application in which the lifetime of the image in the memory is less than the DDR3 refresh interval (t_{REF}).

In this application, an image stream from a camera is rotated by 90 degrees using the DRAM. Each image (frame) consists of 4,525,056 pixels with 48 bits per pixel. An image is stored into the DRAM at consecutive address locations and it is retrieved back from specific locations in a way such that the image is rotated. This application has a real-time constraint (deadline) of $8.6ms$ on the frame rate. As shown in [4] DRAM refresh is an unpredictable factor for worst-case execution (WCET) analysis. Therefore, it makes a lot of sense to disable refresh for applications with real-time deadlines smaller than t_{REF}.

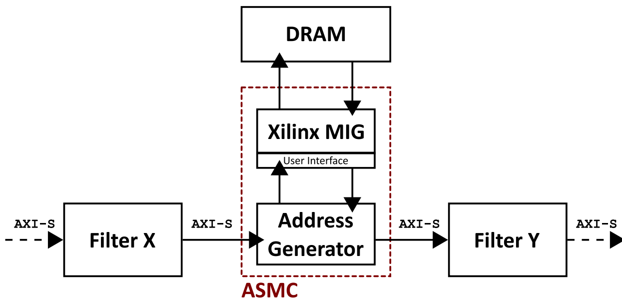


Figure 8: Image Processing Pipeline with Application Specific Memory Controller (ASMC)

Figure 8 shows a part of the image processing pipeline realized on a Xilinx Zynq ZC706 FPGA Evaluation Board [32]. The board consists of a 1 GB, x64 DDR3 SO-DIMM connected to the *Programmable Logic* (PL) side of the FPGA. We use a customized *Application Specific Memory Controller* (ASMC) based on the Xilinx *Memory Interface Generator* (MIG) [33]. This controller has a lean front-end, which consists of an AXI-Stream interface and an address generator. The address generator calculates the addresses to store and retrieve images, since AXI-Stream has no address information. It internally consists of a scrambler, which transforms [23] the bits of the generated addresses to achieve an optimized access pattern for this particular application.

The FPGA evaluation board has four different voltage rails controlled by Texas Instruments UCD90120A power supply sequencer and monitor [32]. We use the TI USB Interface adapter along with the TI Fusion Digital Power graphical user interface to record the voltage and current levels of the rails. The measurements are made only on the voltage rail 3, since this rail solely powers the DDR3 SO-DIMM and the PHY I/Os of the memory controller.

We conducted two different experiments: auto refresh enabled (AR) and with refresh disabled completely (OR). In both cases we sent a fixed number of frames (611,620) to the DRAM and monitored the execution time as well as the rail voltage and currents. With refresh enabled, the MIG issues an *Auto-Refresh* (AR) command to the DRAM every $7.8\mu s$. These refreshes interfere with the normal `READ`/`WRITE` command execution of the MIG memory controller and therefore increases the time to process the complete set of frames. However, in this application each frame is stored in the memory only for less than $9ms$. Since this is much less than the $64ms$ refresh period of the DDR3 DRAM at normal temperature and the DRAM is exclusively used for the rotation task, we disable the refresh mechanism inside the MIG controller (by modifying the Verilog source code) and conducted the second experiment.

In our test setup all frames are checked for data corruption and no bit error occurred. As shown in Figure 9 the normal runtime to process 611,620 frames is $43.7min$. This time is reduced by applying OR by $62s$. The required energy can be reduced by 1.6%. Even though the savings in time and energy are not significant, this experiment shows the feasibility of the OR approach for a real device. As soon as more refresh operations are required for higher capacity commodity DRAM devices OR will be a perfect option for many applications.

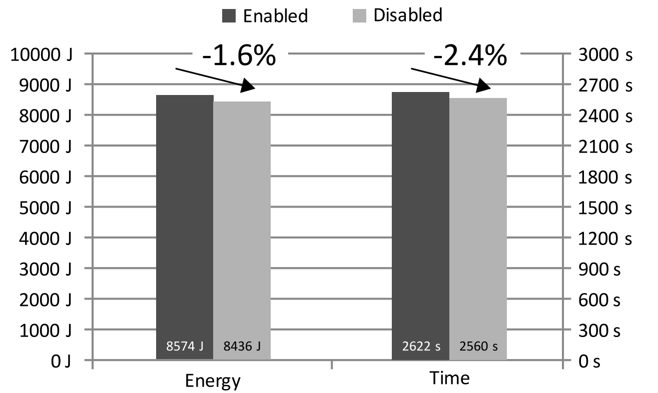


Figure 9: Results of enabled (AR) and disabled refresh (OR) for an image processing task

4. CONCLUSION

In this paper, we investigated the feasibility of the OR-Strategy for three different applications. We demonstrated that switching-off the refresh during co-occurrence calculation in graph processing has a negligible impact on the resulting quality of the prediction. We presented the influence on the QoS of an LDPC decoder for wireless baseband processing, while applying OR on its input data buffer. We found a very small increase in iterations with no impact on the FER. Additionally, we executed an image processing task on a XILINX FPGA with OR applied on the attached DDR3 DRAMs. This application was running error free for a huge number of frames and decreased the execution time and energy consumption. Overall, we have shown that the OR-Strategy is sensible to use in these three applications and in others that are able to tolerate errors to some degree. Especially, for future high-density DRAMs the OR-Strategy will be very beneficial as it will recover the available bandwidth and reduce the average power consumption.

5. ACKNOWLEDGMENTS

The authors thank the company Synopsys for their great support. This work was partially funded by the German Research Foundation (DFG) as part of the priority program Dependable Embedded Systems SPP 1500 (http://spp1500.itec.kit.edu) and the DFG grant no. WE2442/10-1.

6. REFERENCES

[1] S. Advani, N. Chandramoorthy, K. Swaminathan, K. Irick, Y. Cho, J. Sampson, and V. Narayanan. Refresh Enabled Video Analytics (REVA): Implications on power and performance of DRAM supported embedded visual systems. In *Computer Design (ICCD), 2014 32nd IEEE International Conference on*, pages 501–504, Oct 2014.

[2] S. Baek, S. Cho, and R. Melhem. Refresh now and then. *Computers, IEEE Transactions on*, 63(12):3114–3126, 2014.

[3] J. Bennett and S. Lanning. The netflix prize. In *Proceedings of KDD cup and workshop*, volume 2007, page 35, 2007.

[4] B. Bhat and F. Mueller. Making DRAM Refresh Predictable. In *Real-Time Systems (ECRTS), 2010 22nd Euromicro Conference on*, pages 145–154, July 2010.

[5] I. Bhati, M.-T. Chang, Z. Chishti, S.-L. Lu, and B. Jacob. DRAM Refresh Mechanisms, Trade-offs, and Penalties. *Computers, IEEE Transactions on*, PP(99):1–1, 2015.

[6] I. Bhati, Z. Chishti, and B. Jacob. Coordinated Refresh: Energy Efficient Techniques for DRAM Refresh Scheduling. In *Proceedings of the 2013 International Symposium on Low Power Electronics and Design*, ISLPED '13, pages 205–210, Piscataway, NJ, USA, 2013. IEEE Press.

[7] I. Bhati, Z. Chishti, S.-L. Lu, and B. Jacob. Flexible auto-refresh: enabling scalable and energy-efficient DRAM refresh reductions. In *Proceedings of the 42nd Annual International Symposium on Computer Architecture*, pages 235–246. ACM, 2015.

[8] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood. The gem5 simulator. *SIGARCH Comput. Archit. News*, 39(2):1–7, Aug. 2011.

[9] C. Brugger, A. L. Chinazzo, A. F. John, C. De Schryver, N. Wehn, A. Spitz, and K. A. Zweig. Exploiting Phase Transitions for the Efficient Sampling of the Fixed Degree Sequence Model. In *Advances in Social Networks Analysis and Mining (ASONAM), 2015 IEEE/ACM International Conference on*. IEEE, 2015.

[10] C. Brugger, V. Grigorovici, M. Jung, C. Weis, C. D. Schryver, K. A. Zweig, and N. Wehn. A Custom Computing System for Finding Similarties in Complex Networks. In *Proceedings of the 2015 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 262–267, Montpellier, France, July 2015.

[11] K. Chandrasekar, C. Weis, Y. Li, B. Akesson, O. Naji, M. Jung, N. Wehn, and K. Goossens. DRAMPower: Open-source DRAM power & energy estimation tool.

[12] K. K.-W. Chang, D. Lee, Z. Chishti, A. R. Alameldeen, C. Wilkerson, Y. Kim, and O. Mutlu. Improving DRAM performance by parallelizing refreshes with accesses. In *High Performance Computer Architecture (HPCA), 2014 IEEE 20th International Symposium on*, pages 356–367. IEEE, 2014.

[13] Z. Cui, S. A. McKee, Z. Zha, Y. Bao, and M. Chen. DTail: A Flexible Approach to DRAM Refresh Management. In *Proceedings of the 28th ACM International Conference on Supercomputing*, ICS '14, pages 43–52, New York, NY, USA, 2014. ACM.

[14] D. Dutoit, C. Bernard, S. Cheramy, F. Clermidy, Y. Thonnart, P. Vivet, C. Freund, V. Guerin, S. Guilhot, S. Lecomte, et al. A 0.9 pJ/bit, 12.8 GByte/s WideIO memory interface in a 3D-IC NoC-based MPSoC. In *VLSI Technology (VLSIT), 2013 Symposium on*, pages C22–C23. IEEE, 2013.

[15] M. Ghosh and H.-H. Lee. Smart Refresh: An Enhanced Memory Controller Design for Reducing Energy in Conventional and 3D Die-Stacked DRAMs. In *Microarchitecture, 2007. MICRO 2007. 40th Annual IEEE/ACM International Symposium on*, pages 134–145, Dec 2007.

[16] C. Gimmler-Dumont and N. Wehn. A Cross-Layer Reliability Design Methodology for Efficient, Dependable Wireless Receivers. *submitted to ACM Transactions on Embedded Computing Systems*, 2013.

[17] Y.-H. Gong and S. Chung. Exploiting Refresh Effect of DRAM Read Operations: A Practical Approach to Low-power Refresh. *Computers, IEEE Transactions on*, PP(99):1–1, 2015.

[18] T. Hamamoto, S. Sugiura, and S. Sawada. On the retention time distribution of dynamic random access memory (DRAM). *Electron Devices, IEEE Transactions on*, 45(6):1300–1309, Jun 1998.

[19] http://www.netflixprize.com/, last access: 2014-12-01.

[20] C. Isen and L. John. ESKIMO - energy savings using semantic knowledge of inconsequential memory occupancy for DRAM subsystem. In *Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium on*, pages 337–346, Dec 2009.

[21] M. Jung, C. Weis, N. Wehn, and K. Chandrasekar. TLM modelling of 3D stacked wide I/O DRAM subsystems: a virtual platform for memory controller design space exploration. In *Proceedings of the 2013 Workshop on Rapid Simulation and Performance Evaluation: Methods and Tools*, RAPIDO '13, pages 5:1–5:6, New York, NY, USA, 2013. ACM.

[22] M. Jung, C. Weis, N. Wehn, M. Sadri, and L. Benini. Optimized active and power-down mode refresh control in 3D-DRAMs. In *Very Large Scale Integration (VLSI-SoC), 2014 22nd International Conference on*, pages 1–6, Oct 2014.

[23] M. Jung, C. Weis, N. Wehn, and N. Wehn. DRAMSys: A flexible DRAM Subsystem Design Space Exploration Framework. *IPSJ Transactions on System LSI Design Methodology (T-SLDM)*, August 2015.

[24] J.-S. Kim, C. S. Oh, H. Lee, D. Lee, H.-R. Hwang, S. Hwang, B. Na, J. Moon, J.-G. Kim, H. Park, J.-W. Ryu, K. Park, S.-K. Kang, S.-Y. Kim, H. Kim, J.-M. Bang, H. Cho, M. Jang, C. Han, J.-B. Lee, K. Kyung, J.-S. Choi, and Y.-H. Jun. A 1.2V 12.8GB/s 2Gb mobile Wide-I/O DRAM with 4x128 I/Os using TSV-based stacking. In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2011 IEEE International*, pages 496–498, Feb 2011.

[25] J. Krueger, D. Donofrio, J. Shalf, M. Mohiyuddin,

S. Williams, L. Oliker, and F.-J. Pfreundt. Hardware/software co-design for energy-efficient seismic modeling. In *Proceedings of the 2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, pages 1–12, 2011.

[26] J. Lim, H. Lim, and S. Kang. 3D Stacked DRAM Refresh Management with Guaranteed Data Reliability. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, PP(99):1–1, 2015.

[27] C.-H. Lin, D.-Y. Shen, Y.-J. Chen, C.-L. Yang, and M. Wang. SECRET: Selective error correction for refresh energy reduction in DRAMs. In *Computer Design (ICCD), 2012 IEEE 30th International Conference on*, pages 67–74, Sept 2012.

[28] J. Liu, B. Jaiyen, Y. Kim, C. Wilkerson, and O. Mutlu. An Experimental Study of Data Retention Behavior in Modern DRAM Devices: Implications for Retention Time Profiling Mechanisms. *SIGARCH Comput. Archit. News*, 41(3):60–71, June 2013.

[29] J. Liu, B. Jaiyen, R. Veras, and O. Mutlu. RAIDR: Retention-Aware Intelligent DRAM Refresh. In *Proceedings of the 39th Annual International Symposium on Computer Architecture*, ISCA '12, pages 1–12, Washington, DC, USA, 2012. IEEE Computer Society.

[30] S. Liu, K. Pattabiraman, T. Moscibroda, and B. G. Zorn. Flikker: Saving DRAM Refresh-power Through Critical Data Partitioning. *SIGPLAN Not.*, 46(3):213–224, Mar. 2011.

[31] Micron Technology Inc. 4Gb: x16, x32 Mobile LPDDR3 SDRAM. July 2013.

[32] Xilinx. zC706 Evaluation Board for the Zynq-7000 XC7Z045 All Programmable Soc User Guide. April 2015.

[33] Xilinx, Inc. Memory Interface Generator (MIG). http://www.xilinx.com/products/intellectual-property/mig.html, 2015, Last Access: 18.02.2015.

[34] J. Meza, Q. Wu, S. Kumar, and O. Mutlu. Revisiting Memory Errors in Large-Scale Production Data Centers: Analysis and Modeling of New Trends from the Field. In *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2015.

[35] J. Mukundan, H. Hunter, K.-h. Kim, J. Stuecheli, and J. F. Martínez. Understanding and Mitigating Refresh Overheads in High-density DDR4 DRAM Systems. In *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ISCA '13, pages 48–59, New York, NY, USA, 2013. ACM.

[36] P. J. Nair, C.-C. Chou, and M. K. Qureshi. Refresh Pausing in DRAM Memory Systems. *ACM Trans. Archit. Code Optim.*, 11(1):10:1–10:26, Feb. 2014.

[37] K. Patel, L. Benini, E. Macii, and M. Poncino. Energy-Efficient Value-Based Selective Refresh for Embedded DRAMs. In V. Paliouras, J. Vounckx, and D. Verkest, editors, *Integrated Circuit and System Design. Power and Timing Modeling, Optimization and Simulation*, volume 3728 of *Lecture Notes in Computer Science*, pages 466–476. Springer Berlin Heidelberg, 2005.

[38] M. K. Qureshi, D.-H. Kim, S. Khan, P. J. Nair, and O. Mutlu. AVATAR: A Variable-Retention-Time (VRT) Aware Refresh for DRAM Systems. *Memory*, 2(4Gb):20, 2015.

[39] M. Sadri, M. Jung, C. Weis, N. Wehn, and L. Benini. Energy Optimization in 3D MPSoCs with Wide-I/O DRAM Using Temperature Variation Aware Bank-Wise Refresh. In *Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014*, pages 1–4, March 2014.

[40] F. Schmoll, A. Heinig, P. Marwedel, and M. Engel. Improving the Fault Resilience of an H.264 Decoder Using Static Analysis Methods. *ACM Trans. Embed. Comput. Syst.*, 13(1s):31:1–31:27, Dec. 2013.

[41] A. Sridhar, A. Vincenzi, M. Ruggiero, T. Brunschwiler, and D. Atienza. 3D-ICE: Fast compact transient thermal modeling for 3D ICs with inter-tier liquid cooling. In *Proc. of ICCAD 2010*, 2010.

[42] J. Stuecheli, D. Kaseridis, H. Hunter, and L. John. Elastic Refresh: Techniques to Mitigate Refresh Penalties in High Density Memory. In *Microarchitecture (MICRO), 2010 43rd Annual IEEE/ACM International Symposium on*, pages 375–384, Dec 2010.

[43] V. K. Tavva, R. Kasha, and M. Mutyam. EFGR: An Enhanced Fine Granularity Refresh Feature for High-Performance DDR4 DRAM Devices. *ACM Trans. Archit. Code Optim.*, 11(3):31:1–31:26, Oct. 2014.

[44] R. Venkatesan, S. Herr, and E. Rotenberg. Retention-aware placement in DRAM (RAPID): software methods for quasi-non-volatile DRAM. In *Proc. of HPCA*, 2006.

[45] J. Wang, X. Dong, and Y. Xie. ProactiveDRAM: A DRAM-initiated retention management scheme. In *Computer Design (ICCD), 2014 32nd IEEE International Conference on*, pages 22–27, Oct 2014.

[46] C. Weis, M. Jung, P. Ehses, C. Santos, P. Vivet, S. Goossens, M. Koedam, and N. Wehn. Retention Time Measurements and Modelling of Bit Error Rates of WIDE I/O DRAM in MPSoCs. In *Proceedings of the conference on Design, Automation & Test in Europe*. European Design and Automation Association, 2015.

[47] T. Zhang, M. Poremba, C. Xu, G. Sun, and Y. Xie. CREAM: a Concurrent-Refresh-Aware DRAM Memory Architecture. In *High Performance Computer Architecture (HPCA), 2014 IEEE 20th International Symposium on*, pages 368–379. IEEE, 2014.

[48] D. Zhu, R. Wang, Y. Wei, and D. Qian. Reducing DRAM refreshing in an error correction manner. *Science China Information Sciences*, pages 1–14, 2015.

AppendixD PARTIAL REPORT

Partial report presented and evaluated during the development phase in fulfillment of the requirements of this work.

Explorations on Approximate DRAM with DRAMSys

Éder Zulian[1], Matthias Jung[2], Marcelo Johann[1], Norbert Wehn[2]

[1]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brazil

[2]Department of Electrical and Computer Engineering
University of Kaiserslautern – Kaiserslautern – Germany

{efzulian,johann}@inf.ufrgs, {jungma,wehn}@eit.uni-kl.de

Abstract. *In this work DRAMSys, a flexible memory subsystem design space exploration framework, is combined with 3D-ICE, a thermal simulator library which can perform transient thermal analysis of integrated circuits, in order to investigate and better understand the impact on power usage and retention errors occurrence when disabling DRAM's refresh. This investigation may also stimulate reflection on which sort of application could benefit from omitted refreshes.*

1. Introduction

Computer systems permeate today's life. Memory is an essential part of computers and is present in cheap electronic gadgets, mobile computers (phones, tablets, wearables), high availability data communication equipments and data servers to cite a few.

Dynamic Random Access Memory (DRAM) is the primary memory of most computer systems since many decades. The simple hardware structure, the relative high access speed, the relative lower costs of production and the scalability provided by this memory technology corroborate this scenario.

Despite technological advances, there is growing disparity between processor and memory speeds. This issue was formally announced two decades ago in a paper as the *memory wall* [Wulf and McKee 1995].

The process scaling of current semiconductors is approaching its limits. Therefore new solutions are necessary in order to deliver increased storage capacity and bandwidth. Stacked three-dimensional structures, in which each layer consists in a bi-dimensional die, are being explored. This new technology carries new challenges related to power density and removal of heat [Weis et al. 2015].

Along with storage capacity and bandwidth, power efficiency has become a major concern in current computer system designs [Cameron et al. 2005, Benini et al. 2000]. Trade-offs involving power and performance are recurrent among system-level architects. *Approximate computing* breaks with the all-or-nothing correctness philosophy adopted so far by computer systems and, leveraged by power savings and gains in performance, brings the possibility to explore a not so deeply explored trade-off: *power savings against acceptable losses in accuracy*. This concept applied to DRAMs, which is often referred

to as *approximate DRAM storage* [Teman et al. 2015, Jung et al. 2016], explores trade-offs involving energy savings and performance improvements in the memory subsystem against acceptable inaccuracy in the computation caused by non-critical data corruption.

In this work *DRAMSys* [Jung et al. 2015], a flexible memory subsystem design space exploration framework, is combined with *3D-ICE* [Sridhar et al. 2014], a tool which can perform transient thermal analyses of integrated circuits, in order to study and better understand the impact on power usage and retention error occurrence of an audacious strategy to DRAM cells refresh - *omitting refresh*.

2. DRAM

DRAM is a volatile memory, i.e., its content fades when power is turned off, and it is also dynamic, i.e., during normal operation the binary information written to a cell will leak off. The time that a cell can hold the data is called *retention time*. DRAM cells have to be periodically refreshed to maintain the data integrity. While a DRAM bit cell requires only a single transistor and capacitor pair an SRAM cell uses six transistors. Therefore DRAM offers much higher densities and lower cost per bit. However, DRAM performance is worse than SRAM's mostly because the requirement of a specialized circuitry to sense and amplify slight voltage differences in the bitlines caused by a passive storage cell.

Commodity DRAM devices are designed to be simple and inexpensive to manufacture, so they do not bring much intelligence embedded on them. Thus the memory controller is responsible for refreshing all banks, ensuring that all timing constraints are met, avoiding timing violations, avoiding collisions on the data bus, respecting the turnaround time when the directions of the data bus changes, etc. Modern computers use improved versions of the original DRAM architecture. Along with process scaling, several techniques have been applied in order to extend the lifespan of the DRAM architecture up to these days. For example, the bandwidth was increased by doubling the data clock rate, with on-die termination used to match line impedance increasing the quality of signals allowing higher frequency and the inclusion of delay-locked loop circuits used to compensate signal skew.

2.1. Basic Circuits

Modern DRAM devices use One-Transistor One-Capacitor (1T1C) DRAM cells to store a single bit of data. 1T1C DRAM cells consist of an access transistor controlled by a wordline, which selectively connects a bit storage capacitor to a bitline. This storage cell is a passive circuit that must be carefully sensed in order to read the logic level stored in it. The potentials V_{DD} or V_{SS} on the storage capacitor correspond to logical 1 or 0.

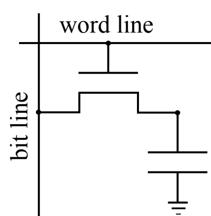


Figure 1. 1T1C DRAM Cell Structure

Multiple cells are connected to the same bitline but only one cell per bitline will have its access transistor open at a given time. When inactive, the bitlines are held at mid-rail potential $V_{DD}/2$. At the beginning of a memory access one of the wordlines is enabled causing a balance of charges between the bitline and the cell capacitor. Since the cell capacitance is roughly an order of magnitude smaller than the bitline capacitance the bitline voltage changes only a few hundred millivolts above or below $V_{DD}/2$. For this reason a *bitline sense amplifier* is needed to sense this small change in the bitline voltage and amplify it to the appropriate voltage.

The bitline sense amplifier is a regenerative latch that amplifies the bitline voltage to full-rail. The sense amplifiers will keep the resultant levels until the DRAM array is precharged for another access. Thus they act as a *row buffer* that caches an entire row of data. Subsequent reads to the same row of cells can be done directly from the sense amplifier without accessing the cells themselves.

To sense very small changes in the bitline voltage modern DRAM devices use differential sense amplifiers which connect to a pair of bitlines. While the slight voltage variation of one line is sensed the other bitline works as a reference voltage. This, however, makes possible to sense only one bitline of the pair and consequently limits the number of memory cells that can be accessed at a given time.

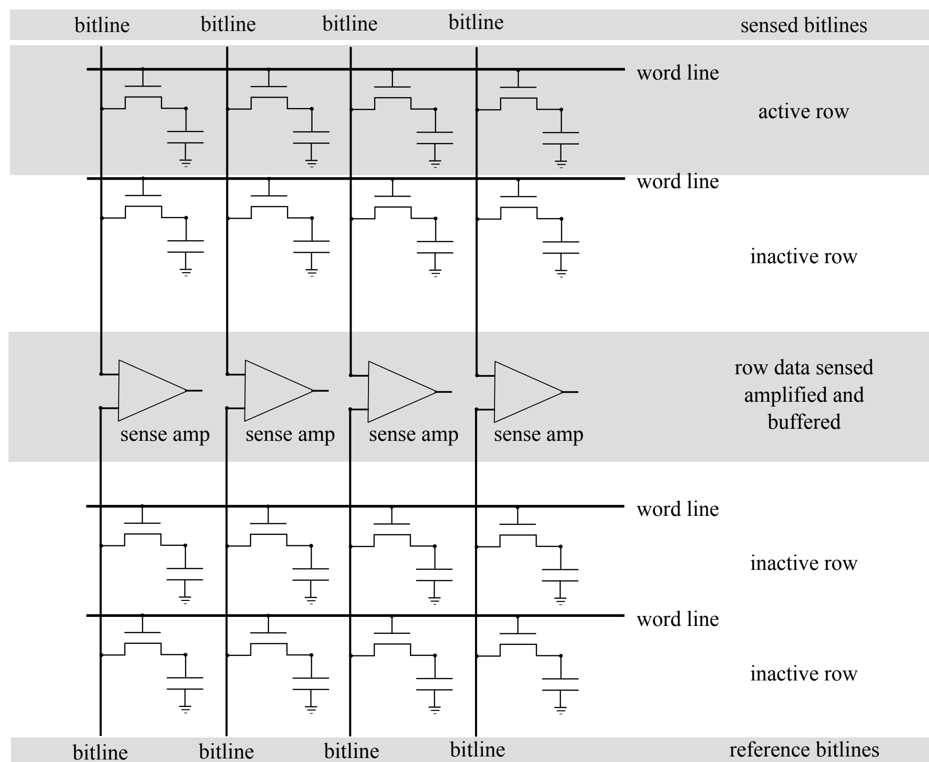


Figure 2. Differential Sense Amplifiers (Open Bitline Structure)

Differential sense amplifiers require bitlines that are very similar in terms of capacitance, voltage, path length and number of cells connected to them. The two main array structures of DRAM devices are open bitline and folded bitline. In the open bitline

structure the array is divided in segments and differential sense amplifiers are connected to bitlines of different segments while in a folded bitline structures the pair of bitlines comes from the same array segment. Basically, the open bitline structure requires less area but, when compared to folded bitline structure, is more susceptible to electronic noise.

Since data reads to 1T1C DRAM cells are destructive (due to the balance of charges between the bit line and the storage cell), to complete the DRAM read cycle the data must be written back into the memory cell. The n-channel access transistor must be fully turned on. To store V_{DD} level in the memory cell, the wordline must be raised to $V_{DD} + V_T$ where V_T is the access transistor threshold voltage. Extra hardware may be required to generate this potential above V_{DD}.

2.2. DRAM Peculiarities

Due to process variation when integrated circuits are fabricated the attributes of nodes (length, width, oxide thickness) vary. Since the beginning of DRAM production different retention times among cells have been noticed, typically ranging from hundreds of milliseconds to tens of seconds [Jacob et al. 2008, p. 356]. Cells that are marked by shorter retention time are known as *weak-cells*.

Advances in fabrication process allow high-density DRAM devices, with smaller cells whose dimensions are approaching fundamental dimensions for current semiconductor technologies. For a given fabrication process, considering changes only in dimensions, not in materials (e.g., dopant elements, insulators), the smaller the number of atoms separating gate and drain the easier the leakage current flows and the higher the influence of the environment is.

Moreover, a not so often emphasised or sometimes forgotten phenomenon is the existence of DRAM cells with variable retention time (VRT). VRT is caused by microdefects in the p-n junction which are very difficult to detect. A microdefect of the right type and at the right place serve as an electron-hole generation-recombination (GR) center. The GR center may conduct current through the p-n junction acting as an additional and significant leakage component [Jacob et al. 2008, p. 826]. Therefore there is a non-zero probability that a given cell changes its retention time for a while. The bi-modal retention time behaviour of some DRAM cells makes unfeasible a 100% accurate map of weak-cells. Since VRT is rare and difficult to detect, DRAM manufacturers are not interested in finding and eliminating it as long as the proposed refresh time is enough to make sure that even cells with VRT can pass all tests and work properly [Jacob et al. 2008, p. 826].

2.3. Refresh

By the time of DRAM invention, as stated in its patent [Dennard 1968], the regeneration of the binary information stored in its cells (capacitors whose charge tends to leak off with time) should take 10% to 20% of the device's operation time and conventional operations the remaining 80%. This means that since the very beginning of DRAM technology it is a well known fact that increases on DRAM refresh rates come with penalties: performance degradation and higher power consumption. Nonetheless for many years power consumption was not the main focus of system designers.

A projection for future DRAM devices is that 40% to 50% of the power consumption will be due to refresh commands [Liu et al. 2012].

The Joint Electron Device Engineering Council (JEDEC) is an independent organization that is responsible for many standards related to semiconductor engineering. The maximum time interval between refresh operations is standardized by JEDEC for each DRAM technology and it is specified by DRAM manufacturers within the device's datasheet.

The refresh interval depends on the operation temperature because leakage currents are affected by the temperature. The higher the temperature the higher the leakage current. Thus the refresh interval needs to be decreased for higher temperatures. Typically a wide range of temperatures is considered as normal operation temperature. For example, the DDR4-SDRAM standard specifies that temperatures ranging from $0\,°C$ to $85\,°C$ are the normal operation temperature for that technology. Nevertheless, for the same technology, a not so wide interval defines the extended temperature range: $85\,°C$ to $95\,°C$.

Though it is known that many cells can retain data for times much longer than the standard refresh interval, refresh requirements are based on a worst-case data decay time in order to ensure data integrity for all cells. The rareness of VRT allied to the capability of some applications to tolerate certain amount of errors make interesting the investigation of the impact on energy savings when disabling DRAM refreshes.

2.4. Error Correction Mechanisms

Along with temperature several aspects of the device's operation environment such as ambient radiation (from inside the computer system), electromagnetic interference, interactions with high-energy particles (from outside the computer system) have influence on chip-level soft errors.

Since errors in DRAM happen even though refresh requirements are met, some DRAM devices provide mechanisms for detection and possible correction of errors. Such mechanisms act decreasing the effective error rate and as consequence may be used to minimize some drawbacks of the approximate DRAM storage. In the other hand extra storage and logic are required by ECC memory with impact on power consumption and final costs. Thus the decision of combining ECC memory with approximate DRAM is non-obvious but is beyond the scope of this work.

2.5. Technological Trends

It is uncertain how long process scaling and improvements will be sufficient to maintain today's DRAM architecture. Efficiency and performance gains using the traditional DRAM have become smaller and more difficult to achieve.

New solutions are necessary to attend the ever growing demand for memory performance. Therefore DRAM manufactures search for innovations to DRAM memory architecture that lead to higher performance and lower power consumption.

2.5.1. Wide I/O

The Wide I/O memory architecture is designed to consume low power and provide a high bandwidth with focus on mobile devices and battery powered embedded systems. Wide

I/O can be stacked on top of system on chip (SoC) devices by means of vertical through-silicon via (TSV) interconnections. This accounts in favor of an optimized package size. On the other hand, the heat radiated from the SoC will pass through the memory die worsening the thermal scenario. Wide I/O high bandwidths are due to a wide memory bus that is up to 1024 bits wide.

The second version of the Wide I/O standard supports memory stacks of up to four memory chips. Each memory chip in the stack is called a *slice* and a single memory chip is divided into four *quadrants*. The electrical connection between two stacked dies is know as *Micropillar*. A *channel* is a set of physically discrete connections within the Wide I/O interface which contains all the control, data and clock signals necessary to independently control a partition of the Wide I/O device. A *rank* is a portion of memory connected to a channel. Multiple slices may be connected to form a rank.

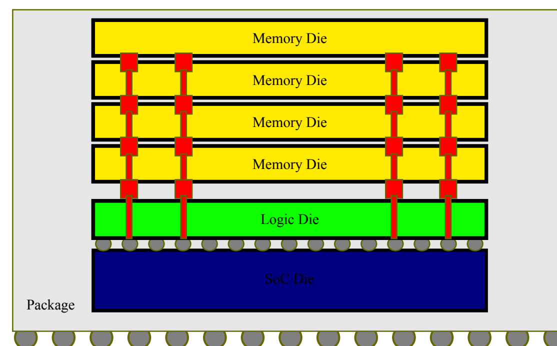


Figure 3. Wide I/O 3D memory stack

2.5.2. Hybrid Memory Cube

A Hybrid Memory Cube (HMC) is a single package containing multiple DRAM dies (four or eight are possible according to the current specification) and one logic die, all stacked together using through-silicon via technology. Within each cube, memory is organized vertically. Each memory die is divided into partitions. The combination of vertically aligned partitions (one of each memory die in the stack) with a memory controller within the logic die is called a vault and is comparable to a channel in Wide I/O architecture.

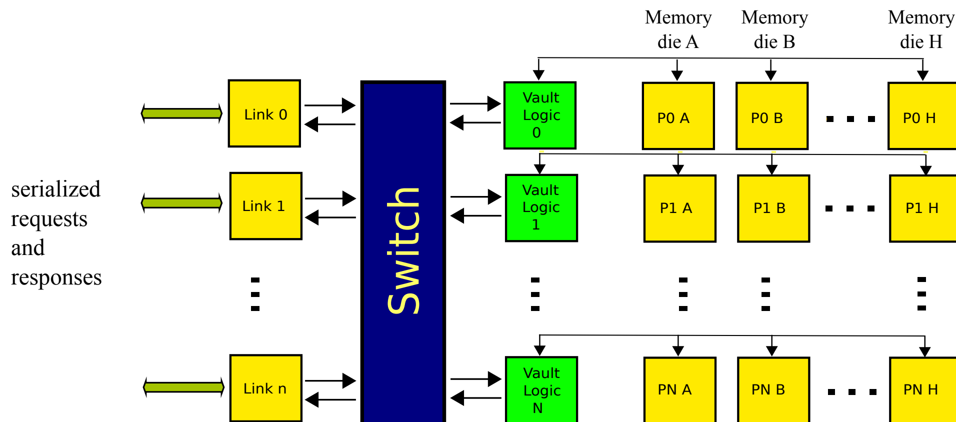


Figure 4. A representation of the HMC scheme in respect to memory accesses

The vault controller manages all memory reference operations within the vault. Operations may be buffered and executed out of order in respect to the order of arrival. Responses from vault operations back to the external serial links will be out of order. However, requests from a single serial link to the same vault/bank address are executed in order. Requests from different external serial links to the same vault/bank address are not guaranteed to be executed in a specific order and must be managed by the host controller. Each vault controller determines its own timing requirements and controls its refresh rate.

3. Design Space Exploration Tools

Scalability, speed and low costs of production allied to a successful technological evolution sustain DRAM hegemony as primary memory of computer systems. Although current technology scaling is reaching its limits, demands for memory capacity, high-performance and energy-efficiency are increasing in what looks like an unstoppable trend. Also, technological advances come with new challenging engineering problems that need to be fully understood and solved.

The design space of DRAM is huge. In order to face the new challenges that arise designers must have an in-depth understanding of the memory system and its subsystems. This combined with a detailed and accurate model of the memory system may provide a way of increasing the intuition and knowledge on how the system behaves when its parameters vary. Hence simulation provides substantial help for design space exploration. It plays an important role saving resources and anticipating the release of new solutions. DRAM design space exploration tools empower researchers and engineers with flexibility and agility to find out how to enhance current devices, to foresee how systems will behave in extreme conditions, to elaborate new refresh strategies aiming optimization, and even to create totally new models.

3.1. SystemC and TLM

SystemC is a set of C++ classes which can be used to develop event-driven applications. SystemC is described and documented in the IEEE Standard for Standard SystemC Language Reference Manual [IEEE Computer Society 2012].

Transaction Level Modelling (TLM) is an approach to modeling digital systems focused in the abstraction of communication between the primary functional blocks within a system. TLM2.0 is described and documented in a reference manual released by the Open SystemC Initiative group [Aynsley 2009].

Combined they represent a powerful alternative to create fast and still accurate virtual platforms typically used for performance analysis and architecture exploration.

3.2. DRAMSys

DRAMSys is a flexible DRAM subsystem design space exploration framework that consists of models reflecting the DRAM functionality, power consumption and retention time errors.

The exploration framework models a wide range of standard and emerging DRAM subsystems such as DDR3, DDR4, LPDDR3, Wide I/O and HMC. Implemented in C++ and SystemC this framework offers interoperability with third party tools.

A key concept behind this tool is the use of TLM2.0 to implement approximately timed (AT) simulation based on function calls instead of a pin accurate register transfer level simulation where all events are simulated. This approach ensures faster simulation with negligible losses in accuracy [Jung et al. 2013, Jung et al. 2015].

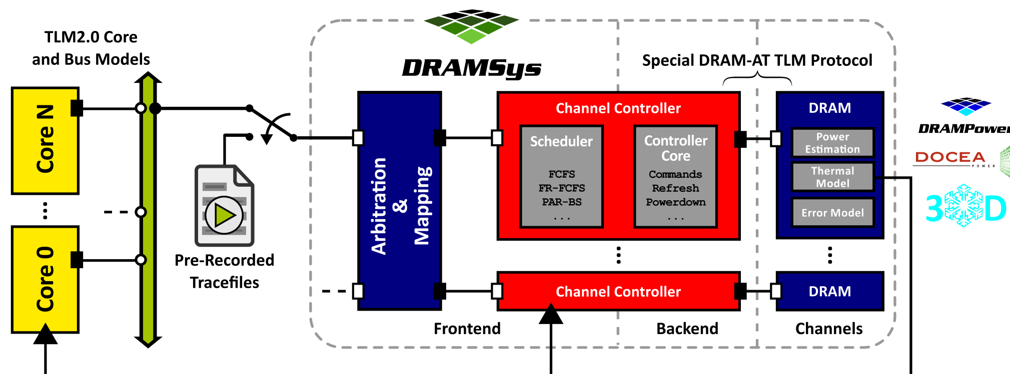


Figure 5. DRAMSys Base Architecture [Jung et al. 2015]

DRAMSys is able to process pre-recorded trace files containing memory transactions. The framework supports trace files from other simulators like Gem5 [Binkert et al. 2011] or Simplescalar [Burger and Austin 1997]. This feature facilitates efficient analysis and explorations.

3.2.1. Error Model

DRAMSys implements a DRAM bit error model [Weis et al. 2015] that enables early investigations on retention time behaviour against temperature.

The DRAM bit error model takes into consideration the possibility of DRAM cells with variable retention time behaviour. The current version supports Wide I/O and DDR3-SDRAM memories.

3.3. 3D-ICE

3D-ICE stands for 3D Interlayer Cooling Emulator. It is a Linux based Thermal Emulator Library written in C, which can perform transient analysis of 2D or 3D integrated circuits. The tool offers a client-server mode which is the operation mode relevant to this work. This feature is very useful when power traces are dynamically generated and to simulate the temperature behaviour in run-time. Client and server exchange messages through the network, the client provides power information to the server and requests the server to perform thermal simulation steps based on it. Thermal maps of individual dies are available as thermal data output of the simulator.

Two input files constitute a 3D-ICE project: a stack description file and a floor plan file. The stack description file describes structure, material properties of the 3D stack, possible heat sinks in the system, the discretization parameters, analysis parameters, and 3D-ICE commands specifying the desired outputs for the simulation. The floor plan file contains the coordinates and dimensions of the logic blocks relevant to the thermal simulation. Power information for every block specified in this file must be provided to the simulator in order to execute a thermal simulation step.

4. Goals

The aim of this work is to investigate the effect that disabling DRAM's refresh has on power usage and also on retention errors occurrence. Data is obtained from simulation experiments with DRAMSys being the main simulation environment. Since temperature plays an important role in the occurrence of retention errors 3D-ICE is used for thermal simulation. In order to achieve the intended goal the following steps are essential:

- Gain knowledge and improve skills regarding the technologies and tools involved.
- Adequate the simulation environment by interconnecting DRAMSys and 3D-ICE.
- Create inputs for the tools describing the setup.
- Execute the pertinent simulations.
- Analyse the output data.
- Generate conclusions.

5. Current Development Status

During the first part of this work a considerable amount of specialized information was studied in order to acquire essential knowledge to accomplish it. Afterwards an implementation phase took place focused on the integration of DRAMSys and 3D-ICE. The execution of simple simulation tests followed.

5.1. Development of Specialized Know-how

Essential aspects of modern DRAM devices like basic circuits and architecture were reviewed (basic building blocks such as DRAM storage cells, DRAM array structure, voltage sense amplifiers) also DRAM timing constraints, DRAM commands and the basic memory access protocol. High performance 3D stacked memories standards such as Wide I/O and Hybrid Memory Cube were visited in order to obtain details about 3D device's organization and features.

The design space exploration tool DRAMSys and the thermal emulator library 3D-ICE were sufficiently understood with respect to usage, capabilities provided and finally

in a source code level. Reference manuals of SystemC and TLM2.0 were studied and skills were developed through the implementation of features and improvements for the simulators.

5.2. DRAMSys and 3D-ICE integration

DRAMSys was adapted to act as 3D-ICE client. The client provides power information to the server and requests thermal simulation steps.

A SystemC module encapsulates a hardware or software description. They represent the basic block of a hierarchical system. A module may contain several concurrent processes used to implement its behaviour. A new SystemC module was created in DRAMSys to connect it to the 3D-ICE server. This module contains a thread process that is initialized at the beginning of the simulation and is responsible for the requests to the thermal simulator. Since thermal simulation adds an extra time overhead to the total simulation time, the period for the execution of thermal simulation steps can be chosen by the user via DRAMSys' configuration.

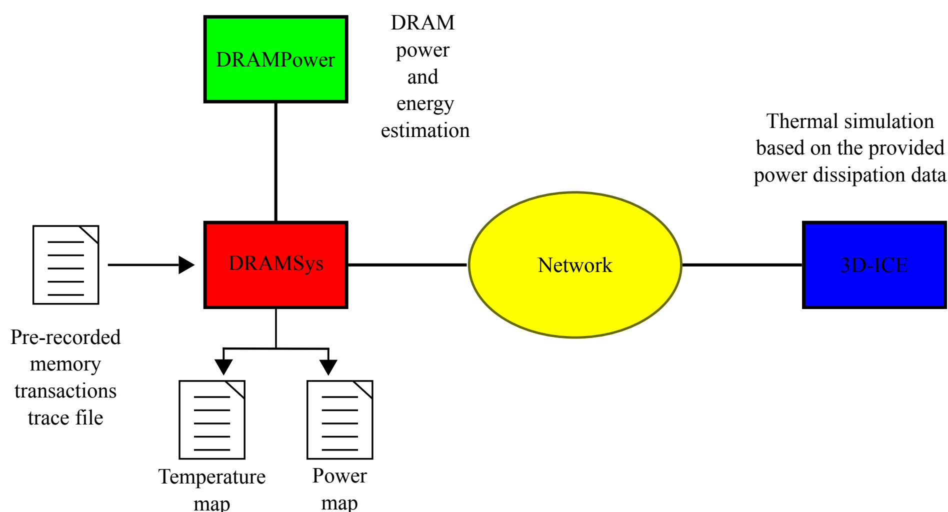


Figure 6. DRAMSys and 3D-ICE integration

An additional piece of software *IceWrapper* was used to implement the interface between DRAMSys and 3D-ICE. It was incorporated to DRAMSys' repository as a submodule. The IceWrapper was extended to provide ability to request a power map from the 3D-ICE server. Later the IceWrapper itself was ported to inside the 3D-ICE project in order to easily provide integration of 3D-ICE with any SystemC/TLM2.0 based simulation environment. Changes are currently being reviewed by the 3D-ICE team and should be incorporated in the near future. The integration of this feature is very convenient, though it is not essential to the continuity of this work.

5.3. Initial Tests

Faster simulations can be achieved by replaying pre-recorded transaction trace files [Kogel 2010]. This convenient approach is adopted in this work for the sake of time saving. Thus all the memory transactions that are executed by DRAMSys come from pre-recorded files previously generated by the Simplescalar simulator.

Two simple 3D-ICE input files were created for tests purpose: a stack description file and a floorplan file. The stack file presented below describes a DRAM die mounted on top of a printed circuit board (PCB). It also defines a spreader layer and a heat sink made of copper on top of the DRAM die. This test file is not intended to be 100% accurate, some details such as the material of the spreader layer that connects the memory die and the heat sink will be reviewed to the final version.

stack.stk

```
material SILICON:
    thermal conductivity 1.30e-4;
    volumetric heat capacity 1.628e-12;

material BEOL:
    thermal conductivity 2.25e-6;
    volumetric heat capacity 2.175e-12;

material COPPER:
    thermal conductivity 4.01e-04;
    volumetric heat capacity 3.37e-12;

heat sink:
    sink height 1e03, area 100e06, material COPPER;
    spreader height 0.5e03, area 70e06, material SILICON;
    heat transfer coefficient 1.3e-09;
    ambient temperature 318.15;

layer PCB:
    height 10;
    material BEOL;

die DRAM:
    layer 58.5 SILICON;
    source 2 SILICON;
    layer 1.5 BEOL;
    layer 58.5 SILICON;

dimensions:
   chip length 6100, width 10600;
   cell length 100, width  100;

stack:
    die DRAM_DIE DRAM floorplan "./mem.flp";
    layer CONN_TO_PCB PCB;

solver:
   transient step 0.01, slot 0.05;
   initial temperature 300.0;

output:
   TflpelDRAM_DIE.channel0, "temp_flp_element_ch0.txt", average, slot;
   TflpelDRAM_DIE.channel1, "temp_flp_element_ch1.txt", average, slot;
   TflpelDRAM_DIE.channel2, "temp_flp_element_ch2.txt", average, slot;
   TflpelDRAM_DIE.channel3, "temp_flp_element_ch3.txt", average, slot;
   Tmap  DRAM_DIE, "temp_map.txt", slot;
```

```
Pmap  DRAM_DIE, "power_map.txt", slot;
```

The following floorplan file contains the positions and the dimensions of four functional blocks within a memory die. Each block acts as a heat source. Power dissipation information regarding each block must be provided to the thermal simulation in order to execute a simulation step.

mem.flp

```
channel0:
    position   150,  100;
    dimension  2600, 5200;

channel1:
    position   3350,  100;
    dimension  2600, 5200;

channel2:
    position   150, 5300;
    dimension  2600, 5200;

channel3:
    position   3350, 5300;
    dimension  2600, 5200;
```

Both 3D-ICE input files together describe a 2D memory divided in four logic blocks each block representing an independent memory channel.

Tests were successfully executed using this simple scheme as a prototype to validate the integration between DRAMSys and 3D-ICE.

6. Next Implementation Steps and Practical Activities

The next phase begins with the creation of an elaborated simulation scenario describing a real 3D stacked memory on top of a SoC that has influence on the memory's temperature. Afterwards relevant tests will be executed in order to obtain data to posterior analysis and finally generate conclusions about the impact of disabling DRAM's refresh over power usage and retention errors occurrence. The following tasks are to be done:

- Create a 3D-ICE simulation project which consists in a stack file and floorplan file describing a 3D stacked memory on top of a SoC.
- Run simulations with refresh enabled (no errors are expected).
- Run simulations with refresh disabled (retention errors may occur).
- Collect data about power usage and retention errors occurrence in both scenarios.
- Analyse the data generating relevant information to the conclusion of this work.
- Suggest possible applications which could benefit from refresh suppression.

References

Aynsley, J. (2009). OSCI TLM-2.0 Language Reference Manual.

Benini, L., Bogliolo, A., and De Micheli, G. (2000). A survey of design techniques for system-level dynamic power management. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 8(3):299–316.

Binkert, N., Beckmann, B., Black, G., et al. (2011). The Gem5 Simulator. *SIGARCH Comput. Archit. News*, 39(2):1–7.

Burger, D. and Austin, T. M. (1997). The SimpleScalar Tool Set, Version 2.0. *SIGARCH Comput. Archit. News*, 25(3):13–25.

Cameron, K. W., Ge, R., and Feng, X. (2005). High-Performance, Power-Aware Distributed Computing for Scientific Applications. *Computer*, 38(11):40–47.

Chandrasekar, K., Weis, C., Li, Y., Goossens, S., Jung, M., Naji, O., Akesson, B., Wehn, N., and Goossens, K. DRAMPower: Open-source DRAM Power & Energy Estimation Tool. `http://www.drampower.info`.

Dennard, R. H. (1968). Field-effect transistor memory. `http://www.google.com/patents/US3387286`. US Patent 3,387,286.

Hybrid Memory Cube Consortium (2014). Hybrid Memory Cube specification 2.1.

IEEE Computer Society (2012). IEEE Standard for Standard SystemC Language Reference Manual.

Jacob, B., Ng, S., and Wang, D. (2008). *Memory systems: Cache, DRAM, Disk*. Morgan Kaufmann.

JEDEC Solid State Technology Association (2012). JESD79-4A.

JEDEC Solid State Technology Association (2014). JESD229-2 Wide I/O 2 (WideIO2).

Jung, M., Mathew, D. M., Weis, C., and Wehn, N. (2016). Efficient reliability management in SoCs - an approximate DRAM perspective. In *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 390–394.

Jung, M., Weis, C., and Wehn, N. (2015). DRAMSys: A Flexible DRAM Subsystem Design Space Exploration Framework. *IPSJ Transactions on System LSI Design Methodology*, 8:63–74.

Jung, M., Weis, C., Wehn, N., and Chandrasekar, K. (2013). TLM Modelling of 3D Stacked Wide I/O DRAM Subsystems: A Virtual Platform for Memory Controller Design Space Exploration. In *Proceedings of the 2013 Workshop on Rapid Simulation and Performance Evaluation: Methods and Tools*, RAPIDO '13, pages 5:1–5:6, New York, NY, USA. ACM.

Kogel, T. (2010). Generating Workload Models from TLM-2.0-based Virtual Prototypes for Efficient Architecture Performance Analysis. `http://www.nascug.org/events/13th/tlm20_workload_models.pdf`.

Liu, J., Jaiyen, B., Veras, R., and Mutlu, O. (2012). RAIDR: Retention-aware intelligent DRAM refresh. In *Computer Architecture (ISCA), 2012 39th Annual International Symposium on*, pages 1–12.

Sridhar, A., Vincenzi, A., Atienza, D., and Brunschwiler, T. (2014). 3D-ICE: A Compact Thermal Model for Early-Stage Design of Liquid-Cooled ICs. *Computers, IEEE Transactions on*, 63(10):2576–2589.

Teman, A., Karakonstantis, G., Giterman, R., Meinerzhagen, P., and Burg, A. (2015). Energy Versus Data Integrity Trade-offs in Embedded High-density Logic Compatible Dynamic Memories. In *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, DATE '15, pages 489–494, San Jose, CA, USA. EDA Consortium.

Weis, C., Jung, M., Ehses, P., Santos, C., Vivet, P., Goossens, S., Koedam, M., and Wehn, N. (2015). Retention Time Measurements and Modelling of Bit Error Rates of WIDE I/O DRAM in MPSoCs. In *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, DATE '15, pages 495–500, San Jose, CA, USA. EDA Consortium.

Wulf, W. A. and McKee, S. A. (1995). Hitting the Memory Wall: Implications of the Obvious. *SIGARCH Comput. Archit. News*, 23(1):20–24.