IPSJ Transactions on System LSI Design Methodology Vol.8 63–74 (Aug. 2015)

[DOI: 10.2197/ipsjtsldm.8.63]

Invited Paper

DRAMSys: A Flexible DRAM Subsystem Design Space Exploration Framework

Matthias Jung[1,a) Christian Weis[1,b) Norbert Wehn[1,c)

Received: February 26, 2015, Released: August 1, 2015

Abstract: In systems ranging from mobile devices to servers, Dynamic Random Access Memories (DRAM) have a big impact on performance and contributes a significant part of the total consumed power. Conventional DDR3-based solutions are stretched thin as their maximum bandwidth is limited by the I/O count and interface speed. As new solutions are coming onto the market (JEDEC DDR4, JEDEC WIDE I/O, Micron's hybrid memory cube: HMC or JEDEC's high bandwidth memory: HBM) it is critical to evaluate the performance of these solutions and assess their suitability for specific applications. Furthermore, in systems with 3D stacking, the challenges of high power densities and thermal dissipation are exacerbated. It is crucial to have a flexible and holistic DRAM subsystem framework for exhaustive design space explorations, which can handle all this different types of memories, as well as the aspects of performance, power and temperature.

Keywords: DRAM, modelling, TLM2, memory subsystem, controller, optimisation

1. Introduction

The increasing gap between the bandwidth requirements of recent multi-core architectures and the I/O data rate delivered by the attached main memories (DRAM), known as the Memory Wall [1], limits the performance of today's data-intensive applications. Recent memory subsystems based on JEDEC Double Data Rate 3 (DDR3) [2] or DDR4 [3] try to hide this gap, to some extent, by using faster and multiple memory interfaces. However, the number of I/O pins is limited by the package, power considerations and costs. The energy consumed per bit for accessing off-chip memory is two to three orders of magnitude higher than the energy required for on-chip memory accesses. This is due to complex and power hungry I/O transceiver circuits that have to deal with the electrical characteristics of the high-speed interconnections (transmission lines) between the chips.

Moreover, 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 [4], in which 49% of the total power consumption has to be attributed to DRAMs. Thus, the efficient utilisation of the available DRAM bandwidth and the efficient usage of DRAM power-down modes are the major contributions to a high energy efficiency of DRAM subsystems and the computing system in which they are integrated.

Three-dimensional (3D) stacked memories like WIDE I/O [5], Micron's Hybrid Memory Cube (HMC) [6], [7], [8] have been proposed as a promising solution to the memory wall and the high power consumption. These memories reduce the distance between CPU and external RAM from centimetres to micrometres by means of TSV (through silicon via) technology. The available bandwidth has increased but more importantly this technology provides a major boost in energy efficiency in comparison to standard off-chip DDR3/4 DRAM devices [9], [10], [11]. The combination of high bandwidth communication with the lower power consumption of 3D integrated memory is an ideal fit for high-performance and embedded applications.

However, a 3D stacked System on Chip (SoC) aggravates the thermal crisis, which can provoke errors in circuits and especially in the stacked DRAMs as they are highly sensitive to temperature changes and have to 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 [12]. This DRAM refresh operation must be issued periodically and causes both performance degradation and increased energy consumption. Liu et al. [13] predicted that 40% to 50% of the power consumption of future DRAM devices will be caused by refresh commands. 3D integrated DRAM worsens the temperature behaviour. Due to the much increased leakage at the cells the refresh frequency needs to be adjusted accordingly to avoid data loss (retention errors).

To tackle the above mentioned challenges with respect to applications, performance, power, temperature, retention errors and different DRAM architectures a holistic exploration framework is needed. **Figure 1** shows an overview of the design space exploration framework DRAMSys:

- It consists of **models** that are reflecting the DRAM functionality, power, temperature and retention time errors.
- With these models system designers are able to **analyse** the limiting parameters and issues. Therefore, the framework

[1] University of Kaiserslautern, Kaiserslautern, Germany
[a)] jungma@eit.uni-kl.de
[b)] weis@eit.uni-kl.de
[c)] wehn@eit.uni-kl.de

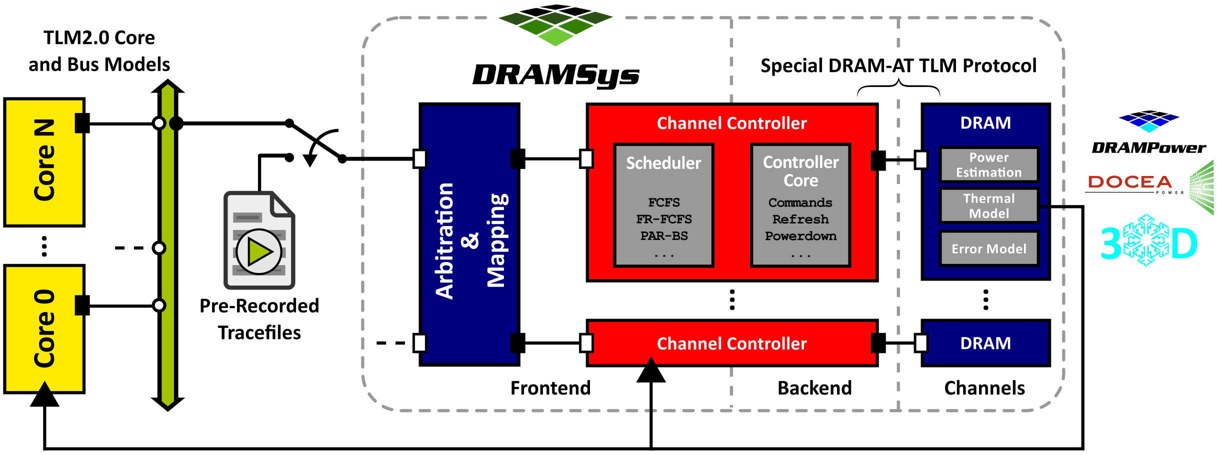


Fig. 2 Base architecture of DRAMSys.

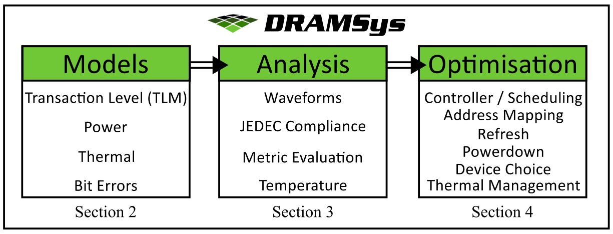


Fig. 1 Design space exploration framework DRAMSys.

provides several analysis tools that assist the designer.

- With this valuable insights the designer is able to **optimise** the DRAM subsystem with respect to the controller architecture, power and thermal management as well as device selection and channel configuration for a specific application.

Consequently, the paper is organised as follows: Section 2 discusses the base models of DRAMSys including functional, power and thermal modelling, as well as a retention time error model for DRAMs. Section 3 explains the analysis and debug capabilities of DRAMSys. Furthermore, Section 4 demonstrates optimisations on several examples. Section 5 surveys the related work and Section 6 finally concludes the paper.

2. Models

The main objective of our exploration framework is to optimise the DRAM subsystem. Hence, fast and accurate models are needed for a truthful exploration. However, there is a challenging trade-off between a fast and an accurate simulation. Traditional cycle and pin accurate (CA) Register Transfer Level (RTL) models provide the highest temporal accuracy, but they are inflexible in terms of the large design space and the very long simulation times. This is due to the large number of signals, processes and events that have to be simulated [14]. However, it is possible to simulate at a higher level of abstraction without loosing simulation accuracy.

One way to achieve a higher abstraction level is to use the C++ based SystemC Transaction Level Modelling (TLM2.0) IEEE Standard [15]. TLM can help to speedup the simulation by replacing all pin-level events with a single function call. For instance, a single bus transaction produces approximately 75 events in an RTL simulation compared to only a handful of events in a TLM simulation [16]. It is possible to reach speedup factors up to 10.000 x [15]. Moreover, TLM provides interoperability and easy integration of other TLM components. However, simulation speed comes at the cost of reduced timing accuracy. For the purpose of modelling DRAM subsystems, the standard TLM coding styles are not accurate enough to reflect a realistic behaviour. Therefore, we show in Section 2.1 a DRAM specific extension of the TLM standard.

Our framework supports a wide range of standard and emerging DRAM subsystems such as DDR3, DDR4, LPDDR3, WIDE I/O and HMC. Therefore, the framework is composed of flexible and extensible models that are designed in a modular fashion.

The DRAMSys framework uses TLM as the main virtual platform infrastructure and can be connected to any TLM2.0 based core and bus models for generating input data for the subsequent memory subsystem. To get even faster simulations it is possible to record transaction traces and replay them with elastic trace players [17]. The ability to process traces from other simulators like Gem5 [18] or Simplescalar [19] opens up the opportunity of using multiple sources for analysis and explorations. It can be used in professional virtual platform environments like Synopsys Platform Architect [20] or it can be used as a standalone simulator with native SystemC TLM2.0.

Figure 2 shows the flexible base architecture of the framework. DRAMSys itself consists like state-of-the-art memory controllers of a frontend and a backend part. The frontend contains an arbitration and mapping block that handles the incoming transactions and forwards them to the different channel schedulers according to specific priority schemes and mappings. The single channels of the subsystem are independent. Therefore each channel has its own scheduler and controller. The scheduler module collects transactions and reorders them with respect to latency and power savings and issues them to the backend with the channel controller that takes care of the correct use of the DRAM. DRAMSys supports state-of-the-art scheduling algorithms, FR-FCFS [21], Par-BS [22] and SMS [23] or it can simply disable the scheduling unit. Furthermore, the model has a Reorder Buffer (ROB) to provide in-order responses to the requester and it also

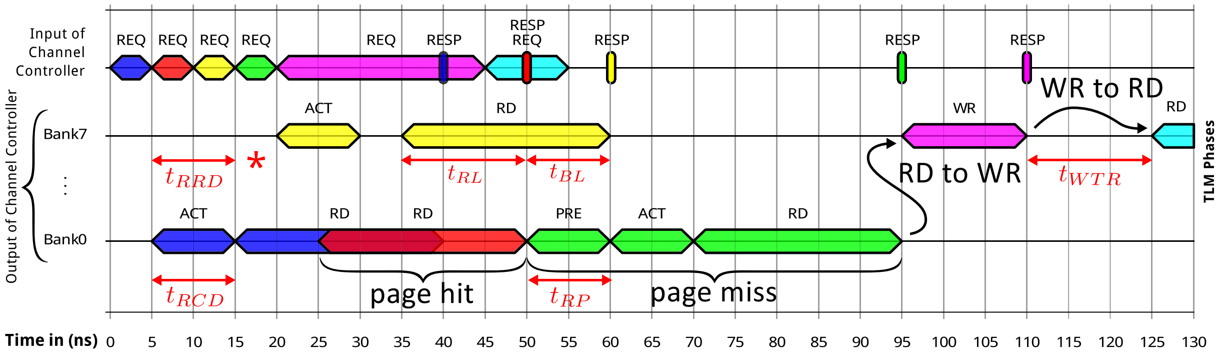


Fig. 3 Example of a typical TLM trace with DRAM related phases.

supports a multi-rank configuration of the DRAM subsystem.

Since SystemC is based on the object oriented C++ language we can easily exchange components, like the scheduling algorithms, due to predefined class interfaces. Therefore, this framework gives us the flexibility for exhaustive explorations and research, which are impossible on Register Transfer Level (RTL).

2.1 Functional DRAM TLM Model

All connections are implemented in the TLM2.0 Approximately Timed (AT) coding style. An exception is the connection between controller and channel: For this connection we extended the TLM2.0 non-blocking protocol with DRAM specific phases, called DRAM-AT [24] (see Fig. 2). With these phase extensions we can achieve the exactly required accuracy to observe e.g., the detailed impact of different address mappings or reordering algorithms of the scheduler.

The TLM non-blocking base protocol consists of the following phases: BEGIN_REQ, END_REQ, BEGIN_RESP and END_RESP. Instead of simulating every clock cycle, the simulator is triggered only at the BEGIN (<) and END (>) phase events. Using the JEDEC standards [2], [3], [5] we have defined additional application specific phases for the different DRAM commands by means of TLM2's DECLARE_EXTENDED_PHASE() macro. These phases are calibrated to the cycle accurate behaviour of JEDEC's DRAM standards.

Figure 3 shows an example of a typical trace with DRAM specific TLM phases, which are depicted per bank. The first line shows the input of the standard TLM2.0 target socket of the channel controller and the following lines the output to the DRAM device. Due to an implemented input buffer (queue) the controller of this example is able to handle a new request every clock cycle. It has a configurable input buffer size, which leads to stalling in case the buffer is full.

The figure shows examples for the timing dependencies, e.g., the ACT in Bank7 needs to be shifted by one clock cycle because of a command bus conflict with the scheduled RD command in Bank0 (*). The second RD command in Bank0 can start already after the burst length (t_{BL}) of the first RD (page hit). The third RD command on Bank0 has to access another row. Therefore a precharge- (PRE) and an activate-command (ACT) are issued in advance (page miss). The dependencies of consecutive RD and WR commands are shown at the end of the trace example.

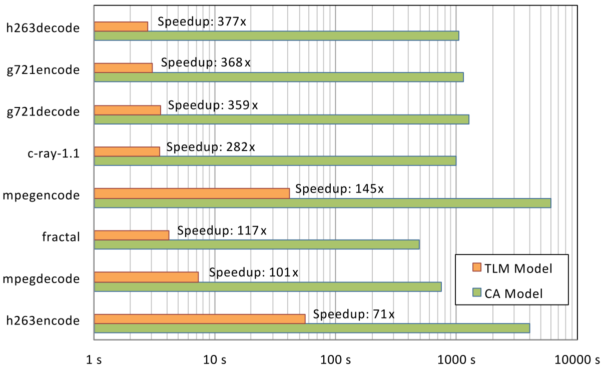


Fig. 4 Speed-up comparison TLM vs. CA SystemC.

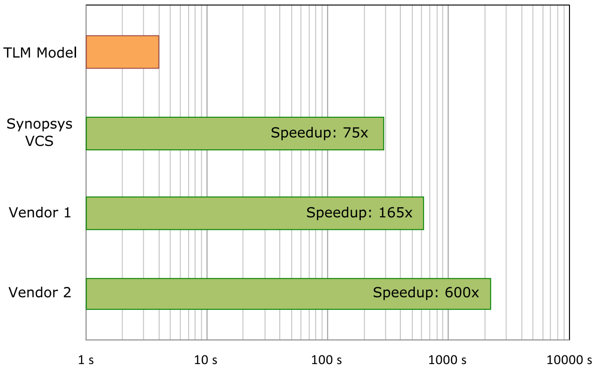


Fig. 5 Speed-up comparison TLM vs. RTL simulation.

We compared the TLM model of the framework with a cycle accurate (CA) SystemC implementation (**Fig. 4**) using the mediabench benchmark [25]. The benchmark traces are generated by means of the Simplescalar simulator with a 16 KB L1 D-cache, 16 KB L1 I-cache, 128 KB shared L2 cache and 32-byte cache line configuration. We filtered out the L2 cache misses for instructions and data, and obtained a trace of the transactions meant for the DRAM. The TLM model is very fast with respect to runtime. For instance the mediabench mpeg2encode runs 1 h 41 m with the CA model compared to 42 s with the TLM model, giving a speedup of 145 x. Similarly with the mediabench h263decode we achieved a speedup of 377 x compared to the CA implementation.

Furthermore, we quantify the speedup of the TLM model against RTL simulations with the image processing application shown in Section 4.1. **Figure 5** shows the simulation time results of our TLM model vs. RTL simulators from three different vendors. We see an expected speedup ranging from 75 x to 600 x. In both comparisons (CA SystemC and RTL) the temporal accuracy of the cycle accurate simulations is maintained by the TLM model. Thus, the DRAM-AT protocol provides, together with the other components of the framework, a perfectly balanced accuracy-speed trade-off.

2.2 DRAM Power Model

Since DRAMs contribute significantly to the power consumption of today's systems, there is a need for accurate power modelling. One of the most common ways in research and industry is using Micron's power calculator [26], which estimates the power from data sheet and workload specifications. However, this model is not accurate enough, as it assumes certain workload characteristics. To overcome this limitation, we focus on an improved version, called DRAMPower [27], [28], which uses the actual timings instead of the minimal timings from datasheets. We modified DRAMPower that it can be used as a library, which can be easily integrated in a C++ based simulator like our TLM2.0 based model to calculate the power consumption online during the simulation.

2.3 3D-DRAM Thermal Model

3D packaging of systems like WIDE I/O DRAM starts to break down the memory and bandwidth walls. However, this comes at the price of increased power density and less horizontal heat removal capability of the thinned dies. The thermal issues of 3D ICs cannot be solved by tweaking the technology and circuits alone. It is crucial to analyse the behaviour of the whole system. Therefore, thermal simulators like 3D-ICE [29] or DOCEA Power [30] can be connected to DRAMSys for closed-loop simulations [31], as shown in Fig. 2. These closed-loop simulations are necessary to quantify the effects on the DRAM (refresh period adoption) and processor throttling analysis through a sophisticated power and thermal management or task migration. In this scope all power contributors which influence the thermal profile are considered, as well the resulting performance impact. In Section 4.3 we show an example where we used the closed-loop simulation to develop a new refresh strategy for 3D-DRAMs.

2.4 DRAM Error Model

DRAM cells use capacitors as volatile and leaky bit storage elements. The time spent without refreshing them is called retention time. It is well known that the retention time depends inverse exponentially on the temperature. In 3D stacking, the challenges of high power densities and thermal dissipation are exacerbated and have a much stronger impact on the retention time of 3D-stacked WIDE I/O DRAMs that are placed on top of an MPSoC.

Consequently, a retention error aware DRAM model is key to analyse, for instance, the impact of lower refresh rates or disabling refresh completely on the executed application. Especially for error resilient applications this can be exploited, to save energy [32]. We measured the retention times of WIDE I/O and DDR3 DRAM devices using different data pattern reaching from simple `0xFF`, `0x55`, `0xAA` to random pattern (`RND`). We observed data pattern dependencies (compare **Fig. 6**) and variable retention times. These data are used to create a DRAM retention time error model [33].

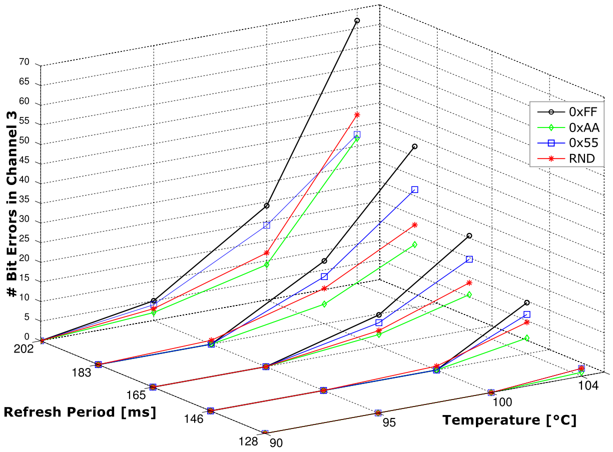


Fig. 6 Measurement results with fixed refresh periods.

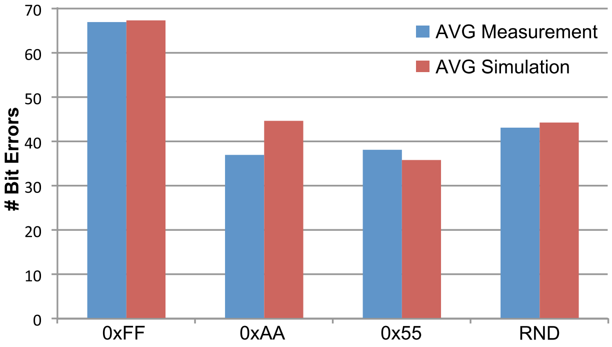


Fig. 7 Comparison of simulation and measurements for a refresh period of 202 ms and a temperature of 90 °C.

Figure 7 shows the comparison of the averaged results of 30 x repeated model simulations and real measurements of the WIDE I/O DRAM. We see that our error model implements the correct trend for the data pattern dependency and has bit error rates near to the measured values. The overhead of the retention-aware DRAM bit error model with respect to the simulation execution time of DRAMSys is in average only 30%. Thus, our proposed model can be used for Monte-Carlo-Simulations and is suitable for the early investigations on the temperature vs. retention time trade-off in future 3D-stacked MPSoCs with 3D-DRAMs.

3. Analysis

Based on the described models of the framework a system designer is able to analyse the behaviour of the DRAM subsystem. To understand the key parameters and limiting issues of the subsystem the DRAMSys framework provides several analysis tools that are required to approach the optimisation goals defined by the system level designers, shown in Fig. 1.

DRAMSys allows to record all phases of the DRAM-AT protocol in a trace SQLite [34] database. The Trace Analyser is a comfortable tool for the evaluation of these recorded traces. It illustrates the different requests and DRAM commands and the utilisation on the different banks as shown in **Fig. 18**. Exploiting the power of SQL, the data aggregation in the mass of data happens quickly and the tool provides a user friendly handling, that offers a quick navigation through the whole trace with millions of

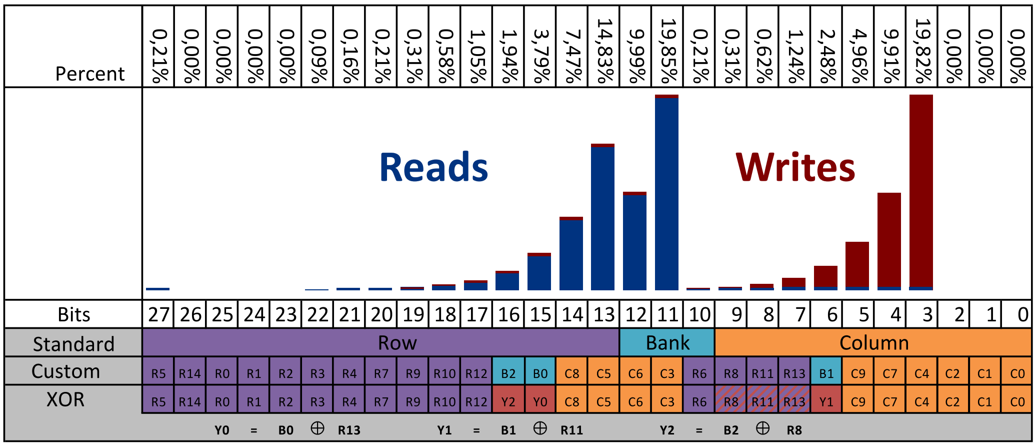


Fig. 8 Address mapping analysis: Standard vs. Custom vs. XOR mapping.

DRAM commands.

An evaluation of the traces can be performed with the powerful Python [35] interface of the Trace Analyser. The different metrics are described as SQL statements and formulas in Python and can be customised and extended without recompiling the tool. Typical metrics are for instance: the memory utilisation (bandwidth), the average response latency or the percentage of time spent in power-down.

The same Python interface is also used to run testing scripts on the recorded traces. Those scripts check if the traces fulfil all constraints defined by the respective JEDEC standards. If a test does not hold, the conflicting transaction is indicated. This feature is really useful if new controller architectures and ideas are evaluated, to validate the JEDEC compliance.

Furthermore, the framework provides several scripts that analyse the access patterns of an application with respect to the addresses and stride accesses, as described in Section 4.1.

4. Optimisation

In this section, we demonstrate the capabilities and advantages of our design space exploration framework by means of several examples. These use cases show how we accomplished the different optimisation targets, such as higher bandwidth or energy efficiency. These are achieved by the creation of a customised memory controller (using a clever address mapping), the implementation of efficient power-down policies (staggered and bankwise power-down) and improved refresh management techniques (bankwise refresh and refresh aware scheduling). To master these intended optimisation goals we deploy the advanced models and analysis tools of our framework, as shown in Fig. 1.

4.1 Address Mapping

The DRAM address mapping defines, which bits of the address are mapped to the DRAM channels, ranks, banks, rows and columns. Usually this mapping is done in a ROW-BANK-COL fashion, as depicted in **Fig. 8** (Standard).

In many applications that have a regular or fixed memory access pattern, a memory controller with advanced scheduling mechanisms is an overbuilt. Especially for FPGA based applications e.g. image processing, an optimised DRAM address mapping can supersede the best scheduler because it can maximise the number of row buffer hits and exploit the bank level parallelism of the DRAM device. An application specific memory controller (ASMC) is lean and energy efficient while it provides exactly the required bandwidth for a specific application. Our framework supports the creation of such memory controllers with the help of our analysis and optimisation tools.

DRAMSys provides a script that analyses a recorded DRAM access trace regarding the toggling rates of each address bit. An example for this analysis is shown for an image processing task on an FPGA in Fig. 8. The framework automatically suggests a new custom address mapping function, which is derived according to following rules:

- Map the bits with the highest activity to the columns. This helps to increase the number of row hits.
- Map the bits with the lowest activity to the rows. This reduces the number of row misses.
- The remaining bits are mapped to the banks.

Figure 8 shows this custom address mapping. Instead of a scheduling component in the frontend of a DRAM controller a small hardware component, called address scrambler, that implements the mapping function by rewiring the address lines is automatically generated from DRAMSys as Verilog code. The advantage of this automatic address scrambler generation is that the system developer gets an improved data placement in the DRAM.

However, for this application the custom mapping in Fig. 8 shows an imbalance of the bank parallelism for reads and writes, since the read requests have more bank bits available than the writes. This issue can be solved by using a technique for CPU based architectures from Refs. [36] and [37], where the bank bits are XORed with selected row bits. In our example we XOR the bank bits with the row bits that have the highest write activity to maintain the required balance and therefore improve the memory bandwidth.

State of the art FPGA memory controllers support only limited possibilities to change the address mapping. For instance, the Xilinx MIG memory [38] controller supports only a ROW-BANK-COLUMN and BANK-ROW-COLUMN address mapping scheme. With our framework we generated the proposed address scrambler and used it as a frontend for the MIG memory

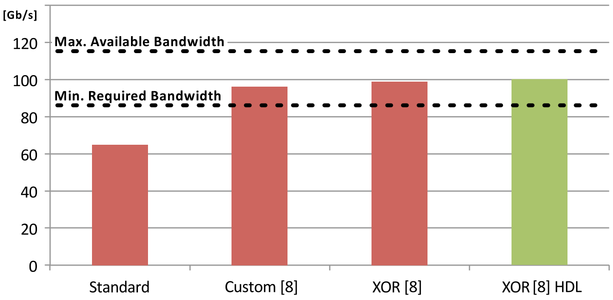


Fig. 9 Results of different address mappings.

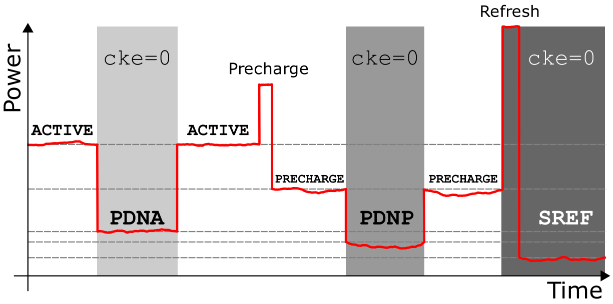


Fig. 10 The three different DRAM power-down modes.

controller. DRAMSys also assists to configure the address mapping of memory controllers that target ASIC implementations, such as Refs. [39], [40], [41].

Figure 9 shows the results of the simulation of the 3 different address mappings: Standard, Custom and XOR with DRAMSys that is configured to model the Xilinx MIG, as well as the archieved bandwidth on the real hardware (XOR HDL). We see that the XOR mapping can archieve a 30% higher bandwidth compared to the standard mapping. Furthermore we see that the simulation with the framework deviates from the real hardware measurement only by 1%.

4.2 Staggered Power-Down

Besides the normal active mode operations (activate, read, write, precharge, refresh), a DRAM is capable to enter power-down modes to save energy (set the clock-enable signal cke to low). The different DRAM powermodes (shown in **Fig. 10**) can be described as follows:

Active (ACTIVE): At minimum one bank is active (in ACT state), no power-down (cke=1), no internal refresh, the DRAM controller has to schedule refresh commands.

Precharge (PRECHARGE): All banks are closed and precharged, no power-down (cke=1), no internal refresh. The DRAM changes the state from ACTIVE to PRECHARGE by issuing a precharge command (PRE).

Precharge Power-Down (PDNP): All banks are closed and precharged (in PRECHARGE state, cke=0) and no internal refresh.

Active Power-Down (PDNA): At minimum one bank is active (in ACTIVE state, cke=0) and no internal refresh.

Self-Refresh (SREF): All banks are precharged and closed, the DRAM internal self-timed refresh is triggered (cke=0).

A non-optimised highly opportunistic self-refresh entry policy results in an increased average power, which should be avoided.

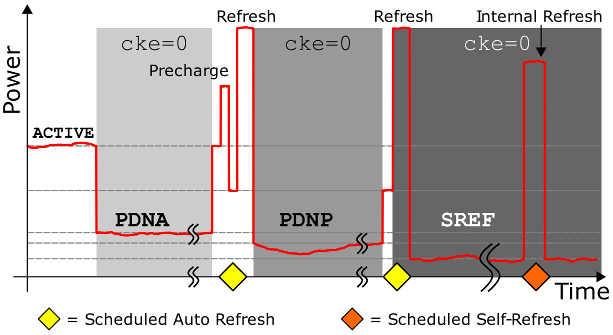


Fig. 11 Staggered power-down sequence.

This higher power consumption can be explained by the fact that each self-refresh entry provokes at the beginning a normal refresh command (see Fig. 10). The increase in DRAM energy consumption was already measured and investigated in Ref. [42]. It presented the overestimation of power savings in the Micron's power calculator [26], when using the DRAM self-refresh mode intensively. However, it was not analysed how to mitigate that issue, nor which power-down mode strategy could be implemented in order to achieve higher energy efficiency in a general way.

We see the power saving potential depends on the duration of each mode. Also the prolongation of execution times of certain applications must be considered when using power-down modes heavily. This is due to non-zero power-down exit times, especially the self-refresh exit time can be several clock cycles (DDR3 = 512, WIDE I/O = 20). State-of-the-art DRAM controllers use either a combination of PDNP and PDNA or SREF and they issue the power-down commands after configurable timeouts.

Our proposed optimised power-down policy [43] considers all three different power-down modes in order to achieve the maximum saving in energy and the minimum in slow-down on the execution of applications. This policy is based on a *staggered* approach.

Figure 11 shows this strategy with open-page policy. After a read or write access the DRAM stays in active mode (at least one bank active) and if no new transaction is scheduled, the controller immediately sets cke to "0" and the DRAM is entering active power-down mode (PDNA). If after a certain time a refresh is issued to the DRAM, the controller switches to precharge power-down mode (PDNP), because all banks have to be precharged before refreshing the DRAM. If there is still no new read or write request and the next refresh should be triggered, the controller performs instead of a normal refresh command a self-refresh entry. This consists of a refresh command and additionally the clock enable is de-asserted (cke=0).

This basic sequence is the key to the additional savings with our proposed *staggered* power-down policy, as the controller uses the DRAM state changes from the refresh command (PDNA→PDNP→SREF) to minimise the energy consumption of the DRAM. With this method, unnecessary SREF entries will be avoided, and the hardware timeout counters, as used in state-of-the-art controllers, are not required anymore.

In close-page policy, where after each write or read the respective bank is closed immediately (with auto-precharge), the active power-down mode (PDNA) is not needed. However, we achieved

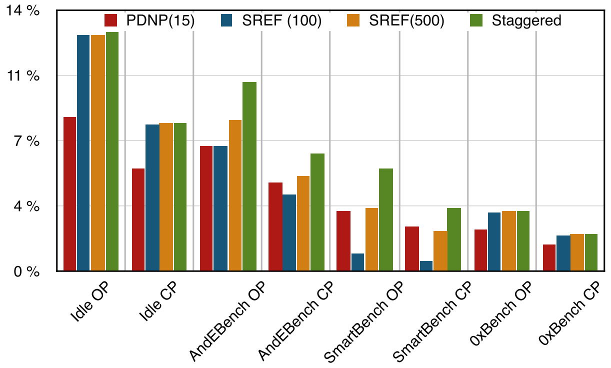


Fig. 12 Comparison of energy savings normalised to completely disabled power-down. The numbers 15, 100 and 500 represent the timeout in clock cycles.

in close-page policy energy savings as well. This is due to the fact that the DRAM controller waits until a refresh occurs and then enters self-refresh without an energy penalty. The performance impact for WIDE I/O DRAMs is low (20 clock cycles ≈ refresh cycle time (t_{RFC}) + 10 ns) [5], as there is no DLL (Delay Locked Loop), which needs to lock after self-refresh exit. Consequently, WIDE I/O DRAMs are ideal candidates to show the advantage of the staggered power-down policy. The TLM model of DRAMSys implements the traditional time-out based policy as well the staggered approach. **Figure 12** depicts the energy savings in percent. It shows that our staggered power-down mode policy is superior to any other methods. We see up to 10% energy savings in active benchmark execution and up to 13% in the idle phase with short activity bursts. The savings compared to the other power-down methods diminish with increased density of the executed benchmarks [44], [45], [46], such as 0xBench. Due to the high locality of all traces the close-page policy causes additional energy overhead (increased number of `ACT`s). In traces with longer idle periods `SREF` and our staggered approach converge, because there are only a few interruptions of the self-refresh periods.

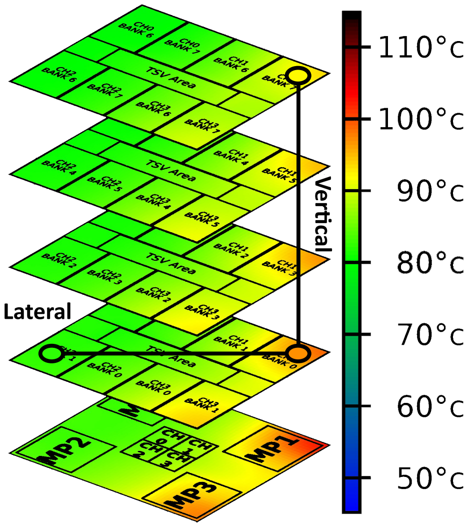


Fig. 13 Thermal simulation of an MPSoC with WIDE I/O DRAM[*1].

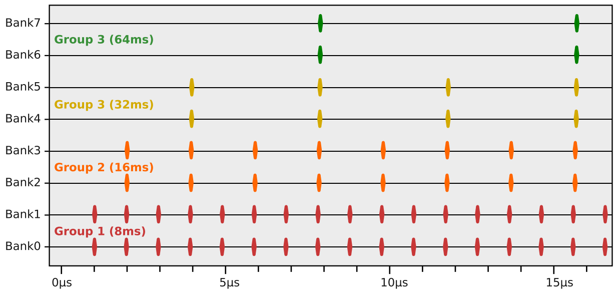


Fig. 14 Various refresh periods on different DRAM banks.

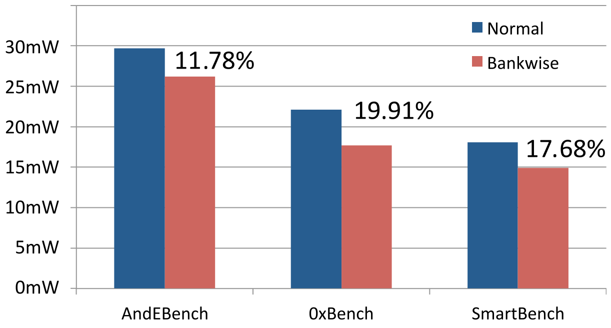


Fig. 15 Normal vs. Bankwise refresh power consumption.

4.3 Bankwise Refresh

In Ref. [31] we performed a statistical analysis on the temperature profile in a 3D MPSoC with 8 CPU cores and WIDE I/O DRAM. For this task we used the closed loop thermal simulation shown in Section 2.3. We measured lateral and vertical temperature variations in the 3D structure as shown in **Fig. 13**. For instance, with AndEBench [44], when all eight CPU cores are running at 1.4 GHz, an averaged vertical temperature variation of 5.6°C can be seen across four DRAM dies. In the first DRAM die, the averaged lateral temperature difference between two adjacent DRAM banks of the same channel is 3.3°C. When the averaged DRAM die temperature is > 85°C, the mentioned lateral and vertical temperature variations cause significant differences in the required refresh rate of each DRAM bank (< 64 ms).

Due to these observations, we implemented the following key idea: instead of defining the refresh rate based on the maximum temperature seen across the entire channel and refreshing all DRAM banks at the same rate, we select the refresh rate of each bank separately based on its own maximum temperature. **Figure 14** shows different refresh periods on several banks, for instance, bank 0 and bank 1 have a refresh period of 8 ms, which results in a refresh command issue every ≈ 980ns to refresh all 8192 rows of the bank. We have extended DRAMSys to support handling of separate per bank refresh commands. This increases the overall refresh period (makes refreshes happen less frequently) and improves the power consumption, as shown in **Fig. 15**.

4.4 Bankwise Staggered Power-Down

The previously presented techniques staggered power-down and bankwise refresh seem to be contradicting. The bankwise refresh strategy tries to reduce the number of refreshes per bank, but the staggered power-down needs a non-bankwise refresh on all banks as trigger for switching the power-down states. How-

[*1] A video of the simulation can be found on YouTube: http://www.youtube.com/watch?v=Eacsq71hHtY

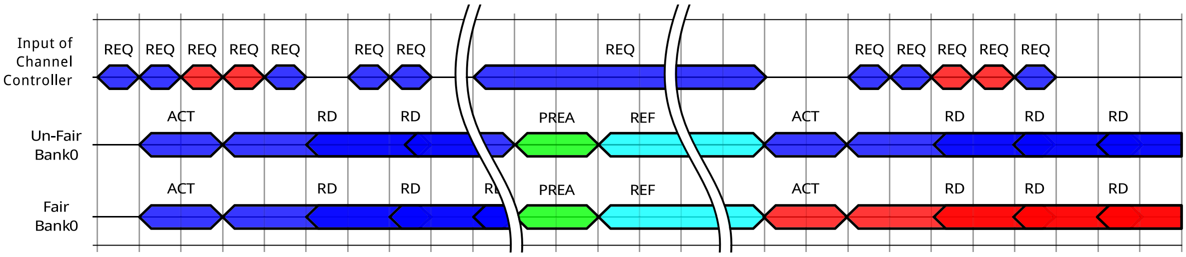


Fig. 17 Refresh aware scheduling.

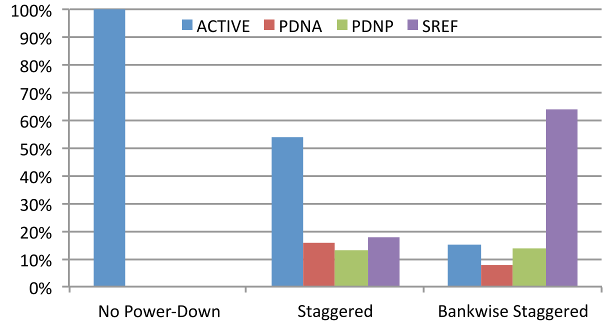


Fig. 16 Power-down usage for different power-down strategies.

Table 1 Staggered bankwise powerdown.

	No Power-Down	Staggered	Bankwise Staggered
Avg. Power	69.76 mW	65.00 mW	60.53 mW
Avg. Latency	29.80 ns	50.90 ns	46.30 ns

ever, both techniques can be combined when the DRAM is able to power down the banks independently.

We run a representative trace (chstone-mips) in three different modes (no power-down, staggered, bankwise staggered) to quantify the impacts for the staggered bankwise power-down approach. **Figure 16** shows the power-down usage of the different strategies. We see that the active periods over all banks are largely reduced (down to 14%) while using bankwise staggered power-down. Contrary, the time the DRAM banks are in SREF increases to 63%.

While a DRAM bank is in SREF another bank can operate on the interface (ACTIVE). Due to this behaviour, the expected power savings are limited, since the I/O part of the DRAM device contributes significantly to the overall power consumption. In **Table 1** the average power and request latency of the three modes are shown. We see for the bankwise staggered power-down an improvement in average power of 13.4% and 7.9% compared to no power-down and staggered, respectively. Additionally, the average request latency is recovered by 9.1% compared to the staggered policy.

4.5 Refresh Aware Scheduling

As mentioned before, there is a trend of increasing refresh rates in DRAM due to higher densities [13] and higher temperatures for 3D-integrated devices [31]. Higher refresh rates impact largely the decissions made by the DRAM scheduler. Current DRAM schedulers are based on the *First Ready First Come First Served* (FR-FCFS) algorithm [21] and are not aware of the point in time when the refresh happens. This can lead to a large unfairness with respect to different threads.

Table 2 Policy violations experiment with DDR4.

Benchmarks	Refresh Interval			
	64 ms	32 ms	16 ms	8 ms
chstone-aes and chstone-motion	30%	23%	30%	27%
chstone-jpeg and chstone-motion	5%	13%	5%	14%
chstone-motion and mb-adpcm-dec	11%	18%	17%	10%
chstone-aes and mb-h236-dec	1%	1%	3%	8%

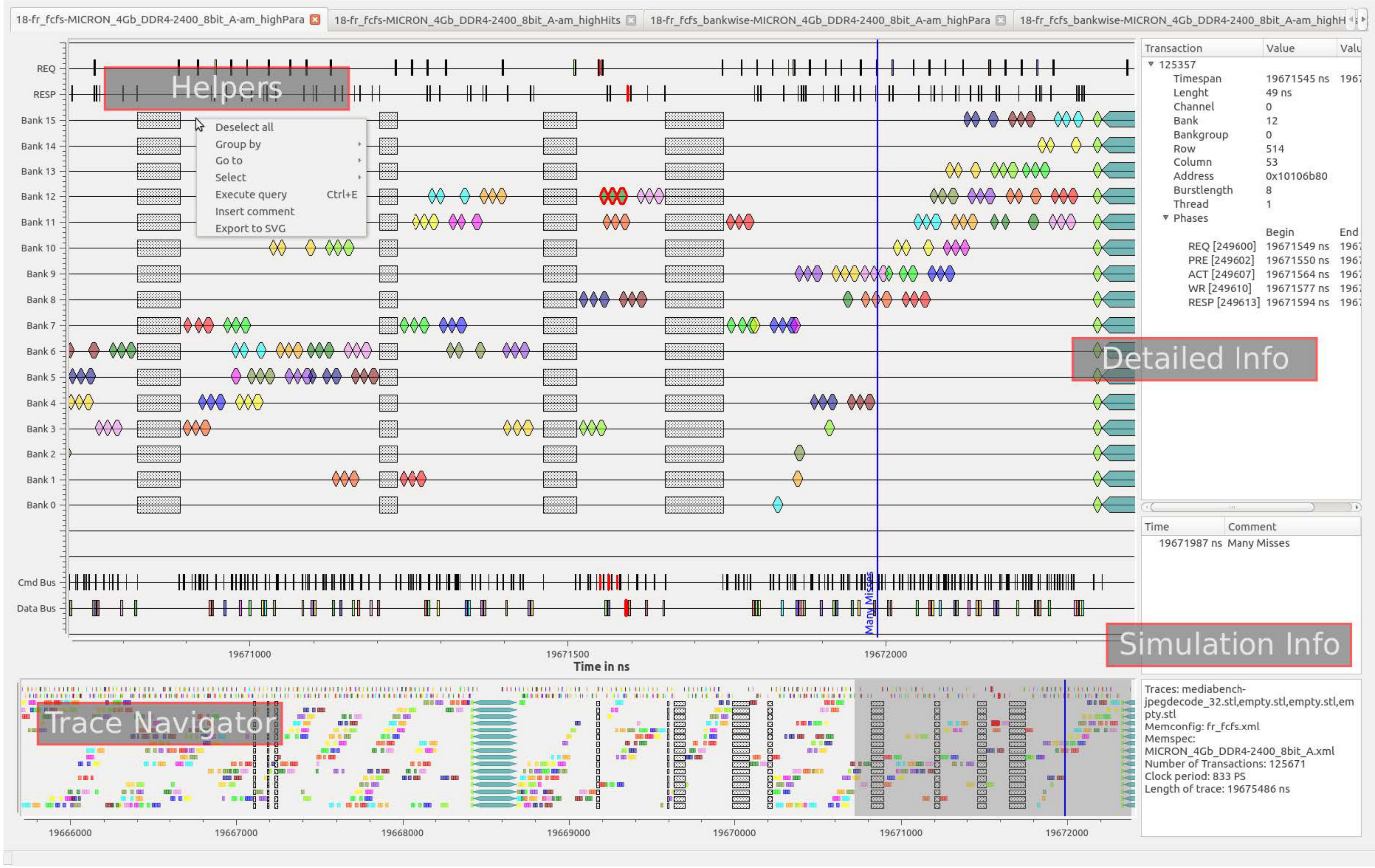
The FR-FCFS scheduler places incoming requests into a queue in such a way that they are placed next to requests that target the same row. By using this strategy, groups of row hits are formed (*row-hit-first* policy). If there are no row hits in the queue of the scheduler, the oldest request in the scheduler will be issued (*oldest-first* policy).

Whenever a refresh command (REF) is executed, the banks of the DRAM are precharged (closed row buffer) because of the precharge all command (PREA) that must be executed before each refresh. However, the scheduler is not aware of the point in time when the next refresh happens. Although the row is closed, the same row is re-opened (ACT) to finish a group of row hits scheduled before the refresh happened, even if there are requests in the scheduler that arrived earlier. In this situation the scheduler violates the oldest-first policy.

An example for such a violation can be seen in **Fig. 17**. There are two threads (blue and red) accessing the DRAM controller. The blue and the red threads are always accessing row 0 and row 1, respectively. In the scenario *Un-Fair* it can be observed that after the refresh, the requests of the blue thread are still prioritised over the requests of the red thread, since the scheduler was not aware of the refresh and followed the row-hit-first policy. **Table 2** shows the relative number of policy violations after a refresh for several examples.

Such violations can have a large impact on overall system performance. When an application keeps generating row hits, request from other applications will have to wait because of the row-hit-first policy. The applications will not be able to make progress at that point and system throughput decreases. However, after a refresh, older requests should be serviced, thereby allowing their threads to continue. The refreshes can actually be exploited to re-establish fairness between the threads.

The requirement for a refresh aware scheduler is that the time of a refresh event must be propagated from the controller backend to the frontend (a priori information), so that the scheduler is informed about refreshes and can use them to service outstanding requests from older threads, as shown in Fig. 17 in the scenario *Fair*. This refresh aware policy can also be applied to more recent DRAM schedulers like Refs. [22] and [23] and can also be used



```
Transaction          Value          Valu
▾ 125357
    Timespan         19671545 ns    1967
    Lenght           49 ns
    Channel          0
    Bank             12
    Bankgroup        0
    Row              514
    Column           53
    Address          0x10106b80
    Burstlength      8
    Thread           1
  ▾ Phases
                     Begin          End
      REQ [249600]   19671549 ns    1967
      PRE [249602]   19671550 ns    1967
      ACT [249607]   19671564 ns    1967
      WR [249610]    19671577 ns    1967
      RESP [249613]  19671594 ns    1967
```

```
Time         Comment
19671987 ns  Many Misses
```

```
Traces: mediabench-
jpegdecode_32.stl,empty.stl,empty.stl,em
pty.stl
Memconfig: fr_fcfs.xml
Memspec:
MICRON_4Gb_DDR4-2400_8bit_A.xml
Number of Transactions: 125671
Clock period: 833 PS
Length of trace: 19675486 ns
```

Fig. 18 DRAMSys trace analyser.

in combination with the previously presented bank-wise refresh.

5. Related Work

When it comes to high-level simulations of DRAM subsystems (DRAM and controller) one of the most cited DRAM system analysis tool is DRAMSim, available as DRAMSim2 [47]. DRAMSim2 is a cycle accurate model written in C++ of a DRAM memory controller, the DRAM modules, which comprise system storage, and the buses (channels) by which they communicate. DRAMSim2's goal is to be small, portable and accurate with a simple interface. However, this simplicity has a negative impact on the DRAM controller behaviour, which is not comparable to state-of-the-art controllers, such as Cadence's DDR controller [40] or others. Additionally, DRAMSim2 is a cycle accurate simulator that slows down event-driven full-system simulations. Moreover, DRAMSim2 misses an implementation of a read reorder buffer (ROB) and has currently no support for WIDE I/O and DDR4 DRAMs.

Another simulator is USIMM from the University of Utah and Intel Corp. [48], which is a simulation infrastructure that models the memory system and interfaces it with a trace-based processor model and a memory scheduling algorithm. Its focus is mainly memory scheduling not modelling of DRAM subsystem architectures. Both DRAMsim2 and USIMM have as far as we know neither error models nor thermal management possibilities integrated.

Gem5 [18], a full-system simulator has recently integrated a complete DRAM controller model [49]. This is very similar to the one implemented in DRAMsys as it uses events to trigger the simulation submodules and to execute the active tasks. Gem5 uses DRAMpower [28] as pre-compiled library and is capable to playback traces as well. However, in the current releases of Gem5 neither DRAM power-down modes nor error modelling are implemented. Thermal management capabilities are in the planning phase for Gem5. Moreover, Gem5 is not implemented in SystemC TLM, thus it cannot be easy attached to commercial tools such as Synopsys Platform Architect [20] or Cadence VSP [50].

A TLM based DRAM model is available from OCP-IP [51]. In contrast to our implementation it uses a clock based calculation of state and delay of DRAM and controller, which leads to an increase in simulation time. The commercial DesignWare TLM Library [52] from Synopsys and Sonics' MemMax Memory Subsystem [53] include AT DDR3 memory controller models that are not changeable and they do not disclose any details.

In contrast to these simulators and tools, the holistic DRAMSys design space exploration framework offers advanced analysis and debugging capabilities. These and the extensible infrastructure permit the exploration and development of new DRAM subsystem architectures and integration of emerging memory technologies.

6. Conclusion

In this paper, we presented DRAMSys, a design exploration framework that considers various design key parameters and aspects ranging from functional, over power and error modelling, to thermal closed-loop simulations. Only this holistic and modular approach embedded into our framework permits the thorough evaluation and characterisation of DRAM subsystems. Moreover, it enables the exploration and implementation of future memory systems and allows integrating new emerging memory technologies as well. We demonstrated in several examples the advantages of our proposed framework. These examples show in different use cases how the modelling, the analysis tools and the optimisation steps interact together to provide improved results.

New memory types, such as emerging resistive RAMs, will play an important role in future memory systems. 3D-integration allows merging all these memories into a single heterogeneous memory cube. However, new challenges arise, such as the modelling of these memory systems, the efficient control to achieve the maximum bandwidth and energy efficiency for a given application and thermal issues due to the limited heat removal. In the future we will couple our framework with gem5 and integrate different types of memories by using the presented TLM methodology.

Acknowledgments The authors thank the companies DOCEA Power and Synopsys for their great support. The authors also thank the involved students: Janik Schlemminger, Robert Gernhardt, Eder Zulian, Patrick Andres, Peter Ehses and Omar Naji for their valuable contributions to the project. 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.

References

[1] Wulf, W.A. and McKee, S.A.: Hitting the memory wall: implications of the obvious, *SIGARCH Comput. Archit. News*, (online), DOI: 10.1145/216585.216588 (1995).

[2] Jedec Solid State Technology Association: DDR3 SDRAM (JESD 79-3) (2012).

[3] Jedec Solid State Technology Association: DDR4 SDRAM (JESD 79-4) (2012).

[4] Krueger, J., Donofrio, D., Shalf, J., Mohiyuddin, M., Williams, S., Oliker, L. and Pfreundt, F.-J.: Hardware/software co-design for energy-efficient seismic modeling, *High Performance Computing, Networking, Storage and Analysis (SC), 2011 International Conference for*, pp.1–12 (2011).

[5] Jedec Solid State Technology Association: Wide I/O Single Data Rate (JESD 229) (2011).

[6] Hybrid Memory Cube Consortium: Hybrid Memory Cube Specification (2013).

[7] Weis, C., Loi, I., Benini, L. and Wehn, N.: Exploration and Optimization of 3-D Integrated DRAM Subsystems, *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, Vol.32, No.4, pp.597–610 (2013).

[8] Zhang, T., Xu, C., Chen, K., Sun, G. and Xie, Y.: 3D-SWIFT: A High-performance 3D-stacked Wide IO DRAM, *Proc. 24th Edition of the Great Lakes Symposium on VLSI, GLSVLSI'14*, pp.51–56, New York, NY, USA, ACM (online), DOI: 10.1145/2591513.2591529 (2014).

[9] Weis, C., Loi, I., Benini, L. and Wehn, N.: An energy efficient DRAM subsystem for 3D integrated SoCs, *Proc. DATE 2012* (2012).

[10] Gomony, M.D., Weis, C., Akesson, B., Wehn, N. and Goossens, K.: DRAM Selection and Configuration for Real-Time Mobile Systems, *IEEE Conference Design, Automation and Test in Europe (DATE)*, Dresden, Germany (2012).

[11] Pawlowski, J.T.: Hybrid Memory Cube, HotChips 23 (2011).

[12] Hamamoto, T., Sugiura, S. and Sawada, S.: On the retention time distribution of dynamic random access memory (DRAM), *IEEE Trans. Electron Devices*, Vol.45, No.6, pp.1300–1309 (1998).

[13] Liu, J., Jaiyen, B., Veras, R. and Mutlu, O.: RAIDR: Retention-Aware Intelligent DRAM Refresh, *Proc. 39th Annual International Symposium on Computer Architecture, ISCA'12*, pp.1–12, Washington, DC, USA, IEEE Computer Society (2012). (online), available

from ⟨http://dl.acm.org/citation.cfm?id=2337159.2337161⟩.

[14] Kesel, F.: *Modellierung von digitalen Systemen mit SystemC: Von der RTL- zur Transaction-Level-Modellierung*, Oldenbourg Wissenschaftsverlag (2012).

[15] IEEE Computer Society: *IEEE 1666: SystemC Language Reference Manual*, 2012 edition (2011).

[16] Black, D., Donovan, J., Bunton, B. and Keist, A.: *SystemC: From the Ground Up, Second Edition*, Springer US (2009).

[17] Kogel, T.: Generating Workload Models from TLM-2.0-based Virtual Prototypes for Efficient Architecture Performance Analysis, available from ⟨http://www.nascug.org/events/13th/tlm20_workload_models.pdf⟩ (2010), (accessed 2015-02-19).

[18] Binkert, N., Beckmann, B., Black, G., Reinhardt, S.K., Saidi, A., Basu, A., Hestness, J., Hower, D.R., Krishna, T., Sardashti, S., Sen, R., Sewell, K., Shoaib, M., Vaish, N., Hill, M.D. and Wood, D.A.: The gem5 simulator, *SIGARCH Comput. Archit. News*, Vol.39, No.2, pp.1–7 (online), DOI: 10.1145/2024716.2024718 (2011).

[19] Burger, D. and Austin, T.M.: The SimpleScalar tool set, version 2.0, *SIGARCH Comput. Archit. News*, Vol.25, No.3 (online), DOI: 10.1145/268806.268810 (1997).

[20] Synopsys, Inc: Synopsys Virtual Prototyping Solution, available from ⟨http://www.synopsys.com/Systems/VirtualPrototyping/Pages/default.aspx⟩ (2015), (accessed 2015-01-20).

[21] Rixner, S., Dally, W.J., Kapasi, U.J., Mattson, P. and Owens, J.D.: Memory Access Scheduling, *Proc. 27th Annual International Symposium on Computer Architecture, ISCA'00*, pp.128–138, New York, NY, USA, ACM (online), DOI: 10.1145/339647.339668 (2000).

[22] Mutlu, O. and Moscibroda, T.: Parallelism-Aware Batch-Scheduling: Enhancing both Performance and Fairness of Shared DRAM Systems, *35th International Symposium on Computer Architecture* (*ISCA*), Association for Computing Machinery, Inc. (2008). (online), available from ⟨http://research.microsoft.com/apps/pubs/default.aspx?id=79626⟩.

[23] Ausavarungnirun, R., Chang, K.K.-W., Subramanian, L., Loh, G.H. and Mutlu, O.: Staged Memory Scheduling: Achieving High Performance and Scalability in Heterogeneous Systems, *Proc. 39th Annual International Symposium on Computer Architecture, ISCA'12*, pp.416–427, Washington, DC, USA, IEEE Computer Society (2012). (online), available from ⟨http://dl.acm.org/citation.cfm?id=2337159.2337207⟩

[24] Jung, M., Weis, C., Wehn, N. and Chandrasekar, K.: TLM modelling of 3D stacked wide I/O DRAM subsystems: A virtual platform for memory controller design space exploration, *Proc. 2013 Workshop on Rapid Simulation and Performance Evaluation: Methods and Tools, RAPIDO'13*, pp.5:1–5:6, New York, NY, USA, ACM (online), DOI: 10.1145/2432516.2432521 (2013).

[25] MediaBench Consortium: Mediabench, available from ⟨http://euler.slu.edu/~fritts/mediabench/⟩ (2015), (accessed 2015-02-28).

[26] Micron: DDR3 SDRAM System Power Calculator (2011), (accessed 2014-07-03).

[27] Chandrasekar, K., Akesson, B. and Goossens, K.: Improved Power Modeling of DDR SDRAMs, *Proc. DSD'11*, (online), DOI: 10.1109/DSD.2011.17 (2011).

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

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

[30] DOCEA Power: AceThermalModeler and Aceplorer, available from ⟨http://www.doceapower.com⟩ (2014), (accessed 2015-01-20).

[31] Sadri, M., Jung, M., Weis, C., Wehn, N. and Benini, L.: Energy Optimization in 3D MPSoCs with Wide-I/O DRAM Using Temperature Variation Aware Bank-Wise Refresh, *Design, Automation and Test in Europe Conference and Exhibition* (*DATE*)*, 2014*, pp.1–4 (online), DOI: 10.7873/DATE2014.294 (2014).

[32] Gimmler-Dumont, C. and Wehn, N.: A Cross-Layer Reliability Design Methodology for Efficient, Dependable Wireless Receivers, *ACM Trans. Embedded Computing Systems*, Vol.13, No.4S, pp.1–29 (2014).

[33] Weis, C., Jung, M., Ehses, P., Santos, C., Vivet, P., Goossens, S., Koedam, M. and Wehn, N.: Retention Time Measurements and Modelling of Bit Error Rates of WIDE I/O DRAM in MPSoCs, *Proc. Conference on Design, Automation & Test in Europe*, European Design and Automation Association (2015).

[34] Hipp, D.R., Kennedy, D. and Mistachkin, J.: SQLite, available from ⟨http://www.sqlite.org⟩ (accessed 2015-02-18).

[35] van Rossum, G., Kuchling, A.M. and L. Drake Jr., F.: Python Reference Manual, available from ⟨https://www.python.org⟩ (2014), (accessed 2015-02-23).

[36] Zhang, Z., Zhu, Z. and Zhang, X.: A Permutation-based Page Interleaving Scheme to Reduce Row-buffer Conflicts and Exploit Data Locality, *Proc. 33rd Annual ACM/IEEE International Symposium on Microarchitecture, MICRO 33*, pp.32–41, New York, NY, USA, ACM (online), DOI: 10.1145/360128.360134 (2000).

[37] Lin, W.-F., Reinhardt, S. and Burger, D.: Reducing DRAM latencies with an integrated memory hierarchy design, *The 7th International Symposium on High-Performance Computer Architecture, 2001, HPCA*. pp.301–312 (online), DOI: 10.1109/HPCA.2001.903272 (2001).

[38] Xilinx, Inc.: Memory Interface Generator (MIG), available from ⟨http://www.xilinx.com/products/intellectual-property/mig.html⟩ (2015), (accessed 2015-02-18).

[39] Synopsys, Inc.: DesignWare DDR IP, available from ⟨http://www.synopsys.com/IP/InterfaceIP/DDRn/Pages/default.aspx⟩ (2015), (accessed 2015-02-18).

[40] Cadence Inc.: Cadence® Denali® DDR Memory IP, available from ⟨http://ip.cadence.com/ipportfolio/ip-portfolio-overview/memory-ip/ddr-lpddr⟩ (2014), (accessed 2015-02-18).

[41] Bojnordi, M.N. and Ipek, E.: PARDIS: A Programmable Memory Controller for the DDRx Interfacing Standards, *SIGARCH Comput. Archit. News*, Vol.40, No.3, pp.13–24 (online), DOI: 10.1145/2366231.2337162 (2012).

[42] Schmidt, D. and Wehn, N.: DRAM Power Management and Energy Consumption: A Critical Assessment, *Proc. 22nd Annual Symposium on Integrated Circuits and System Design*, No.32, Natal, Brazil (2009).

[43] Jung, M., Weis, C., Wehn, N., Sadri, M. and Benini, L.: Optimized active and power-down mode refresh control in 3D-DRAMs, *2014 22nd International Conference on Very Large Scale Integration* (*VLSI-SoC*), pp.1–6 (online), DOI: 10.1109/VLSI-SoC.2014.7004159 (2014).

[44] EEMBC Inc.: AndEBench: An EEMBC Benchmark for Android Devices, available from ⟨http://www.eembc.org/andebench/about.php⟩ (2013), (accessed 2015-02-18).

[45] 123 SmartMobile: SmartBench: A multi-core friendly benchmark application, available from ⟨play.google.com/store/apps/details?id=com.smartbench.eleven⟩ (2011), (accessed 2015-02-18).

[46] Google Inc.: 0xBench: Comprehensive Benchmark Suite for Android (2013).

[47] Rosenfeld, P., Cooper-Balis, E. and Jacob, B.: DRAMSim2: A Cycle Accurate Memory System Simulator, *Computer Architecture Letters*, Vol.10, No.1, pp.16–19 (online), DOI: 10.1109/L-CA.2011.4 (2011).

[48] Chatterjee, N., Balasubramonian, R., Shevgoor, M., Pugsley, S., Udipi, A., Shafiee, A., Sudan, Kshitij amd Awasthi, M. and Chishti, Z.: USIMM: The Utah SImulated Memory Module, A Simulation Infrastructure for the JWAC Memory Scheduling Championship, Utah and Intel Corp. (2012).

[49] Hansson, A., Agarwal, N., Kolli, A., Wenisch, T. and Udipi, A.: Simulating DRAM controllers for future system architecture exploration, *2014 IEEE International Symposium on Performance Analysis of Systems and Software* (*ISPASS*), pp.201–210 (online), DOI: 10.1109/ISPASS.2014.6844484 (2014).

[50] Cadence, Inc: Virtual System Platform, available from ⟨http://www.cadence.com/products/sd/virtual_system/pages/default.aspx⟩ (2015), (accessed 2015-01-20).

[51] Li, N., Wang, Y., Zhou, H. and Liang, L.: Design and Implementation of an Accurate Memory Subsystem Model in SystemC, Technical report (2010).

[52] Synopsys Inc.: DesignWare TLM Library, available from ⟨http://www.synopsys.com/Systems/VirtualPrototyping/VPModels/Pages/DW-TLM-Library.aspx⟩.

[53] Sonics: MemMax-Scheduler, available from ⟨http://sonicsinc.com/products/memory-subsystems/memmax-scheduler/⟩.

Matthias Jung reveived his Diploma degree in electrical engineering from the University of Kaiserslautern, Kaiserslautern, Germany in 2011. Since this time, he is pursuing his Ph.D. in the Microelectronic Systems Design Research Group of the University of Kaiserslautern. His research interest are SystemC based virtual prototypes, especially with the focus on the modelling of memory systems.

Christian Weis received his Ph.D. degree in electrical engineering from the University of Kaiserslautern, Kaiserslautern, Germany, in 2014. From 1996 to 1998, he was with Mitsubishi Semiconductor Europe, Germany, where he was engaged in the development of microcontrollers. From 1998 to 2009, he was with Siemens Semiconductor, Infineon Technologies AG and Qimonda AG, Munich, Germany, in DRAM design. During this time frame, he was involved in DRAM design for graphics and commodity DRAM products. In 2006, he was a Design Team Leader for the 1Gb DDR3 DRAM, the first DDR3 volume product at Infineon/Qimonda. Since 2009, he has been with the Microelectronic System Design Research Group, University of Kaiserslautern, Kaiserslautern, Germany. He holds several patents related to DRAMs and DRAM design. His current research interests include 3D integrated DRAMs, DRAM controller design, and heterogeneous memories and MPSoCs.

Norbert Wehn holds the chair for Microelectronic System Design in the department of Electrical Engineering and Information Technology. He is also the Vice President at the University of Kaiserslautern and has more than 300 publications in various fields of microelectronic system design. He holds several patents and has founded two Start-Ups. He served among others as program chair for DATE 2003, general chair for DATE 2005 and general Co-Chair at FPL 2014. He is associate editor of various journals and member of several scientific advisory boards. His special research interests are VLSI-architectures for mobile communication, forward error correction techniques, advanced SoC architectures and methodologies, 3D integration, reliability issues in SoC and high performance computing on embedded devices.

(Invited by Editor-in-Chief: *Hiroyuki Tomiyama*)