Using Run-Time Reverse-Engineering to Optimize DRAM Refresh

Deepak M. Mathew
University of Kaiserslautern
Erwin-Schrödinger-Straße 12
Kaiserslautern, Germany 67663
deepak@eit.uni-kl.de

Éder F. Zulian
University of Kaiserslautern
Erwin-Schrödinger-Straße 12
Kaiserslautern, Germany 67663
zulian@eit.uni-kl.de

Matthias Jung
Fraunhofer IESE
Fraunhofer-Platz 1
Kaiserslautern, Germany 67663
matthias.jung@iese.fraunhofer.de

Kira Kraft
University of Kaiserslautern
Erwin-Schrödinger-Straße 12
Kaiserslautern, Germany 67663
kraft@eit.uni-kl.de

Christian Weis
University of Kaiserslautern
Erwin-Schrödinger-Straße 12
Kaiserslautern, Germany 67663
weis@eit.uni-kl.de

Bruce Jacob
University of Maryland
1333 A.V. Williams Building
College Park, MD, USA 20742
blj@umd.edu

Norbert Wehn
University of Kaiserslautern
Erwin-Schrödinger-Straße 12
Kaiserslautern, Germany 67663
wehn@eit.uni-kl.de

ABSTRACT

The overhead of DRAM refresh is increasing with each density generation. To help offset some of this overhead, JEDEC designed the modern *Auto-Refresh* command with a highly optimized architecture internal to the DRAM—an architecture that violates the timing rules external controllers must observe and obey during normal operation. Numerous refresh-reduction schemes manually refresh the DRAM row-by-row, eliminating unnecessary refreshes to improve both energy and performance of the DRAM. However, it has been shown that modern *Auto-Refresh* is incompatible with these schemes, that their manual refreshing of specified rows through explicit *Activate* and *Precharge* precludes them from exploiting the architectural optimizations available internally for *Auto-Refresh* operations.

This paper shows that various DRAM timing parameters, which should be followed during normal DRAM operations can be reduced for performing *Refresh* operation, and by reverse engineering those internal timing parameters at system-init time an external memory controller can use them in conjunction with individual *Activate* and *Precharge* commands, thereby reducing the performance overhead afforded *Auto-Refresh*, while simultaneously supporting row-by-row refresh reduction schemes.

Through physical experiments and measurement, we find that our optimized scheme reduces t_{RFC} by up to 45% compared to the already highly-optimized *Auto-Refresh* mechanism. It is also 10% more energy-efficient and 50% more performance-efficient than the non-optimized row-by-row refresh. Further evaluations done by simulating future 16 Gb DDR4 devices show how the reduction in t_{RFC} improves the application performance and energy efficiency. The proposed technique enhances all of the existing refresh-optimization schemes that use row-by-row refresh, and it does so without requiring any modification to the DRAM or DRAM protocol.

MEMSYS 2017, Alexandria, VA, USA

DOI: 10.1145/3132402.3132419

CCS CONCEPTS

• **Social and professional topics** → **Hardware reverse engineering**; • **Hardware** → **Dynamic memory**;

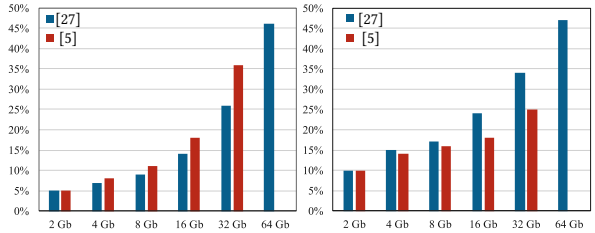
KEYWORDS

DRAM, Refresh, Row-Granular, Reverse Engineering

ACM Reference format:
Deepak M. Mathew, Éder F. Zulian, Matthias Jung, Kira Kraft, Christian Weis, Bruce Jacob, and Norbert Wehn. 2017. Using Run-Time Reverse-Engineering to Optimize DRAM Refresh. In *Proceedings of MEMSYS 2017, Alexandria, VA, USA, October 2–5, 2017,* 10 pages.
DOI: 10.1145/3132402.3132419

1 INTRODUCTION

As shown in Figure 1, DRAM refresh is expensive in both time and energy, and its overhead is getting worse: the costs grow linearly with capacity, which means exponentially with each density generation [5, 27]. Modern JEDEC SDRAMs use a special *Auto-Refresh* command that is opaque to the controller and that handles all *Refresh* operations and timing internally. To help offset some of the increasing refresh overheads, JEDEC designed *Auto-Refresh* with a highly optimized architecture internally—in particular, the architecture violates the inter-operation timing rules that external controllers must observe and obey during normal operation (e.g.,



(a) Impact on Performance **(b) Impact on Power**

Figure 1: Impact of Refresh vs. Device Size [5, 27]

for bank *Precharge*, row *Activate*, and/or column *Read*/ *Write* operations). The DRAM can violate external timing parameters internally because the internal mechanism refreshes numerous rows simultaneously (not just one at a time), there are no command/address bus constrains, and because, during refresh, it is understood that, unlike "normal" operation, a read or write operation will not follow the (multi-row) *Activate* operation.

There exists a large body of research, developing schemes that manually refresh the DRAM row-by-row, characterizing each row's ability to retain data and eliminating unnecessary *Refresh* operations on rows that can be refreshed less often. These schemes have been shown to be extremely effective, and, because eliminating refresh improves both energy and performance of the memory system, this offers the potential for significant gains in DRAM-system efficiency. However, these schemes are incompatible with the modern *Auto-Refresh* mechanism: they need to operate on a row-by-row basis, whereas *Auto-Refresh* operates on multiple rows at once. In addition, *Auto-Refresh* cannot skip any row, whether that row needs to be refreshed or not. Thus, the manual schemes use explicit row-level *Activate* (ACT) and *Precharge* (PRE) commands to refresh row-by-row, called *Row Granular Refresh* (RGR), and, because of this, studies have shown that these refresh schemes are unable to exploit the optimizations available internally through the *Auto-Refresh* mechanism. Previous work has proposed minimal alterations to the DRAM architecture and protocol, to allow both row-granular control of *Refresh* operations *and* the use of the internal optimizations. Previous work has also claimed it to be impossible to equal the performance and energy savings of optimized *Auto-Refresh* by using individual ACT and PRE commands, i.e. RGR. In this paper, we disprove this commonly held wisdom.

This paper shows that by reverse-engineering, at system initialization time, the DRAM's internal inter-operation timing parameters (which are otherwise opaque to the outside), an external memory controller can both discover and exploit the optimized timing used by *Auto-Refresh*. We show that an optimized refresh controller can use this reverse-engineered timing data to schedule individual row-level ACT and PRE commands, thereby achieving the best of both worlds: the controller can obtain the same energy and performance savings that the *Auto-Refresh* mechanism enjoys, while simultaneously supporting legacy RGR refresh-reduction schemes. By conducting physical experiments and taking physical measurements on 4 Gb x16 DDR3 SDRAMs, using our FPGA-based evaluation platform, we find that our optimized refresh-reduction scheme is actually *more* performance-efficient than the already highly-optimized *Auto-Refresh* mechanism.

In this paper we propose an optimized RGR refresh technique called *Optimized Row Granular Refresh* (ORGR), which can be implemented inside the memory controller without making any changes to the DRAM. In our invented technique we reduce four DRAM timing parameters while performing refresh:

- the time between ACT and PRE commands to the same bank (t_{RAS}),
- the time between two successive ACT commands (t_{RRD}), and
- the *Four-bank Activate Window* (t_{FAW}).
- the *the time for a* PRE *command to complete* (t_{RP}).

Furthermore we present a new run-time reverse engineering technique in order to find the reduced t_{RAS}.

Table 1 shows the relevant DRAM timings and currents for this paper. Soon after DRAM initialization and calibration, the memory controller performs certain tests to find out the vendor-specific minimum values for the above-mentioned timing parameters. The reduced timings are used only when performing the optimized *Refresh* operations, whereas we use the JEDEC-specified inter-command timings for all other operations (e.g., normal *Read*/*Write* operation). We make a number of observations:

- t_{RRD} and t_{FAW} can be reduced for *Refresh* operation since they are partially external and internal power network constraints, in order to reduce the peak power when many ranks perform a normal *Read*/*Write* operation in parallel, and because a *Read*/*Write* operation which would normally follow an *Activate* command will not be performed.
- Once we reduce t_{RRD} for performing row-by-row refresh, we hit the t_{RAS} limit, which prevents activating another row in the same bank. The parameter t_{RAS} can be reduced, since there is no voltage drop in the bitline, which usually happens due to *Read*/*Write* commands. Therefore, the bitlines will restore faster.
- DRAM vendors use a built-in analog timer (the t_{RAS}^{min} timer), which prevents precharging an already activated row before the minimum restoration time is over—this ensures that the data is not disrupted by an early *Precharge*. This counter is set to a much lower value than the JEDEC-specified t_{RAS}. Therefore, the limit for reducing t_{RAS} can be set dynamically by finding out the vendor specified t_{RAS}^{min} value.

We perform our advanced refresh technique on state-of-the-art DDR3 SDRAMs, and we show, in terms of performance and energy, the benefits of this technique compared to *Auto-Refresh* as well as with RGR using the standard JEDEC-specified timing (i.e., how it behaves without our optimizations). Among other findings, our measurements show that the optimized refresh-reduction scheme improves the t_{RFC} by up to 45% compared to *Auto-Refresh*, which as mentioned is already highly optimized. We also estimate the benefits of this technique for future DRAMs using our high-level simulation framework. The proposed technique enhances all of the existing refresh-optimization schemes that use RGR, and it does so without requiring any modification to the DRAM or DRAM protocol.

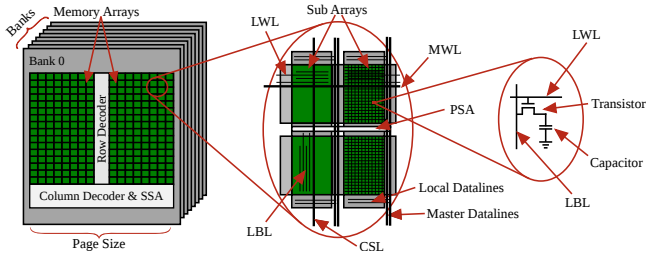


Figure 2: DRAM Architecture

2 BACKGROUND

First, we explain the basic DRAM architecture and functionality, to understand the limiting factors with respect to energy and bandwidth. As depicted in Figure 2, a DRAM device is organized as a set of memory banks (e.g. eight) that include memory arrays (e.g. two). Each memory array has row and column decoders, master wordline drivers and *Secondary Sense Amplifiers* (SSA). Buses, buffers, control signals, voltage regulators, charge pumps and other peripherals are shared between the different banks. The memory arrays are formed in a hierarchical structure out of *sub-arrays* (SA) for efficient wiring, increased speed and reduced power consumption. Therefore, each SA is equipped with *Primary Sense Amplifiers* (PSA). A typical memory SA consists of e.g. 512 cells × 512 cells = 256 Kb[1]. For instance, a 64 Mb DRAM bank is formed out of two memory arrays, where each memory array consists of 8 × 16 = 128 SAs. A single memory cell is built as a transistor capacitor pair where the data is stored in the capacitor as a charge. The individual cells in each sub-array are connected to *Local Wordlines* (LWL) and *Local Bitlines* (LBL). The LBLs and LWLs are connected to global *Master Bitlines* (MBL) and *Master Wordlines* (MWL), respectively, which span over the complete memory array. To read data from the memory, a precharge command is issued by the memory controler (PRE) to prepare the LBLs to a halfway voltage level and an activate command (ACT) is issued to drive the LWL high and transfer the charge between the memory cells and the connected LBLs. The voltage difference caused by this transfer of charge (data) is sensed by the PSAs, as shown in Figure 3 (More information about the sensing process will be given in Section 4). Then, read (RD) or write (WR) commands can be sent to read or write specific columns of data from or to SSAs, which are interacting with the I/Os. Once finished, the wordlines can be switched off, the cell capacitors disconnected, and the LBLs can be precharged again.

The combination of primary and secondary sense amplifiers of the memory arrays in one bank can be conceived as a *row buffer* that has usually a size ranging from 512 B to 8 KB (called DRAM page size, see Figure 2). It acts like a small cache that stores the most recently accessed row of the bank. The latency of a memory access to a bank largely varies depending on the state of this row buffer. If a memory access targets the same row as the currently cached row in the buffer (called *row hit*), it results in a low latency and low energy memory access. Whereas, if a memory access targets a different row as the current row in the buffer (called *row miss*), it results in higher latency and energy consumption: A precharge command (PRE) must be issued before the required row can be loaded (ACT) from the DRAM array into the row buffer. Accessing activated rows in two different banks has no penalty.

[1]We use JEDEC's notation for storage capacity: $K = 2^{10}$, $M = 2^{20}$, $G = 2^{30}$

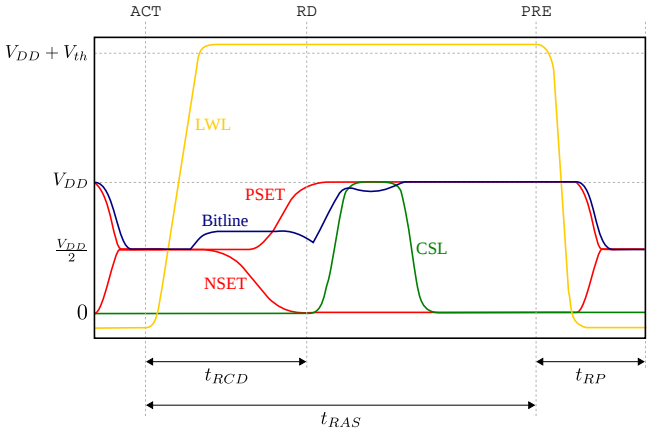


Figure 3: Sense Amplifier Voltage Waveform [9, 17]

Table 1: Key Parameters for a DDR3-1600 Device [17, 31]

Name	Explanation
t_{RCD}	*Row to Column Delay*: The time interval between row access and data ready at PSAs, in other words: The time interval between ACT and RD on the same bank.
t_{RP}	*Row Precharge*: The time interval that it takes for a DRAM array to be precharged (PRE) and prepared for another row access.
t_{RAS}	*Row Access Strobe*: The minimum active time for a row, in other words: The time interval between row access command and data restoration in a DRAM array.
t_{RC}	*Row Cycle*: The fastest time to ACT and PRE the same row ($t_{RC} = t_{RAS} + t_{RP}$), in other words: The time interval between accesses to different rows in a bank.
t_{WR}	*Write Recovery*: The minimum time interval between the end of a WR burst and a PRE command.
t_{FAW}	*Four Activate Window*: Only four ACT commands can be issued in this time window.
t_{RRD}	*Row-to-Row Delay*: The minimum time interval between two ACT commands to different banks.
t_{REF}	*Refresh Period*: The time period a DRAM cell must be refreshed (e.g. t_{REF} = 64 ms).
t_{REFI}	*Refresh Interval*: The time interval between two refresh commands (e.g. t_{REFI} = 7.8 μs).
I_{DD5}	*Refresh Current*: Measured during refresh operation, with REF commands issued every t_{RFC}.
V_{DD}	*Supply Voltage*

Furthermore, DRAM cells need to be refreshed periodically to retain the data stored in the cell capacitors. As shown in Section 1, the refresh overhead in terms of both, performance and energy, is increasing with each density generation. This is clearly visible from the increase in the JEDEC-specified *Auto-Refresh cycle time* (t_{RFC}) for denser DRAMs. In the following we describe both, *Auto-Refresh* and RGR.

2.1 Auto-Refresh in Modern DRAMs

A DRAM cell must be refreshed every refresh window t_{REF} = 64 ms to retain the data stored in it, at normal temperature conditions. Modern DRAMs are equipped with an *Auto-Refresh* (AREF) command to perform this operation. A single AREF command does not refresh the entire DRAM at once, but it refreshes only a certain number of rows in all banks, depending on the density of the DRAM

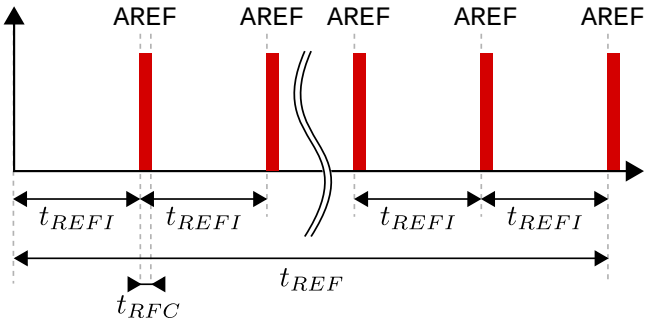


Figure 4: *Auto-Refresh* Commands and Timings

device. Therefore, the memory controller has to issue AREF commands at regular intervals, called *Refresh Interval* (t_{REFI}), to refresh the whole DRAM in a 64 ms Refresh Window t_{REF}. Notionally, the refresh interval can be calculated as

$$t_{REFI} = \frac{t_{REF} \cdot r}{R}, \tag{1}$$

where R is the total number of rows per bank and r the number of rows that an AREF command refreshes per bank. However, for DDR3 DRAMs, this refresh interval is always fixed by JEDEC to 7.8 μs [30] for all DRAM densities at normal operating temperatures, which means that r is variable and grows with DRAM density. It can be calculated as

$$r = \left\lceil \frac{t_{REFI} \cdot R}{t_{REF}} \right\rceil. \tag{2}$$

The number of refresh commands that have to be issued can be calculated as

$$N = \frac{R}{r}. \tag{3}$$

Figure 4 shows how the *Refresh* commands are issued in DDR3 DRAMs. In total, $N = 8192$ AREF commands must be issued in every t_{REF} window, in order to refresh the complete DRAM. The DRAM internally refreshes one or multiple rows per bank (r) in response to an AREF command. An AREF command blocks the DRAM for the *Refresh Cycle Time* (t_{RFC}) from performing other operations. The duration of t_{RFC} depends on r and it increases with the density of the DRAM. The increase in t_{RFC} affects the overall system performance as well as energy efficiency.

There exists a large body of research developing schemes that manually refresh the DRAM row-by-row, eliminating unnecessary refreshes to improve both energy and performance of the DRAM. In the following subsection we describe their operation.

2.2 Row Granular Refresh

While it is not specified how DRAM vendors perform the *Refresh* operation internally, it is an open secret that an internal *Refresh* operation is a sequence of ACT and PRE operations done in parallel on a set of banks (e.g. bank groups). An external DRAM controller can perform the similar operations to specific rows and banks to mimic the internal DRAM *Auto-Refresh*. Figure 5 shows such a RGR operation for a 2 Gb x16 DDR3 device. The device has $B = 8$ banks and $R = 2^{14}$ rows in total. Therefore, $r = 2$ rows in all banks have to be refreshed in every $t_{REFI} = 7.8\,\mu$s to refresh the entire DRAM in a $t_{REF} = 64$ ms refresh window. The memory controller has to

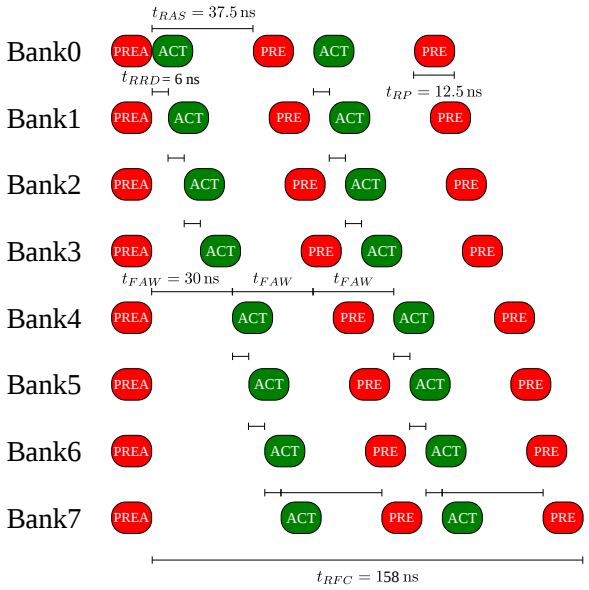


Figure 5: Example for RGR with 2Gbit x16 DDR3 Device with $r = 2$ and $B = 8$

follow all the JEDEC specified timings: t_{RRD}, t_{FAW}, t_{RAS} and t_{RP}. Since this technique follows the strict timings for t_{RRD} and t_{RAS}, each *Refresh* operation will take

$$\begin{aligned} t_{RFC} &= (r \cdot B - 1) \cdot t_{RRD} + t_{RAS} + t_{RP} \\ &+ \left(r \cdot \frac{B}{4} - 1\right) \cdot t_{wait} \\ &+ (r - 1) \cdot [t_{RAS} + t_{RP} - (B \cdot t_{RRD} + t_{wait})]_{\geq 0}, \end{aligned} \tag{4}$$

where $t_{wait} = [t_{FAW} - 4 \cdot t_{RRD}]_{\geq 0}$ and $[\cdot]_{\geq 0} = \max\{\cdot, 0\}$. The last two summands of the formula can be viewed as the time the DRAM controller has to wait in order not to violate the timing constraints: The second summand takes care to wait for t_{FAW} if $t_{FAW} > 4 \cdot t_{RRD}$ and the third summand ensures to wait for $t_{RAS} + t_{RP}$ whenever it is larger than $B \cdot t_{RRD} + t_{wait}$. For the DDR3 scenario in Figure 5, the DRAM is blocked for performing other operations for $t_{RFC} = 158$ ns (and additionally the time for the required PREA if needed), which reduces the performance and energy efficiency of the overall system. Our proposed ORGR uses reduced timings to perform the *Refresh* operations parallel in all banks thereby reducing the t_{RFC}. This technique is explained in Section 4.

3 RELATED WORK

Here we present the related work. For a detailed survey on today's refresh techniques we refer to [5].

3.1 Shortening DRAM timing Parameters

The characteristic parameters of DRAMs, such as timings (e.g. t_{RAS}) and currents (I_{DDX}), listed in datasheets are very pessimistic due to the high process margins added by the vendors to ensure correct functionality under worst-case conditions and a high-enough yield [8, 9]. Chandrasekar et al. [8] present a post-manufacturing

performance characterization methodology that identifies this excess in process-margins for any given DRAM device at runtime. This information can be used in order to optimize the access latencies. Similarly, [24] and [11] show mechanisms that adaptively reduce the important timing parameters (t_{RCD}, t_{RAS}, t_{WR} and t_{RP}). The authors of [40, 41] perform detailed circuit simulations and show that recently refreshed rows have more charge and therefore they can access the recently refreshed rows with reduced t_{RAS}. In the works of [50, 51], the write recovery time t_{WR} is reduced in order to gain more performance. Similarly, the t_{WR} is reduced in [49] by jeopardizing data reliability.

3.2 Selective Refresh

Several selective refresh techniques have been proposed in the last years. *PARIS* [3], *DTail* [13] and *ESKIMO* [16] exclude rows that do not store useful data from being refreshed. *Smart Refresh* [14] refreshes only rows that have not been accessed recently. *CREAM* [48], [39] and [12] show per-bank and per-subarray refresh techniques. The authors of [36] issue refresh commands according to the stored data values. *Flexible Auto-Refresh* [7] shows that DRAM *Auto-Refresh* is highly optimized and RGR cannot be as effective as *Auto-Refresh*, even if 70% of the rows are skipped. Therefore the authors presented a realistic implementation of a flexible and row selective refresh, which requires only small changes to the DRAM and its controller.

3.3 Refresh Scheduling

The idea of postponing DRAM refresh into self-refresh phases is presented in [6]. The authors of [33] show how to use JEDEC's DDR4 fine granularity refresh (FGR) efficiently, while [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* [34] 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].

3.4 Retention Aware Refresh

There are several works on retention aware refresh, that also try to reduce the number of refresh events by maintainig data integrity [3, 25, 27, 44, 45, 52]. These techniques can be applied in fields of non-resilient applications. The authors of [15] increase the refresh period and refresh weak cells with dedicated ACT and PRE commands. The CLARA scheme [2] proposes a circular linked-list based refresh architecture for improving auto- and self-refresh. The DRAM device must be modified, and it also requires a characterization of weak cells at system initialization. However, it is very difficult to characterize DRAMs, as they experience *Variable Retention Times* (VRTs) and *Data Pattern Dependencies* (DPD) [26, 46] 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. *AVATAR* [37], tries to overcome VRT issues by combining an online *Error-Correcting Code* (ECC) mechanism with row selective refresh.

3.5 Approximate DRAM

Recently, Approximate DRAM [19, 20] is discussed to lower the influence of DRAM refresh. Liu et al. presented the first work on Approximate DRAM, called *Flikker* [28], which reduces the number of refreshes by partitioning a DRAM bank in a critical and non-critical region. The non-critical region will be refreshed with a lower refresh rate. A similar approach is followed by [38] called *Quality Aware Approximate DRAM*, which characterizes the used DRAM by extensive retention time measurements. As a result the DRAM pages are sorted into quality bins. During allocation critical data is stored in high quality bins, whereas approximate data is allocated in low quality bins. The whole DRAM is then refreshed with the same refresh rate, which makes this approach applicable to today's DRAM devices, since no changes of the internal DRAM structure are required. However, this approach is very time consuming due to the prior characterization and characterization is very challenging because of the VRT phenomenon. Moreover, there is an overhead to store the essential information to apply this technique (sorted page order). 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. *Sparkk* [29] proposes the idea of permutation of the data bits on several DRAM chips that are refreshed with different rates. The most significant bits of a byte are located in a highly refreshed DRAM device and the least significant bits are stored in a less refreshed chip. *Omitting Refresh* (OR) [23] shows 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 is error resilient and can tolerate bit errors to some degree in a given time window.

In contrast to prior works [7], [12], [13], [36], [39], [48], our proposed technique does not require any modification to the DRAM. All of the previous research which uses RGR benefits directly from our technique. Furthermore, our method can be easily integrated to any DRAM controller and in any computing devices, performing a start-up calibration to obtain the minimum timings for the DRAM device used.

4 OPTIMIZED ROW GRANULAR REFRESH

In our proposed *Optimized Row Granular Refresh* (ORGR), we do a holistic optimization of the RGR by reducing multiple DRAM timing parameters: t_{RRD}, t_{RAS}, t_{RP}, and ignoring t_{FAW} (by setting it to $t_{FAW} = 4 \cdot t_{RRD}$). We define the new timing parameters as t^*_{RRD}, t^*_{RAS} and t^*_{RP}. The values of t_{RRD} and t_{FAW} can be reduced for *Refresh* operation since they are partially external and internal power network constraints. They serve to reduce the peak power when many ranks perform a normal *Read/Write* operation in parallel, and they are unnecessary here because a *Read/Write* operation, which would normally follow an *Activate* operation, will not be performed. We used a similar approach like showed in [11] for determining t^*_{RP}. The justification for t^*_{RAS} is explained below.

The sensing scheme for a normal *Read* operation is shown in Figure 3. The ACT command triggers the *Local Wordline* (LWL) drivers and they raise LWL to a voltage above $V_{DD} + V_{th}$. Bitlines develop a small charge, which is amplified by the *Primary Sense*

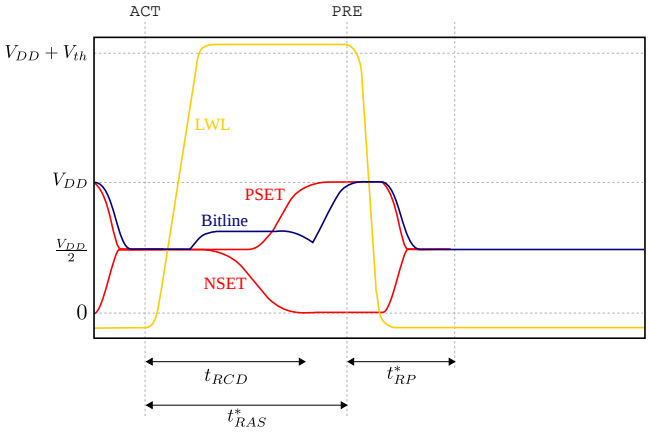


Figure 6: Sense Amplifier Voltage Waveform for ORGR

Amplifiers (PSA) to V_{DD}. When the bitline voltage is raised to approximately 92% of the V_{DD}, a RD command is issued and the *Column Select Line* (CSL) is raised. This will create a drop in the bitline voltage until the CSL is lowered, increasing the restoration time, which is reflected in the t_{RAS} specified by JEDEC. But, in case of a RGR, a RD does not follow an ACT, and therefore there is no voltage drop in the bitline, shown in Figure 6. This enables the fast restoration of bitlines, and therefore t_{RAS} can be reduced.

Thus, in ORGR we perform row granular refresh using reduced timings as shown in Figure 7. We use the reduced timing parameters $t^*_{RRD} = 3.75$ ns, $t^*_{RAS} = 20.625$ ns and $t^*_{RP} = 9.375$ ns. The time to perform a single *Refresh* using this method can be also calculated using Equation 4. With our reduced timings, we achieved 45% savings in t^*_{RFC} compared to non-optimized RGR.

Later, in Section 6, we validate by physical measurements that the reduction of timing parameters does not hurt the data stored in the DRAM, by varying the refresh rate and measuring the number of bit flips in actual DRAMs. The method to find the minimum t_{RAS} value, t^{min}_{RAS} is described in the following subsection.

4.1 Determination of Minimum Timings

Since t_{RAS} is the key parameter in our optimization strategy— violation of which will cause bit flips inside the DRAM—we have invented a method to find the minimum possible value of t_{RAS}: t^{min}_{RAS}, using reverse-engineering at run-time. We used the hardware platform described in Section 5 to perform the reverse-engineering. Figure 8 depicts the general reverse-engineering technique. We initially issue an ACT command, followed by a PRE command after t_{RAS} to the same bank: e.g Bank7 in the example in Fig. 8. The RD command which comes t_{RP} after PRE will be ignored by the DRAM since Bank7 is already precharged, and therefore the DRAM will not drive the data and the *data strobe* (DQS) in response, and thus the DQS will not be present on the bus. Then we gradually reduce the t_{RAS} by moving PRE closer to the ACT until the presence of DQS is detected by our experimental setup in response to the RD command, which indicates that the DRAM detects a correctly timed RD command, thereby confirming the value of the t^{min}_{RAS} counter inside the DRAM. When the PRE comes within the t^{min}_{RAS}, the DRAM ignores

Figure 7: Example for ORGR with 2Gbit x16 DDR3 Device with $r = 2$ and $B = 8$

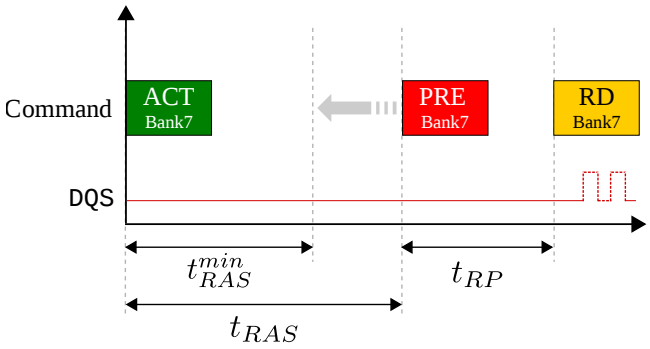


Figure 8: Determination of t^{min}_{RAS}

the precharge to prevent data restoration failure, and therefore the RD command which comes later will *Read* from the previously activated bank. Similarly we find the minimum t_{RAS} for different vendors. We utilize the pessimistic guard band set by the vendors for minimizing t_{RP}, and use similar techniques to find t^{min}_{RP}. We run these test-sets for t^{min}_{RAS} calibration from our controller soon after the DRAM initialization and calibration step is finished. Based on the detected t^{min}_{RAS} value, our controller decides the t^*_{RAS} for performing ORGR. Note that the vendor's internal counter already takes process variation effects into account. To account for any temperature variations, and to make our technique more reliable, we can also perform this calibration step at any time during the normal DRAM maintenance operations: e.g. along with ZQ calibration.

In the following section we will explain the experimental platform that we have used for validating our technique

Table 2: Measurement Results

f	Refresh Type	$t^{(*)}_{RAS}$ [ns]	$t^{(*)}_{RRD}$ [ns]	$t^{(*)}_{RP}$ [ns]	$t^{(*)}_{RFC}$ [ns]	$t^{(*)}_{RFC}/t_{REFI}$	I_{DD5} [mA]	Power [W]	Energy [nJ]
400 MHz	*Auto Refresh*	%	%	%	260	3.33%	468	0.702	182.52
	RGR	62.5	10	17.5	390	5.00%	407	0.6105	238.10
	ORGR	27.5	5	12.5	195	2.50%	742	1.113	217.03
533 MHz	*Auto Refresh*	%	%	%	262.5	3.37%	473	0.7095	186.24
	RGR	46.875	7.5	13.125	292.5	3.75%	525.3	0.78795	230.48
	ORGR	20.625	3.75	9.375	146.25	1.88%	956	1.434	209.72

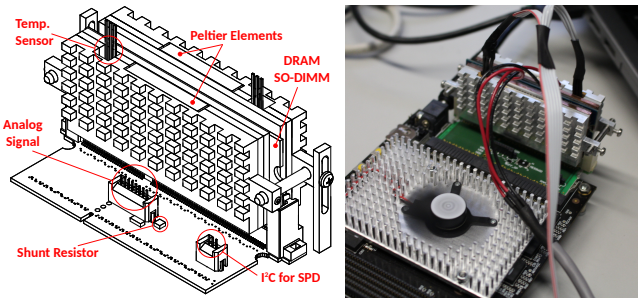


Figure 9: Measurement Platform

5 EXPERIMENTAL SETUP

To validate our technique at higher temperatures, and to conduct accurate current measurements, a precise and reliable measurement platform had to be chosen. FPGA rail power measurements will not give reliable results due to the significant noise involved. Therefore, we developed a custom platform [18] shown in Figure 9 to measure the current consumption and to heat up the DRAM devices of DDR3 SO-DIMM modules. The heating section consists of a mechanical setup, which is placed on the surface of the DRAM devices. For analyzing the current consumption of DRAMs, we designed an adapter PCB for DDR3 SO-DIMMs (6 Layers) that conforms to JEDEC standard requirements. The DDR3 SO-DIMM adapter board is plugged into a Xilinx FPGA based evaluation platform. A digital precision multimeter from Keithley was used for measuring SO-DIMM adapter board current. The state-of-the-art *Memory Interface Generator* (MIG) memory controller from Xilinx [32] is customized to generate the required command and data sequences for the measurements. A *Virtual Input/Output* (VIO) core connected to the Custom MIG enables to control of the internal FPGA signals. For real-time monitoring of the signals we used Vivado Logic Analyzer.

In the following section we present the results that we have measured using our measurement platform.

6 EXPERIMENTAL RESULTS

To verify that ORGR works for state-of-the-art DRAMs, we performed retention tests on a 2 GB DDR3 SO-DIMM (each SO-DIMM containing four 4 Gb, x16 devices at 30 nm technology node from *Vendor-X*). We filled the entire DIMM (all of the DRAMs) with a specific data pattern ($0xFF$) and then we performed ORGR with varying t_{REF} from 1 s to 100 s. Afterward we read the entire DIMM and measured the number of bit flips. Each measurement was taken 10 times. Figure 10a and Figure 10b show the average for the cumulative failure probability for 40°C and 90°C for various retention times. The results shows that our proposed technique using reduced timings outperforms *Auto-Refresh* at lower temperatures and performs well even at 90°C.

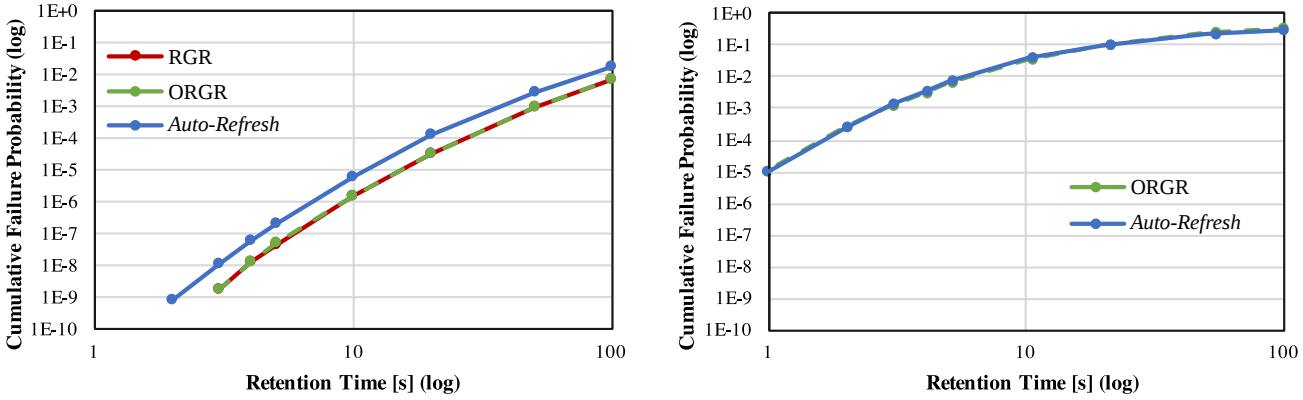
We measured the I_{DD5} for the same SO-DIMM from *Vendor-X* using the platform discussed in Section 5 to obtain the power and energy values. Table 2 shows a comparison of the measured I_{DD5} values for ORGR with RGR and *Auto-Refresh* at 400 MHz and 533 MHz. The DRAM timing values shown in Table 2 for RGR are following the best possible schedule, and we ensured that they never violate the JEDEC-specified minimum values: $t_{RRD} = 6$ ns, $t_{RAS} = 35$ ns and $t_{RP} = 12.5$ ns. For ORGR we used the previously estimated reduced DRAM timings.

The results show that as we reduce the timings as we propose in our technique, the energy efficiency is also increased and approaches that of *Auto-Refresh*. The results also show that the performance of ORGR exceeds that of *Auto-Refresh* by up to 45% and is around 10% more energy efficient compared to RGR. We have also determined t^{min}_{RAS} for different vendors: for *Vendor-X* it is 18.75 ns and for other vendors it varies from 20.325 ns to 24.375 ns.

7 SIMULATION RESULTS

To estimate the benefits of ORGR for future high density DRAM devices, which do not exist yet, we evaluated our technique for an 8 GB DDR4 SO-DIMM consisting of four 16 Gb DDR4 devices. The DRAM specifications shown in Table 3 were generated using the open source DRAM current and timing generator tool DRAMSpec [35, 47]. We executed two different applications: one with a dense memory access pattern, and the other with a sparse memory access pattern. We use the design space exploration framework DRAMSys [21, 22] and DRAMPower [10], to evaluate the performance and energy for *Auto-Refresh*, RGR, and ORGR. The *Average Response Latency* (ARL) is used as the performance evaluation metric for both the applications.

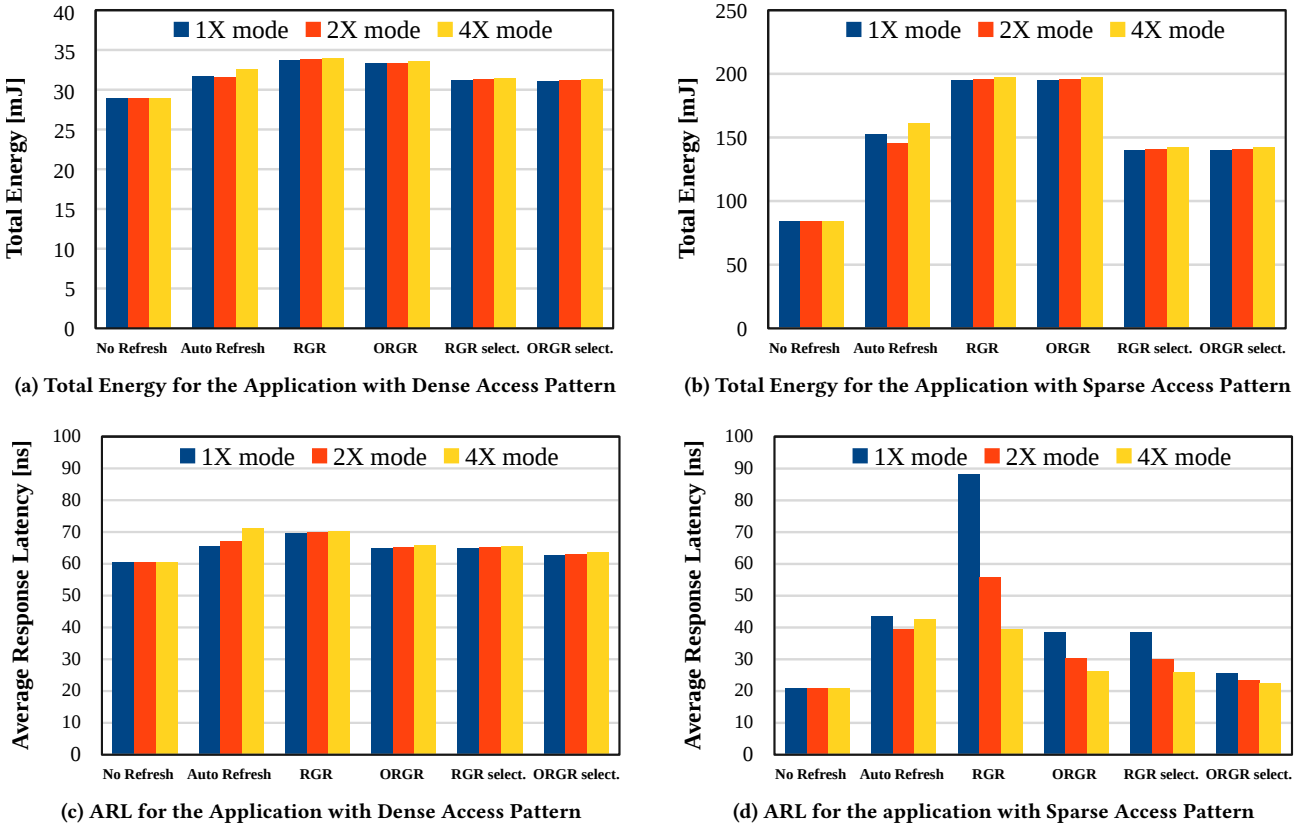
We evaluated six different cases (No Refresh, Auto-Refresh, RGR, ORGR, RGR selective, ORGR selective) each for the three different DDR4 *Fine Granular Refresh* (FGR) modes, shown in Figure 11. For the selective refresh case only half of the DRAM is refreshed, since the applications use only 50 percent of the total memory space. In



(a) **Temperature** = 40°C

(b) **Temperature** = 90°C

Figure 10: Retention Error Behaviour



(a) Total Energy for the Application with Dense Access Pattern

(b) Total Energy for the Application with Sparse Access Pattern

(c) ARL for the Application with Dense Access Pattern

(d) ARL for the application with Sparse Access Pattern

Figure 11: Simulation Results for DDR4

contrast to RGR and ORGR, the *Auto-Refresh* does not provide the flexibility to selectively refresh the DRAM by its nature.

ORGR outperforms *Auto-Refresh* and RGR in terms of ARL for all the cases that we have evaluated. Moreover, for the case where only half the DRAM is refreshed (selective refresh), ORGR is also considerably more energy efficient than *Auto-Refresh* by exploiting flexibility, and it is slightly more energy efficient than RGR.

8 CONCLUSION

In this paper we presented a novel technique to optimize *Row Granular Refresh* using reduced DRAM timings. We have also invented a method to determine the minimum t_{RAS} timing for different vendors by reverse-engineering the DRAM during initialization, and finding out the internal counter value set by vendors. With the use of these reverse engineered timing parameters to optimize the

Table 3: Specifications for 16 Gb DDR4 Device

Parameter	Auto-Ref	RGR	ORGR
t_{RP}	15 ns	15 ns	12.5 ns
t_{RAS}	28.3 ns	28.3 ns	18.3 ns
t_{FAW}	30.8 ns	30.8 ns	6.6 ns
t_{RRD}	6.7 ns	6.7 ns	1.7 ns
t_{REF}	64 ms	64 ms	64 ms
t^{1X}_{REFI}	7.8 μs	7.8 μs	7.8 μs
t^{2X}_{REFI}	3.9 μs	3.9 μs	3.9 μs
t^{4X}_{REFI}	1.95 μs	1.95 μs	1.95 μs
t^{1X}_{RFC}	560.6 ns	1018.2 ns	504.7 ns
t^{2X}_{RFC}	350.6 ns	525.4 ns	258.3 ns
t^{4X}_{RFC}	260.7 ns	279 ns	135.1 ns
I^{1X}_{DD5}	500 mA	%	%
I^{2X}_{DD5}	361.5 mA	%	%
I^{4X}_{DD5}	306.5 mA	%	%
I_{DD0}	52.6 mA	52.6 mA	52.6 mA

Row Granular Refresh we achieve more performance and energy efficiency. We implemented our *Optimized Row Granular Refresh* technique and the required reverse engineering method inside a state-of-the-art memory controller and demonstrated the functionality on an FPGA based evaluation platform. Experimental evaluations show that our optimized technique outperforms *Auto-Refresh* by up to 45% in terms of t_{RFC}, and is approximately 10% more energy efficient compared to the non-optimized *Row Granular Refresh*. Furthermore, we have evaluated the benefits of our technique for future 16 Gb DDR4 SDRAMs using simulations. The simulation results show that our optimized technique performs much better than *Auto-Refresh* and *Row Granular Refresh*, and is more energy efficient than *Auto-Refresh* when the DRAM is only partially refreshed. The proposed technique can be applied to all of the existing refresh-optimization schemes that use row-by-row refresh, especially for future high density DRAMs, and it does so without requiring any modification to the DRAM or DRAM protocol. We assume that these reduced timings are used by the DRAM vendors for performing the *Auto-Refresh* since it is otherwise not possible to complete the *Refresh* operation within the specified t_{RFC}.

ACKNOWLEDGMENTS

The authors thank Chirag Sudarshan, Vitor Kieling, Frederik Lauer and Martin Schultheis, for their 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) the DFG grant no. WE2442/10-1 (http://www.uni-kl.de/3d-dram) and the Carl-Zeiss Stiftung. The project OPRECOMP acknowledges the financial support of the Future and Emerging Technologies (FET) programme within the European Unions Horizon 2020 research and innovation programme, under grant agreement No.732631 (http://www.oprecomp.eu). Furthermore, the paper was supported by the *Fraunhofer High Performance Center* for *Simulation-and Software-based Innovation.*

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] A. Agrawal, M. O'Connor, E. Bolotin, N. Chatterjee, J. Emer, and S. Keckler. CLARA: Circular Linked-List Auto and Self Refresh Architecture. In *Proceedings of the Second International Symposium on Memory Systems*, MEMSYS '16, pages 338–349, New York, NY, USA, 2016. ACM.

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

[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, Penalties, and Trade-Offs. *IEEE Transactions on Computers*, 65(1):108–121, Jan 2016.

[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] K. Chandrasekar, S. Goossens, C. Weis, M. Koedam, B. Akesson, N. Wehn, and K. Goossens. Exploiting Expendable Process-margins in DRAMs for Run-time Performance Optimization. In *Proceedings of the Conference on Design, Automation & Test in Europe*, DATE '14, pages 173:1–173:6, 3001 Leuven, Belgium, Belgium, 2014. European Design and Automation Association.

[9] K. Chandrasekar, C. Weis, B. Akesson, N. Wehn, and K. Goossens. Towards Variation-Aware System-Level Power Estimation of DRAMs: An Empirical Approach. In *Proc. 50th Design Automation Conference*, Austin, USA, June 2013.

[10] 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. http://www.drampower.info.

[11] K. K. Chang, A. Kashyap, H. Hassan, S. Ghose, K. Hsieh, D. Lee, T. Li, G. Pekhimenko, S. Khan, and O. Mutlu. Understanding Latency Variation in Modern DRAM Chips: Experimental Characterization, Analysis, and Optimization. In *Proceedings of the 2016 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Science*, SIGMETRICS '16, pages 323–336, New York, NY, USA, 2016. ACM.

[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] 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.

[15] Y.-H. Gong and S. W. Chung. Exploiting Refresh Effect of DRAM Read Operations: A Practical Approach to Low-Power Refresh. *IEEE Trans. Comput.*, 65(5):1507–1517, May 2016.

[16] 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.

[17] B. Jacob, S. Ng, and D. Wang. *Memory Systems: Cache, DRAM, Disk*. Elsevier Science, 2010.

[18] M. Jung, D. Mathew, C. Rheinländer, C. Weis, and N. Wehn. A Platform to Analyze DDR3 DRAM's Power and Retention Time. *IEEE Design & Test*, 2017.

[19] M. Jung, D. M. Mathew, C. Weis, and N. Wehn. Approximate Computing with Partially Unreliable Dynamic Random Access Memory: Approximate DRAM. In *IEEE/ACM Design Automation Conference (DAC)*, June 2016.

[20] M. Jung, D. M. Mathew, C. Weis, and N. Wehn. Efficient Reliability Management in SoCs - An Approximate DRAM Perspective. In *21st Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2016.

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

[22] 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.
[23] M. Jung, E. Zulian, D. Mathew, M. Herrmann, C. Brugger, C. Weis, and N. Wehn. Omitting Refresh - A Case Study for Commodity and Wide I/O DRAMs. In *1st International Symposium on Memory Systems (MEMSYS 2015)*, Washington, DC, USA, October 2015.
[24] D. Lee, Y. Kim, G. Pekhimenko, S. Khan, V. Seshadri, K. Chang, and O. Mutlu. Adaptive-latency DRAM: Optimizing DRAM timing for the common-case. In *High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on*, pages 489–501, Feb 2015.
[25] 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.
[26] 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.
[27] 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.
[28] 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.
[29] J. Lucas, M. Alvarez-Mesa, M. Andersch, and B. Juurlink. Sparkk: Quality-Scalable Approximate Storage in DRAM. In *The Memory Forum*, June 2014.
[30] Jedec Solid State Technology Association. DDR3 SDRAM (JESD 79-3), 2012.
[31] Micron Technology Inc. 1Gb: x4, x8, x16 DDR3 SDRAM. July 2006.
[32] Xilinx, Inc. Memory Interface Generator (MIG). http://www.xilinx.com/products/intellectual-property/mig.html, 2015, Last Access: 18.02.2015.
[33] 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.
[34] 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.
[35] O. Naji, C. Weis, M. Jung, N. Wehn, and A. Hansson. A High-Level DRAM Timing, Power and Area Exploration Tool. In *Embedded Computer Systems Architectures Modeling and Simulation (SAMOS)*, July 2015.
[36] 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.
[37] 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.
[38] A. Raha, H. Jayakumar, S. Sutar, and V. Raghunathan. Quality-aware Data Allocation in Approximate DRAM. In *Proceedings of the 2015 International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, CASES '15, pages 89–98, Piscataway, NJ, USA, 2015. IEEE Press.
[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] W. Shin, J. Choi, J. Jang, J. Suh, Y. Moon, Y. Kwon, and L. S. Kim. DRAM-Latency Optimization Inspired by Relationship between Row-Access Time and Refresh Timing. *IEEE Transactions on Computers*, 65(10):3027–3040, Oct 2016.
[41] W. Shin, J. Yang, J. Choi, and L.-S. Kim. NUAT: A non-uniform access time memory controller. In *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*, pages 464–475. IEEE, 2014.
[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 IEEE Conference on Design, Automation & Test in Europe (DATE)*. European Design and Automation Association, 2015.
[47] C. Weis, A. Mutaal, O. Naji, M. Jung, A. Hansson, and N. Wehn. DRAMSpec: A High-Level DRAM Timing, Power and Area Exploration Tool. *International Journal of Parallel Programming*, pages 1–26, 2016.
[48] 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.
[49] X. Zhang, Y. Zhang, B. Childers, and J. Yang. AWARD: Approximation-aWAre Restore in Further Scaling DRAM. In *Proceedings of the Second International Symposium on Memory Systems*, MEMSYS '16, pages 322–324, New York, NY, USA, 2016. ACM.
[50] X. Zhang, Y. Zhang, B. R. Childers, and J. Yang. Exploiting DRAM restore time variations in deep sub-micron scaling. In *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 477–482, March 2015.
[51] X. Zhang, Y. Zhang, B. R. Childers, and J. Yang. Restore truncation for performance improvement in future DRAM systems. In *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 543–554, March 2016.
[52] 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.