Real-Time Systems
https://doi.org/10.1007/s11241-019-09338-8

Reduced latency DRAM for multi-core safety-critical real-time systems

Mohamed Hassan[1]

Abstract
Predictable execution time upon accessing shared memories in multi-core real-time systems is a stringent requirement. A plethora of existing works focus on the analysis of Double Data Rate Dynamic Random Access Memories (DDR DRAMs), or redesigning its memory to provide predictable memory behavior. In this paper, we show that DDR DRAMs by construction suffer inherent limitations associated with achieving such predictability. These limitations lead to (1) highly variable access latencies that fluctuate based on various factors such as access patterns and memory state from previous accesses, and (2) overly pessimistic latency bounds. As a result, DDR DRAMs can be ill-suited for some real-time systems that mandate a strict predictable performance with tight timing constraints. Targeting these systems, we promote an alternative off-chip memory solution that is based on the emerging Reduced Latency DRAM (RLDRAM) protocol, and propose a predictable memory controller (RLDC) managing accesses to this memory. Comparing with the state-of-the-art predictable DDR controllers, the proposed solution provides up to **11**× less timing variability and **6.4**× reduction in the worst case memory latency.

Keywords DRAM · RLDRAM · Latency · Memory · Real-time systems · Safety-critical systems

1 Introduction

With the Internet-of-Things (IoT) revolution, real-time systems are unprecedentedly becoming ubiquitous in our daily life. Examples include healthcare devices, automotive, and smart power grids. In these systems, a failure can result in severe consequences such as loss of lives. This failure is possible not only by incorrect functionality, but

✉ Mohamed Hassan
mohamed.hassan@mcmaster.ca

[1] Electrical and Computer Engineering Dept., McMaster University, ITB A318, 1280 Main Street West, Hamilton, ON L8S 4K1, Canada

Published online: 10 September 2019

also by violating temporal requirements. Accordingly, a detailed worst-case execution time (WCET) analysis (statically or experimentally) of a real-time task's execution is necessary to ensure satisfying the task's temporal requirements. Hardware components have to follow a predictable behavior to allow for this analysis. Unfortunately, conventional computing systems are not designed to be predictable. Numerous architectural optimizations such as deep pipelines, branch prediction, and aggressive reordering aim to provide high performance at the expense of immense timing variability. Since failures in real-time systems have to be avoided at all costs, hardware architecture has to be reconsidered to account for predictability in the first place.

Considering off-chip main memory, which is a critical component of most computing systems (Mutlu and Subramanian 2014), we find that Double Data Rate Dynamic Random Access Memories (DDR DRAMs) or simply (DDRx[1]) are the most used nowadays. This is because they provide a low-cost, a high-capacity, and a high-bandwidth solution for performance oriented systems. Despite the name, DDRx DRAMs in reality do not provide random access. Their access latency varies notably based on many factors such as access patterns, transaction type (read or write), and the DRAM state from previous accesses. Moreover, to further boost DDRx performance and reduce access latency, memory controllers usually employ complex optimizations such as multiple reordering levels, prioritizations, and adaptive policies (Hassan and Pate 2017). These memory controller optimizations along with the variability of DDRx's access latency result in highly pessimistic worst case latency (WCL) (Kim et al. 2014; Yun et al. 2015), which encumbers the deployment of DDRx in real-time systems. To address this challenge, researchers proposed redesigning the memory controller to provide predictable access to DDRx memories (Akesson et al. 2007; Ecco and Ernst 2015; Ecco et al. 2016, 2014; Goossens et al. 2013; Hassan et al. 2015, 2016; Jalle et al. 2014; Kim et al. 2015; Krishnapillai et al. 2014; Li et al. 2014; Paolieri et al. 2009; Reineke et al. 2011; Valsan and Yun 2015; Wu et al. 2013). This approach helps in reducing the interference latency amongst requests belonging to different tasks or processing elements (PEs). Nonetheless, it does not reduce the variability in the access latency of the DDRx itself. This access latency is inherently bounded by the physical characteristic of the DDRx chips and is enforced by the JEDEC standard constraints (2018). DDRx internal circuitry by construction targets average-case performance with complex interactions between DDRx commands and more than 20 timing constraints (Wu et al. 2013). As we show in this paper, this inherent variability greatly affects the resulting WCL even when adopting one of the aforementioned predictable memory controllers in the system. As a consequence, we believe revolutionary solutions have to be devised to provide more predictable memory behavior with lower WCL for real-time systems.

Towards this target, we make the following contributions.

1. We thoroughly study the variability of DDRx's access latency and expose its limitations with regard to real-time systems (Sect. 3).
2. Motivated by these limitations, we explore alternatives to DDRx memories. Namely, we promote the adoption of Reduced Latency DRAM (RLDRAM) in

[1] we use the letter x since observations we make in this paper are generic for any DDR protocol (DDR2, DDR3, DDR4, etc.).

real-time systems. RLDRAM is an emerging DRAM protocol that is currently led by Micron Technology Inc. (2018a) and provides predictable behavior with lower access latency compared to DDRx protocols. We illustrate how RLDRAM can provide a considerable reduction in WCL as well as less access variability, which establishes the motivation towards adopting RLDRAM in real-time systems (Sect. 4).
3. To enable this adoption, we propose RLDC: a memory controller design that predictably manages accesses to the RLDRAM. We also conduct timing analysis that provides an upper bound on the latency suffered by any memory request upon accessing the RLDRAM using RLDC (Sect. 5).
4. To show the effectiveness of the proposed solution, we compare with eight of the state-of-the-art predictable DDRx controllers using representative benchmarks from the automotive domain. Results show that the proposed solution provides up to 6.4× reduction in WCL and 11× less latency variability (Sect. 6).

1.1 Extended version

A preliminary version of this paper was proposed in Hassan (2018). Compared to Hassan (2018), this paper makes the following contributions.

1. We highlight the fact that RLDRAM can be configured to use two addressing modes. In addition to the non-multiplexed addressing mode used in Hassan (2018), we investigate in this paper the usage of a multiplexed address mode. We thoroughly study the effect of both modes on the latency variability as well as access latency of RLDRAM. We also compare the advantages and disadvantages of each mode when used in real-time systems.
2. We extend the proposed RLDC controller to enable the usage of both addressing modes. In addition, we extend the timing analysis (Sect. 5.3) to derive latency bounds under both addressing modes. We then compare both addressing modes with regard to the resulting total WCL guarantees provided by RLDC upon accessing RLDRAM (Sect. 6.5).
3. We study the effect of using various possible burst lengths. RLDRAM supports a burst length of 2, 4 and 8. The evaluation in Hassan (2018) focused on a burst length of 8. In this paper, we study the effect of different burst lengths on the access latency of the RLDRAM device (Sect. 4.2.2) as well as on the WCL behavior of the RLDC controller (Sect. 6.5).
4. The study of the access latency variability of DDRx DRAMs in Hassan (2018) focused on scenarios incurred by read transactions. For sake of completeness, we extend this study to include also scenarios incurred by write transactions (Sect. 3.2). This extensions shows that write transactions incur even higher variability in access latency to DDRx DRAMs than the read transactions.
5. In the study of state-of-the-art predictable DDRx controllers (Sect. 3.3), we detail the analysis of computing the latency bounds as well as latency variability.

2 Related work

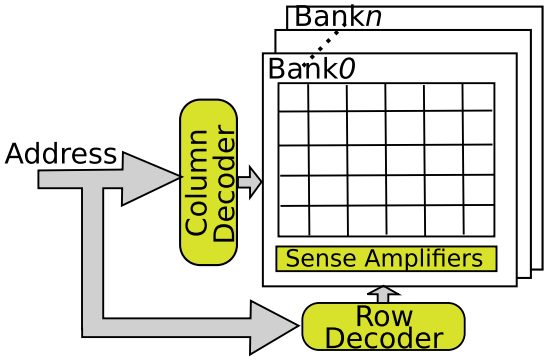
Predictable DDRx solutions Existing works focus on providing predictability to real-time tasks upon accessing DDRx main memories (e.g. Akesson et al. 2007; Ecco and Ernst 2015; Ecco et al. 2016, 2014; Goossens et al. 2013; Hassan et al. 2015, 2016; Hassan and Pellizzoni 2018; Jalle et al. 2014; Kim et al. 2015, 2014; Krishnapillai et al. 2014; Li et al. 2014; Paolieri et al. 2009; Reineke et al. 2011; Valsan and Yun 2015; Wu et al. 2013; Yun et al. 2014, 2015). These efforts follow two major directions. The first direction is to analyze existing memory controllers used in conventional high-performance systems to upper bound the latency suffered by any request upon accessing DDRx main memory (Hassan and Pellizzoni 2018; Kim et al. 2014; Yun et al. 2015). Following a similar direction, (Yun et al. 2014) targets to bound DRAM interference in conventional platforms by enforcing bank partitioning at the operating system level. As aforementioned, these commodity controllers target to increase average-case performance through complex optimizations at the expense of predictability. Consequently, the provided bounds by these approaches are pessimistic, which entails them ill-suited for real-time systems with tight timing requirements. The second direction is to entirely or partially redesign the memory controller to account for predictability (e.g. Akesson et al. 2007; Ecco and Ernst 2015, 2017; Ecco et al. 2016, 2014; Goossens et al. 2013; Guo and Pellizzoni 2017; Hassan et al. 2015, 2016; Jalle et al. 2014; Kim et al. 2015; Krishnapillai et al. 2014; Li et al. 2014; Paolieri et al. 2009; Reineke et al. 2011; Valsan and Yun 2015; Wu et al. 2013). A comparative study that highlights strengths and limitations of some of these controllers is proposed in Guo et al. (2018). Although this approach reduces latency variabilities due to delay interferences among requests of different tasks, it suffers from the two main drawbacks. (1) It still suffers from high WCLs due to the complex interactions between DDRx commands. (2) It can not address the variability in the access latency of the DDRx chips. We show that access latency variability in DDRx memories severely affects the predictability of memory requests. We address this problem by promoting the deployment of RLDRAM emerging memories in real-time systems. We provide a predictable RLDRAM memory controller with tighter latency bounds and less variability compared to these DDRx controllers.

RLDRAM RLDRAM memory is originally targeted at high-speed routers (Micron Technology Inc. 2018b) and network processing (Toal et al. 2007). Researchers also envisioned the usage of RLDRAM as a low-latency memory module in a heterogeneous memory system to increase overall memory performance (Chatterjee et al. 2012; Phadke and Narayanasamy 2011). To our best knowledge, we are the first to investigate the usage of RLDRAM in real-time systems.

3 DDRx limitations

We first introduce the basics of the DDRx protocol. Then, we study its limitations with regard to providing predictability.

Fig. 1 DRAM architecture



3.1 DDRx DRAM basics

Figure 1 illustrates the basic structure of DDRx DRAMs. They consist of an array of memory cells arranged as *banks*. Cells in each bank are organized in *rows* and *columns*. Multiple banks can construct a logical entity called a *rank*. Each bank has sense amplifiers, which also cache the most-recently accessed row in each bank (known as *row buffer*). As Fig. 1 shows, the address bits are split into two segments. One segment is used by the row decoder to determine the requested row, and the other is used by the column decoder to determine which column to access within this row. DDRx memories use a multiplexed address mode such that these two segments are provided to the memory in two steps. First, the row address is provided to activate the requested row. Then, in a later cycle, the remaining address is provided to index specific column(s) in the activated row.

Although address multiplexing reduces the pin count of the DDRx chip (and hence reduces its cost), it increases the access latency as one memory access is now split into multiple stages. Namely, one access to a DDRx memory can comprise a maximum of three stages: (1) precharge, (2) activate, and (3) read or write. All these stages have to be orchestrated by an on-chip memory controller through issuing memory commands. A precharge command (P) writes back the data in the row buffer into DRAM cells. It is needed if the access is to a row different than the one in the row buffer. An activate command (A) fetches the requested row from the cells into the row buffer. Read/Write commands (R/W) conduct the requested memory operation. The controller also needs to periodically issue a REF command to negate the charge leakage from the capacitors that store the data. The effect of REF on predictability is deterministic and is limited to around 2% of task's memory latency (Kim et al. 2014). In addition, the majority of the DDRx controllers do not incorporate this effect on their analysis as they conduct a per-request analysis, while the REF effect should be incorporated in WCET analysis at the task level to avoid pessimism (Guo et al. 2018; Hassan et al. 2015). For these reasons, we do not consider the REF command in this paper.

All commands have strict timing constraints that are dictated by the JEDEC standard (2018) and must be satisfied by all memory controller designs. Table 1 tabulates the most relevant timing constraints for DDR3-1600 DRAM. It is worth noting that, in addition to increasing the access latency, the aforementioned three stages (precharge,

Table 1 JEDEC timing constraints (JEDEC DDR3 SDRAM 2018)

Parameter	Delay description	Cycles
$tRCD$	A to R/W	10
$tCCD$	R to R or W to W (same rank)	4
tRL	R to start of data transfer	10
tRP	P to A	10
tWL	W to start of data transfer	9
$tRTW$	R to W	6
$tRTP$	R to P	5
$tWTR$	End of data transfer of W to R	5
tWR	End of data transfer of W to P	10
$tRAS$	A to P	24
tRC	A to A (same bank)	34
$tRRD$	A to A (diff bank in same rank)	4
$BL/2$	Data bus transfer	4 ($BL = 8$)[a]
$tRTRS$	Rank to rank switch	1

[a]BL is the burst length, which indicates the number of data beats to be transferred by one access

activate, and read/write) lead to high variability in access latency. This is because one request can consist of one, two, or three stages based on the memory state as follows. (1) If the request targets a row that already exists in the row buffer (denoted as *open row*), it only consists of a single stage and the controller issues only either a R or W command based on the request type. (2) If the request targets a bank that is already precharged (i.e. does not have an open row in the row buffer), it consists of two stages: the activate stage to bring the row to the row buffer in addition to the read/write stage. We say in this case that the request targets a *closed row*. (3) Finally, if the request targets a bank that has an open row in the row buffer that is different than the targeted row by the request, it needs all the three stages and the controller in this case issues three commands on behalf of that request: P, A, and then R or W. We say in this case that the request is targeting a *conflict row*.

Finally, the data bus of the DDRx is bidirectional: same wires are used to both read from and write to the DRAM. This also increases access latency since the data bus needs to switch from read to write or vice versa. For instance, in DDR3-1600 devices, the R-to-W switching delay is 6 cycles, while the W-to-R delay is 18 cycles.

3.2 Predictability considerations

Since predictability is of utmost importance in real-time systems, we investigate in details the effect of DDRx's access latency variability on predictability. Predictability has different definitions in the real-time literature. One important measure of system predictability is the relative difference between best- and worst-case execution times (or latencies in case of memories) (Wilhelm et al. 2008). The latency of a memory request can be anywhere between the best-case latency (BCL) and WCL. To differ-

entiate between the effects from the access protocol of the DRAM and the effects from the scheduling techniques of the controller, we define two latency components: memory access latency (Definition 1), and total memory latency (Definition 2).

Definition 1 *Memory access latency* of a request to the DRAM is measured from the time stamp when the request is elected by the scheduling mechanisms of the controller to be sent to the memory (i.e. it is at the head of the scheduling queue) until its data starts the transfer on the data bus.

Definition 2 *Total memory latency* of a request to the DRAM is measured from its arrival at the memory controller until the start of its data transfer.

Memory access latency accounts only for time consumed by the request itself to perform the access including delays suffered due to the current state of the memory (i.e. remaining timing constraints from previously issued requests). In contrast, total memory latency accounts for delays due to both arbitration decisions at the memory controller as well as the access latency to the DRAM. Since the target of this subsection is to study the predictability of the DRAM device itself and not the controller, we eliminate any scheduling-specific effects induced by the controller. Therefore, we focus only on the access latency as per Definition 1. The controller's scheduling effects on predictability is discussed in Sect. 3.3. To obtain best- and worst-case DRAM access latencies, we study all possible access scenarios and their corresponding access latency. Then, to quantitatively measure the DRAM predictability, we define the term variability window in Definition 3.

Definition 3 *Variability window* of a latency component, VW, is a measure of the possible variations in the value of this component and is computed as the percentage increase from the best (BCL) to the worst case latency (WCL) values of this component.

$$\mathrm{VW} = \frac{WCL - BCL}{BCL} \times 100 \tag{1}$$

In DDRx memories, many factors determine the access latency of the request such as (1) the DRAM state from previous accesses, (2) the data bus direction based on the type of the current and previous transactions (read or write), and (3) the request address (i.e. targeted rank, bank, and row). Figures 2 and 3 illustrate the variability in the access latency by dictating example access scenarios that exhibit the effects from one or more of these factors. Both figures consider a sequence of two consecutive requests accessing DDRx. Using a sequence of two requests instead of just one request is necessary to show the case for delays due to the change of the DRAM state caused by commands of a previous request. The second request with subscript 1 in the figure is the request under consideration, while the first request (with subscript 0) is the previous request. Figure 2 delineates command interactions for the considered request being a read, while the scenarios for a write transaction are delineated in Fig. 3. The chart in the upper right of Figs. 2 and 3 delineates the access latency of the considered request for each scenario, while the table briefly explains the scenarios. As per Definition 1, access latency is measured from the arrival of the considered request at the head of

the queue at cycle 0, until the start of its data transfer (DATA_1). Complying with the majority of the commodity as well as predictable DRAM controllers, we assume that once the DRAM started executing one command of a request, it cannot be preempted, and commands from different requests can be pipelined to increase performance. The specific clock cycle values in Figs. 2 and 3 reflect the timing constraints of DDR3-1600 as tabulated in Table 1; however, the general scenarios apply for all DDRx devices since the basics of the access protocol remain the same. Since both a read request (Fig. 2) and a write request (Fig. 3) encounter similar access scenarios and the only differences are in timing constraints. In the following subsections, we focus on detailing the access scenarios for only the read request in Fig. 2. Afterwards, we compute BCL, WCL, and VW for both the read and the write transactions.

3.2.1 Targeting an open row

Figure 2a–e represent scenarios in which the considered request targets a row that is already open in the row buffer. Accordingly, the request does not need the precharge nor the activate stages, and only issues a R command. Figure 2a depicts the best case scenario: in addition to targeting an open row, R_1 satisfies all timing constraints upon arrival; hence, the controller issues R_1 immediately. DRAM takes tRL cycles to place the data into the data bus. In Fig. 2b, R_1 arrives directly after the memory has serviced another read, R_0, to a different rank; accordingly, R_1 has to be delayed by $BL/2 + tRTRS$ cycles. This is because data transfers from different ranks according to the standard has to be separated by the rank switching latency of $tRTRS$ cycles. Similarly, in Fig. 2c, arrives directly after the memory has serviced another write, W_0. Therefore, R_1 has to be delayed by $tWL + BL/2 + tRTRS - tRL$ cycles.

In Fig. 2d, R_1 arrives directly after the memory has serviced R_0 to the same rank (either same bank or not). According to the JEDEC standard, it has to be delayed by $tCCD$ cycles before conducting the access. R_1 in Fig. 2e is further delayed by the data bus turnaround time. Since the DRAM bus is bidirectional, certain delay has to elapse between every two successive requests of different types to same rank. R_1 arrives after a write request, W_0, to same rank; thus, it has to be delayed by a W-to-R delay of $tWL + BL/2 + tWTR$ cycles.

3.2.2 Targeting a closed row

In Fig. 2f–j, the considered request targets a closed row. Accordingly, it consists of A_1 and R_1 commands. In Fig. 2f, the request arrives such that all timing constraints invoked due to previous requests are already satisfied. Therefore, the controller immediately issues A_1 at cycle 0, waits for the row to be activated ($tRCD$ cycles), and then issue the R_1 command at cycle 10. In Fig. 2g, h, the request arrives while the memory is servicing a request to a different rank. A_1 is still issued immediately since the standard does not impose constraints among A commands to different ranks. However, R_1 has to wait for additional delays due to this previous request. In Fig. 2g, it has to wait for $BL/2 + tRTRS$ cycles after R_0 similar to Fig. 2d, while in Fig. 2h, it has to wait for $tWL + BL/2 + tRTRS - tRL$ cycles after W_0 similar to Fig. 2e. In Fig. 2i, j, the request arrives while the memory is servicing a previous request to a different bank in

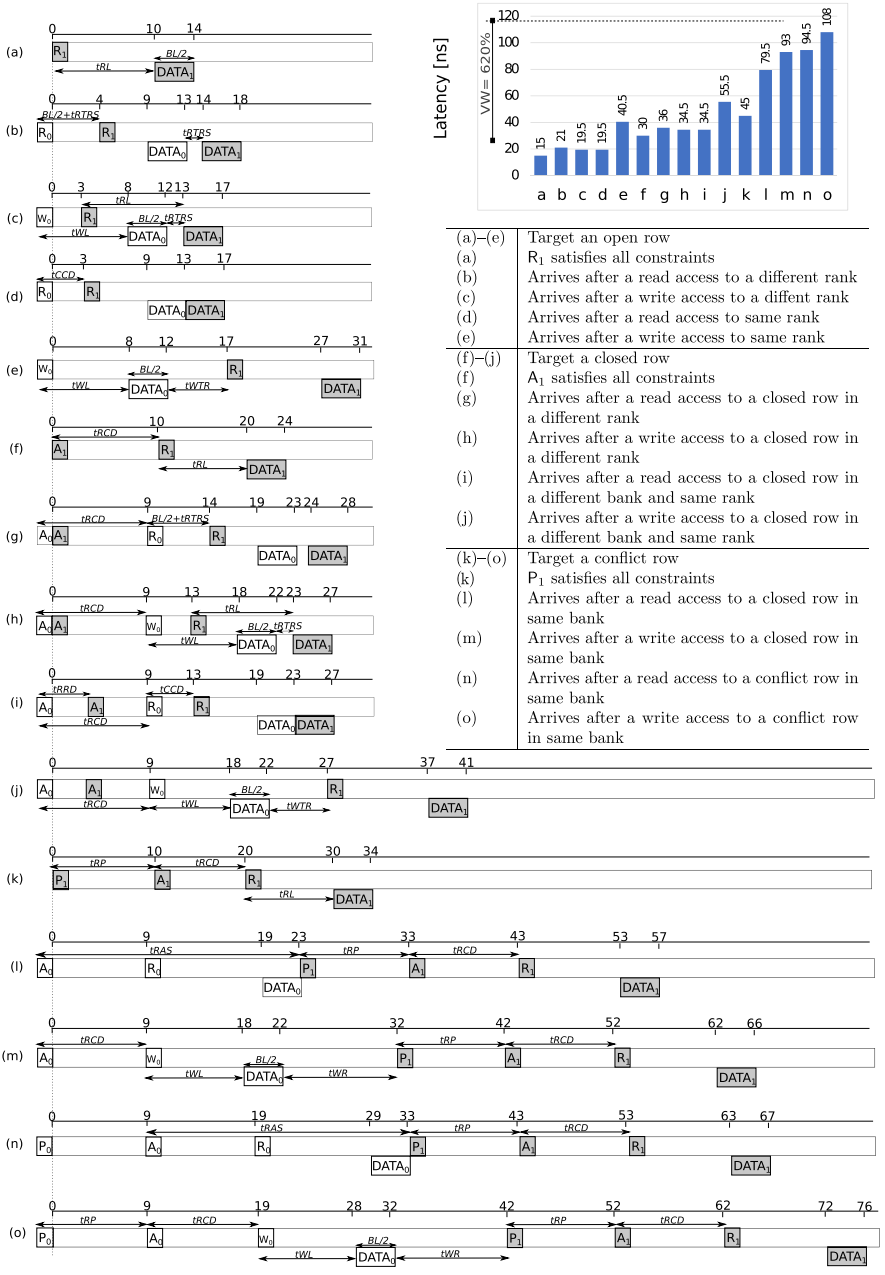


Fig. 2 Different DDRx access scenarios based on the arrival time of the request and the state of the memory from the directly previous request. The considered request is a read and arrives at the head of the queue at time 0. Timing constraints are relatively scaled based on JEDEC standard for DDR3-1600 (2018) (Table 1), with a clock of 1.5 ns. Subscripts are for request numbers. Latency is measured from arrival to the start of data transfer [latency for (a) is 10 cycles or $10 \times 1.5 = 15$ ns]

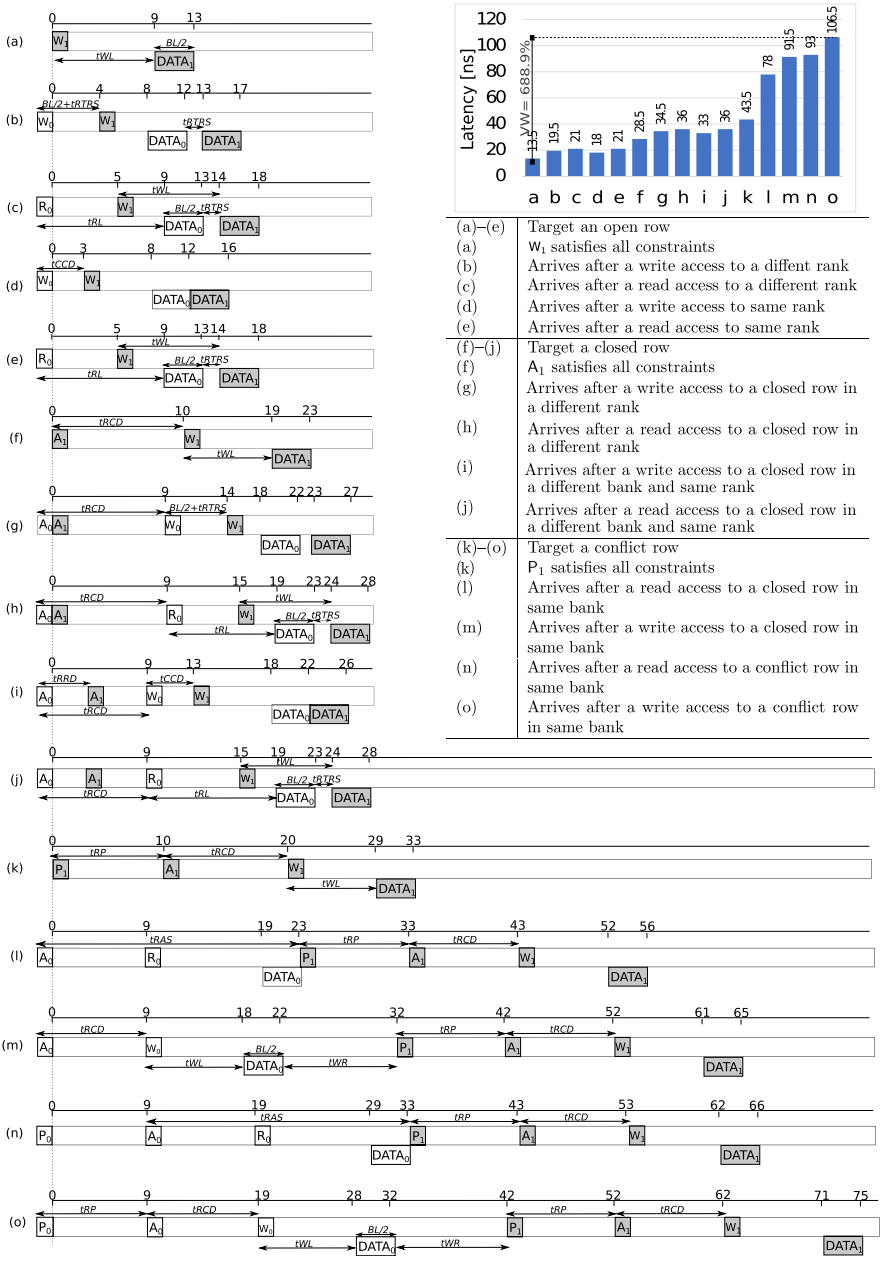


Fig. 3 Different DDRx access scenarios based on the arrival time of the request and the state of the memory from the directly previous request. The considered request is a write and arrives at the head of the queue at time 0. Timing constraints are relatively scaled based on JEDEC standard for DDR3-1600 (2018) (Table 1), with a clock of 1.5 ns. Subscripts are for request numbers. Latency is measured from arrival to the start of data transfer [latency for (a) is 9 cycles or $9 \times 1.5 = 13.5$ ns]

same rank. In this case, A_1 and A_0 has to be separated by $tRRD$ cycles in addition to the aforementioned timing constraints related to R_1.

3.2.3 Targeting a conflict row

Figure 2k–o are for a request that targets a conflict row, and thus, consists of P_1, A_1, and R_1. In Fig. 2k, the request satisfies all timing constraints upon arrival. The controller immediately issues P_1 at cycle 0 to precharge the existing row in the row buffer, an operation that consumes tRP cycles. Afterwards, it issues the A_1 command to activate the row, and then issues R_1 after additional $tRCD$ cycles. In Fig. 2l, m, the request arrives one cycle after the memory controller issued A_0 command of a previous request to a different row in the same bank. This previous request is a read in Fig. 2l. Consequently, P_1 is delayed by $tRAS - 1$ cycles. On the other hand, in Fig. 2m, the previous request is a write. Hence, P_1 cannot be issued before tWR cycles after writing the data of that previous request. Fig. 2n and o are similar to Fig. 2l and m, respectively. However, the considered request arrives directly after a P_0 command instead of a A_0. Therefore, P_1 is further delayed by extra tRP cycles.

3.2.4 Variability window for the read request

While Fig. 2a represents the best-case scenario for the two-request sequence with the access latency of the considered read request being 10 cycles, Fig. 2o represents the worst-case scenario with access latency of 72 cycles. As the chart in Fig. 2 highlights, the resulting variability window is 620%.

3.2.5 Variability window for the write request

Similar to the read request case, the scenario in Fig. 3a represents the best case where the write request suffers an access latency of $tWL = 9$ cycles, while the scenario in Fig. 3o represents the worst case with access latency of 71 cycles. Therefore, the resulting variability window for the write request is 688.9%, which is even higher than the variability window of the read request case.

3.2.6 A concluding remark

From this detailed discussion, it is obvious that the variability window for DDR DRAMs is significantly high, which makes achieving predictability in DRAM a challenging task. Moreover, this value does not include the variability of the memory controller behavior. When considered, the variability window is expected to further increase since the memory controller scheduler accounts for large number of arbitration decisions. Examples include prioritizing requests from certain processing elements over the others, arbitrating based on transaction type (read vs write), and scheduling based on request addresses such as per-rank and per-bank arbitration. As we show in the next subsection, even state-of-the-art DDRx controllers aimed at real-time systems suffer significant variability in memory latency.

3.3 Variability window in predictable memory controllers

We study both analytically and empirically the memory behavior of the state-of-the-art predictable memory controllers: AMC Paolieri et al. (2009), PMC Hassan et al. (2015), RTMem Li et al. (2014), DCmc Jalle et al. (2014), ORP Wu et al. (2013), MCMC Ecco et al. (2014), ROC Krishnapillai et al. (2014), and ReOrder Ecco and Ernst (2015), Ecco et al. (2016). In this section, we discuss the analytical results, while we discuss the empirical results in the evaluation section (Sect. 6). From Definition 3, to compute VW for each of the eight DDRx controllers, we need to compute both WCL and BCL.

3.3.1 Worst-case latency

For the WCL, we use the generalized model provided in Guo et al. (2018). In particular, we use Eqs. 2 and 3 from Guo et al. (2018) for open- and close-page controllers (replicated by Eqs. 2 and 3 in this report), respectively. HR is the row hit ratio of the task. Hit ratio is the percentage of requests accessing a row that is already existing in the row buffer. $REQr$ is either the number of PEs sharing the same rank as the PE under analysis (for controllers with rank support), or the total number of PEs in the system (for controllers without rank support). The terms *BasicAccess*, *RowAccess*, *Interference*, and *RowInter* are controller dependent and are defined in Table 2 (a replication of Table 3 in Guo et al. (2018)). In Table 2, BI determines the number of banks accessed by a request, BC determines the number of read (R) or write (W) commands generated for each bank, while R represents the number of ranks.

$$Latency^{Req} = BasicAccess + Interference \cdot (REQr - 1) \tag{2}$$

$$Latency^{Req} = (BasicAccess + RowAccess \cdot (1 - HR)) + (Interference + RowInter \cdot (1 - HR)) \cdot (REQr - 1) \tag{3}$$

In this section, we derive the VW for a system with four PEs. For controllers aimed at multi-rank DRAMs (MCMC, ROC, and ReOrder), we calculate the VW for a 4-rank memory system ($R = 4$ in Table 2). Accordingly, in Eqs. 2 and 3, as well as in Table 2: $REQr = 4$ for controllers that do not support multi ranks, while $REQr = 1$ for controllers supporting multi ranks. It is worth noting that PMC has a new version that supports multi-ranks (Hassan et al. 2016); however, this analysis only covers the version with no multi-rank support in Hassan et al. (2015). For controllers that require knowledge about the hit ratio (DCmc, ORP, ROC, and ReOrder), we assume a hit ratio of $HR = 35\%$. Computing the WCL for different number of ranks or different hit ratios is not the focus of this paper and is already studied in the the corresponding papers of these controllers as well as in the comparative study in Guo et al. (2018). Finally, we derive the analysis for a single memory access (i.e. $BI = 1$ and $BC = 1$). Based on all these substitutions as well as the timing constraints for DDR3 shown in Table 1, we delineate the analytical WCL in Fig. 4.

Table 2 MC general equation components ($\mathcal{K}(cond)$ equals 1 if *cond* is satisfied and 0 otherwise)

	RowInter	*Interference*	*BasicAccess*	*RowAccess*
AMC	NA	$\big(15 \cdot \mathcal{K}(BI = 8) + 42\big) \cdot BC$	$\big(15 \cdot \mathcal{K}(BI = 8) + 42\big)$	NA
PMC RTMem	NA	$\mathcal{K}(BC = 1) \cdot \big((15 \cdot \mathcal{K}(BI = 8) + 42)\big) + \mathcal{K}(BC > 1) \cdot \big((4 \cdot BC + 1) \cdot BI + 13 + 4 \cdot \mathcal{K}(BI = 8)\big)$	$\mathcal{K}(BC = 1) \cdot \big((15 \cdot \mathcal{K}(BI = 8) + 42)\big) + \mathcal{K}(BC \neq 1) \cdot \big((4 \cdot BC + 1) \cdot BI + 13 + 4 \cdot \mathcal{K}(BI = 8)\big)$	NA
DCmc	0	$28 \cdot BC$	$13 \cdot BC$	18
ORP	7	$13 \cdot BC$	$19 \cdot BC + 6$	27
ReOrder	$7 + 3R$	$8R \cdot BC$	$(8R + 25) \cdot BC$	$33 + 3R$
ROC	$3 \cdot R + 6$	$\big(3 \cdot R + 12\big) \cdot BC$	$\big(3 \cdot R + 24\big) \cdot BC + 6$	$3 \cdot R + 27$
MCMC	NA	$Slot \cdot R \cdot BC$	$Slot \cdot R \cdot BC + 22$	
		Where $Slot = \begin{cases} 42/PE & if(REQr \leq 6) \wedge (R \leq 2) \\ 9 & if(R = 2) \wedge (REQr > 6) \\ 7 & Otherwise \end{cases}$		NA
FR-FCFS	0	$224 \cdot BC$	$24 \cdot BC$	18

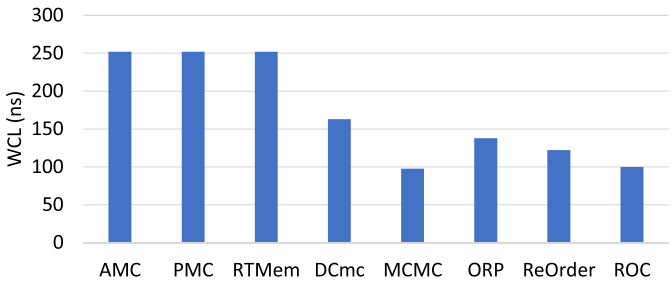


Fig. 4 Worst-case analytical latencies for the DDRx controllers for DDR3-1600 (1 cycle = 1.5 ns)

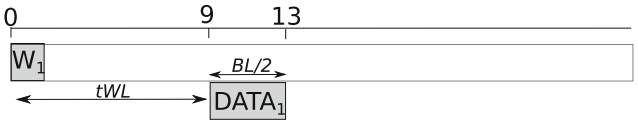


Fig. 5 Best-case analytical latency for the DDRx controllers with open-page policy for DDR3-1600 (1 cycle = 1.5 ns)

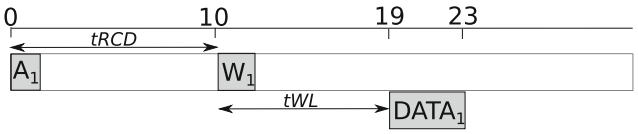


Fig. 6 Best-case analytical latency for the DDRx controllers with close-page policy for DDR3-1600 (1 cycle = 1.5 ns)

3.3.2 Best-case latency

We compute the BCL assuming the considered request does not suffer from any additional delays due to other requests. In other words, the BCL is due to timing constraints for only commands of the request under analysis. We have two cases, based on the controller type. For open-page controllers, in best case, a request will consist of a single R (or W) command. Therefore, the data will start transferring on the bus as soon as the corresponding tRL (or tWL) constraint is satisfied. From the JEDEC DDR DRAM standard [e.g. for DDR3 (2018) as shown in Table 1] , $tWL \leq tRL$. Therefore, in best case the request is a W request. Figure 5 delineates this scenario. Accordingly, the BCL for open-page controllers is computed as

$$BCL^{Open} = tWL = 9 \text{ cycles} = 13.5 \text{ ns}.$$

On the other hand, for close-page controllers, the request has to start with an Activate (A) command followed by a write command W after $tRCD$ cycles. Figure 6 illustrates this scenario. Hence, the BCL for close-page controllers is computed as

$$BCL^{Close} = tRCD + tWL = 19 \text{ cycles} = 28.5 \text{ ns}.$$

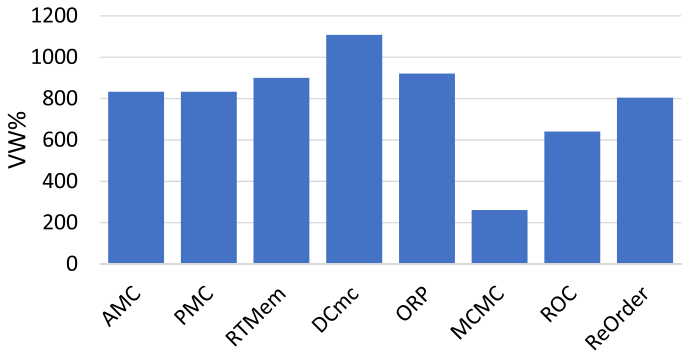


Fig. 7 Analytical variability window for different predictable memory controllers

3.3.3 Variability window

Based on the derived WCL and BCL, the final step is to calculate the analytical VW using Definition 3. Figure 7 delineates the VW values for the considered DDRx controllers.

As Fig. 7 illustrates, the variability window of these predicable controllers is huge. It exceeds 800% in 6 out of the 8 studied controllers. We observe that controllers with multi-rank support (MCMC, ROC, and ReOrder) provide less variability window. For instance, MCMC has the least variability window of 261%. This is because these controllers mitigate the interference among different PEs by partitioning banks among PEs as well as mitigates the bus switching delays by alternating between different ranks. However, the is still large and can be ill-suited for real-time systems with tight safety-critical timing requirements.

As aforementioned, this high variability is due to the physically inherent limitations of the DDRx memories that induce large timing constraints, which all controllers have to satisfy. As a result, we believe that exploring other types of off-chip memories that address these limitations is unavoidable towards providing more predictable memory performance with less variability and tighter bounds.

4 RLDRAM for real-time systems

RLDRAM is an emerging DRAM currently led by Micron (Micron Technology Inc. 2018a) and provides a remarkable lower access latency compared to DDRx protocols. RLDRAM has a similar structure to DDRx as depicted in Fig. 1. Nonetheless, RLDRAM achieves lower access latency by adopting unique architecture features that do not exist in commodity DDRx DRAMs. Two major features are of particular interest. First, RLDRAM uses an SRAM-like non-multiplexed address mode. All address bits are provided to the memory in one step as opposed to the two-step multiplexed mode in DDRx. Second, the row management through activation and precharging is handled internally by the RLDRAM device instead of the memory controller in case of DDRx. These two features together lead to multiple advantages of RLDRAM: (1)

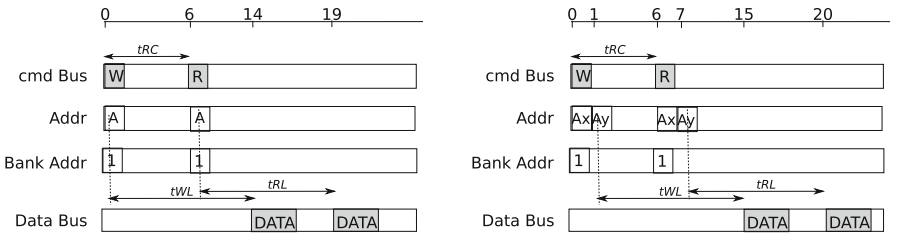
Table 3 Timing constraints for RLDRAM3 (Micron Technology Inc. 2018a)

Parameter	Delay description	Cycles
tRC	Minimum time between two commands to same bank	6
tWL	Minimum time between W to start of data transfer	14
tRL	Minimum time between R to start of data transfer	13
$BL/2$	Minimum time between two commands of same type to different banks	4 (BL = 8)
$tRL - tWL + BL/2$	Minimum time between R to W commands to different banks	3 (BL = 8)
$tWL - tRL + BL/2$	Minimum time between W to R commands to different banks	5 (BL = 8)

simplifying the access protocol, as accesses to RLDRAM consist of only R or W commands; (2) achieving low random access delay (tRC), which has a direct effect on worst-case access latency; and (3) decreasing bus turnaround (W-to-R and R-to-W) delays. Table 3 lists the most relevant timing constraints of RLDRAM3 running at 1600 MHz. Overall, these advantages enable RLDRAM to provide a significant reduction in access latency, while incurring less variability as compared to DDRx memories as we illustrate in next subsection. In addition to the described SRAM-like non-multiplexed address mode, RLDRAM also supports a DRAM-like multiplexed address mode. In the latter, the address is divided into two parts and is provided to the RLDRAM device in two consecutive cycles. In the next subsection, we discuss the key differences between the two modes.

4.1 Multiplexed vs non-multiplexed addressing in RLDRAM

RLDRAM devices can be configured to use one of two addressing modes: multiplexed and non-multiplexed. This configuration is done through a mode configuration register in the device. In the non-multiplexed address mode (Fig. 8a), all the address bits are provided to the memory at one cycle with the access command and bank bits. In Fig. 8a, the address of the write request is provided at cycle 0 and the address of the read request is provided at cycle 6. On the other hand, in the multiplexed address mode (Fig. 8b), the request to the RLDRAM consists of two cycles. In the first cycle, the



(a) Non-multiplexed address mode. (b) Multiplexed address mode.

Fig. 8 Effects of address modes in RLDRAM

command, the bank bits, and a portion of the address bits are sent to the RLDRAM. This is shown in Fig. 8b at cycle 0 for the write request and at cycle 6 for the read request, where Ax is the first portion of the address. Then, in the subsequent cycle, the remaining portion of the address bits are sent to the RLDRAM. This is the Ay portion in Fig. 8b at cycles 1 and 7 for the write and read requests, respectively.

4.1.1 Effects of multiplexed addressing on RLDRAM access latency

Using the multiplexed address mode affects the RLDRAM latency by imposing certain rules on the timing constraints as follows.

1. tRC timing constraint between two commands (cmd_1 and cmd_2) to the same bank is not affected. It will still be measured from the cycle of issuing cmd_1 to the cycle of issuing cmd_2. This is because in the multiplexed address mode, commands are issued in the first cycle with the bank address and the Ax portion of the address as aforementioned. This is shown in Fig. 8a, b by having the same behavior for tRC in both addressing modes.
2. tRL and tWL in a multiplexed address mode has to be measured from the cycle of issuing the Ay portion of the address as Fig. 8b delineates. This increases the effective delay between the R/W command and the start of its data transfer by one cycle. This explains why data transfer of the write and read requests start at cycles 15 and 20, respectively in Fig. 8b as opposed to starting at cycles 14 and 19 in Fig. 8a.
3. From Table 3, the minimum separation between any two commands of same type (i.e. both are reads or writes) accessing different banks has to be $BL/2$, where BL is the burst length. This is necessary to ensure no collision on the data bus since the time taken to transfer the data on the bus is $BL/2$. This still applies to the multiplexed address mode, with one exception. If $BL = 2$, that means those two commands can be issued in consecutive cycles. However, this is not possible in a multiplexed address mode since each request takes effectively two cycles instead of one to be issued to the RLDRAM. Accordingly, in case of $BL = 2$, the minimum separation between two commands of same type to different banks is effectively 2 cycles. This can be generalized by stating the following rule: *the minimum separation between two commands of same type to different banks under a multiplexed address mode is* $MAX(BL/2, 2)$.

Table 4 Differences in timing constraints between multiplexed and non-multiplexed address modes for RLDRAM3 (Micron Technology Inc. 2018a)

Parameter	Non-multiplexed address	Multiplexed address
tRC	Same behavior as explained in Table 3	
tWL	Measured from the W command to the start of data transfer	Measured from the Ay portion of the address (one cycle after the W command) to the start of data transfer
tRL	Measured from the R command to the start of data transfer	Measured from the Ay portion of the address (one cycle after the R command) to the start of data transfer
R-to-Ror W-to-W	$BL/2$	MAX($BL/2$, 2)
R-to-W	MAX($tRL - tWL + BL/2$, 1)	MAX($tRL - tWL + BL/2$, 2)
W-to-R	MAX($tWL - tRL + BL/2$,1)	MAX($tWL - tRL + BL/2$, 2)

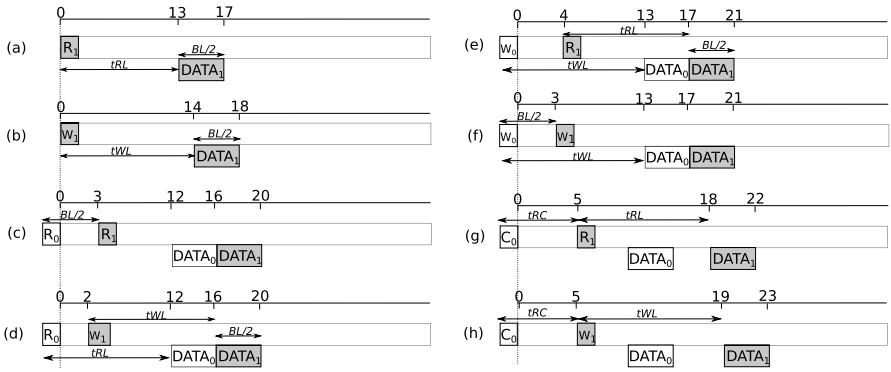
4. From Table 3, the minimum separation between a R and a W command accessing different banks has to be $tRL - tWL + BL/2$. Similar to the situation explained in the previous point, this can lead to a separation between the two commands that is less than 2 cycles (since $tRL < tWL$). As aforementioned, this is not possible in multiplexed address mode. Therefore, under the multiplexed address mode, we modify the R-to-W constraints for commands accessing different banks to be: $MAX(tRL - tWL + BL/2, 2)$. Similarly, the W-to-R delay changes to $MAX(tWL - tRL + BL/2, 2)$.

Table 4 summarizes the differences between multiplexed and non-multiplexed address modes with regard to timing constraints.

4.1.2 Advantages of multiplexed address mode

From this discussion, it is clear that the multiplexed address mode can have bad effects on the access latency of RLDRAM as well as the effective bandwidth. Nonetheless, the multiplexed address mode has two advantages.

1. It reduces the number of address pins used by the memory controller, which reduces the area, and hence, the cost of the memory system.
2. It allows the RLDRAM interfacing to be compatible with existing DRAM controllers in terms of addressing.



(a)	R_1 satisfies all constraints
(b)	W_1 satisfies all constraints
(c)	R_1 Arrives after a read access to a different bank
(d)	W_1 Arrives after a read access to a different bank
(e)	R_1 Arrives after a write access to a different bank
(f)	W_1 Arrives after a write access to a different bank
(g)	R_1 Arrives after a read or write access to same bank
(h)	W_1 Arrives after a read or write access to same bank

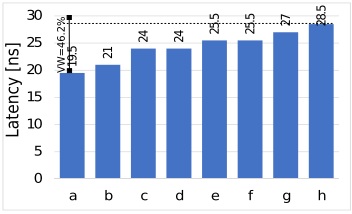
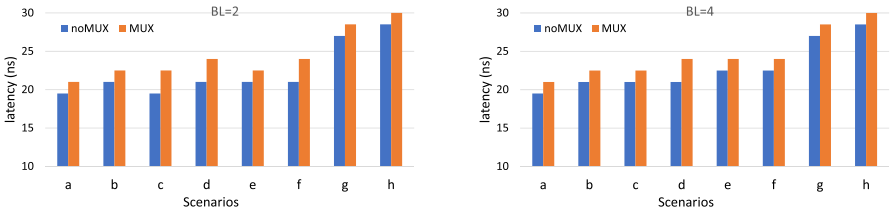


Fig. 9 Different RLDRAM access scenarios. Timing constraints are relatively scaled based on the timing constraints of RLDRAM3-1600 with a clock of 1.5 ns (Table 3). Subscripts are for request numbers

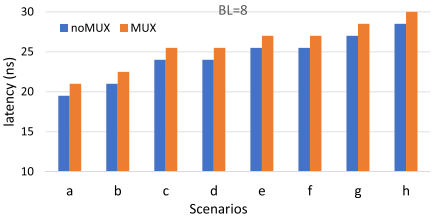
4.2 RLDRAM variability window

Figure 9 delineates possible access scenarios of a request to RLDRAM similar to Fig. 2 for DDRx. The chart in the bottom right shows the latency of the considered request for each scenario, while the table in the bottom left briefly explains each scenario. Unlike DDRx, RLDRAM is more deterministic, which explains the small number of access scenarios. Figure 9a, b depicts the best case scenario for a read (write) request that satisfies all timing constraints upon arrival. In Fig. 9c, f, the considered request arrives at the head of the queue directly after the memory started servicing a previous request of the same type to a different bank. Thus, it is delayed by $BL/2-1$ cycles. In Fig. 9d, e, the request arrives at the head of the queue directly after the memory started servicing a previous request of a different type to a different bank. Therefore, W_1 in Fig. 9d has to be separated from the previous R_0 by a R-to-W delay of $tRL+BL/2-tWL$. Similarly, R_1 in Fig. 9e has to be separated from the previous W_0 by $tWL+BL/2-tRL$, which is the W-to-R bus turnaround time. Finally, Fig. 9g, h show the worst-case scenario, which is a bank conflict. The considered request arrives after the memory started servicing a request to the same bank; thus, it has to be delayed by $tRC-1$ cycles. Since the type of the previous request is irrelevant in Fig. 9g, h, C_0 indicates either a R_0 or W_0.



(a) BL=2.

(b) BL=4.



(c) BL=8.

Fig. 10 Access latency for different scenarios, addressing modes, and burst lengths

4.2.1 Variability window

The scenario in Fig. 9a incurs the *BCL*, which equals to 13 cycles. Contrarily, the scenario in Fig. 9f encounters the *WCL*, which equals to 19 cycles. As a result, the variability window for the given sequence to the RLDRAM3-1600 is 46.2% as the chart in Fig. 9 delineates.

Compared to DDRx, RLDRAM provides 13.4× reduction in the latency variability and 3.79× reduction in the worst-case access latency. This identifies RLDRAM as promising solution towards providing a main memory with better predictable behavior and tighter WCL for real-time systems.

4.2.2 VW for different addressing modes and burst lengths

The scenarios delineated in Fig. 9 and discussed in Sect. 4.2.1 assumes an RLDRAM configured with a non-multiplexed address mode and uses $BL = 8$. In this section, we study the access latency of RLDRAM assuming the same scenarios, while exploring different address modes (both multiplexed and non-multiplexed) as well as various burst lengths; namely, $BL = 8, 4$ and 2. For each combination of addressing mode and BL, we construct the command interactions of each scenario similar to Fig. 9. Then, we calculate the access latency of the second request, which we plot in Fig. 10.

We highlight two main observations from the results shown in Fig. 10.

1. The non-multiplexed address mode increases the RLDRAM access latency by only one cycle in most scenarios. This one cycle is again because the address is provided to the RLDRAM device in two cycles and the read/write latency (tRL/tWL) is measured from the second cycle (Table 4).

2. There are cases where the multiplexed address mode increases the access latency by two cycles as compared to the non-multiplexed mode. This occurs for $BL = 2$: scenarios *c*,*d*, and *f*, and $BL = 4$: scenario *d* in Fig. 10. This can be explained as follows.
   - For $BL = 2$, consecutive commands in non-multiplexed address mode can be issued in consecutive cycles if they are of the same type since $BL/2 = 1$ (the case for scenarios *c* and *f*), or a R followed by a W since $tRL+tWL-BL/2 = 0$ and to prevent command bus conflict they can be issued consecutively one after the other (the case for scenario *d*). On the other hand, consecutive commands in multiplexed address mode have to be separated by a minimum of two cycles (Table 4).
   - For $BL = 4$ in a non-multiplexed address mode, consecutive R and W commands are separated by $tRL + tWL - BL/2 = 1$, while in a multiplexed address mode they have to be separated by two cycles.

   This adds one extra cycle of delay in addition to the one cycle added by the tRL/tWL as explained in the observation 1.

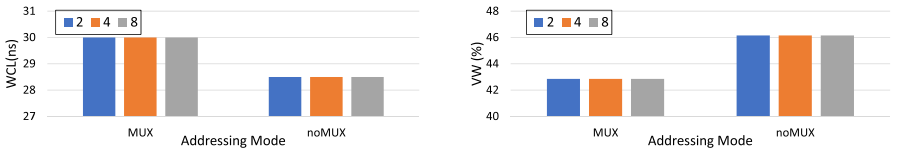
Studying access latency under all the aforementioned scenarios and combinations, we can conclude that the non-multiplexed address mode does not severely degrade the low access latency advantage of RLDRAM over commodity DDRx.

Worst-case latency We calculate the WCL for both addressing modes and different burst lengths and we depict the results in Fig. 11a. As Fig. 11a illustrates, the WCL of the multiplexed address mode is larger than that of the non-multiplexed address mode by one cycle (or 1.5 ns). As aforementioned, this is because in multiplexed address mode, the start of data transfer has to wait for 14/15 cycles after the issuance of R/W commands, respectively, as opposed to 13/14 cycles in case of the non-multiplexed address mode. We also observe that the WCL for a specific addressing mode does not change with the change of the burst length. From Fig. 9, the WCL (Fig. 9h) is $tRC + tWL$, which is independent of the burst length.

Variability window We also calculate the variability window under both addressing modes for different burst lengths and we depict the results in Fig. 11b. From Fig. 11b, the VW for the multiplexed address mode is 42.9%, and the VW of the non-multiplexed address mode is 46.2%. The multiplexed address mode has a lower VW because it has a higher BCL, while its difference between the WCL and BCL is the same as the non-multiplexed address mode (Eq. 1. Fig. 11b also shows that the VW of each addressing mode does not change with varying the burst length. The reason is that the burst length does not affect the best and worst case scenarios. From Fig. 9, the BCL (Fig. 9a) is tRL and the WCL (Fig. 9h) is $tRC + tWL$. Both are independent of BL; hence, the VW does not actually change across different burst lengths.

5 Predictable RLDRAM controller

To enable the usage of RLDRAM in real-time systems, we propose RLDC as a predictable RLDRAM memory controller that manages accesses to the RLDRAM.



(a) Worst-case latency. (b) Variability window.

Fig. 11 Variability window and WCL under both multiplexed and non-multiplexed address mode for $BL = 2, 4$ and 8

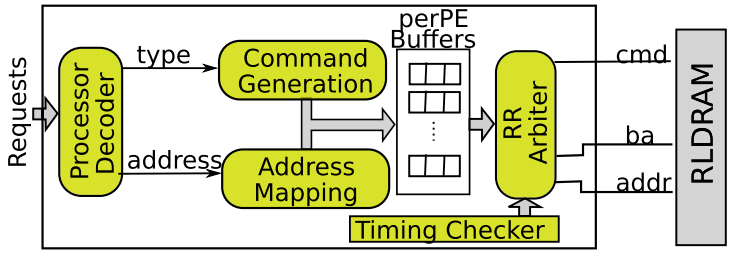


Fig. 12 High level architecture of RLDC

Figure 12 depicts the high-level architecture of RLDC. RLDC translates the memory requests into the corresponding RLDRAM commands and ensures the satisfaction of timing constraints amongst commands. It also predictably arbitrates amongst requests from PEs in a multi-processor system.

5.1 Bank partitioning vs bank sharing

Once a request is received by the memory controller, the Processor Decoder decodes the request's PE identification (Id) by using the PE bits encoded with the request. Then the Command Generation block generates the corresponding command by using the operation type of the request (read or write). Simultaneously, address translation is conducted by the Address Mapping block in Fig. 12 to determine which bank, row, and column to access. The proposed controller allows for two different memory layouts: bank partitioning and bank sharing. Bank partitioning partitions RLDRAM banks across PEs, where each PE obtains an exclusive access to specific bank(s). On the other hand, bank sharing allows each PE to access any bank. Bank partitioning reduces the interference among PEs, while bank sharing provides more flexibility. The user selects the layout through one bit in a memory configuration register inside the controller (not shown in Fig. 12). We conduct latency analysis for both mechanisms and experimentally compare their behaviors. For bank partitioning, the Address Mapping conducts the partitioning based on the Id provided by the Processor Decoder. Once a request's command is generated and its bank is calculated, this information is buffered in the corresponding PE queue to be scheduled by the predictable arbiter.

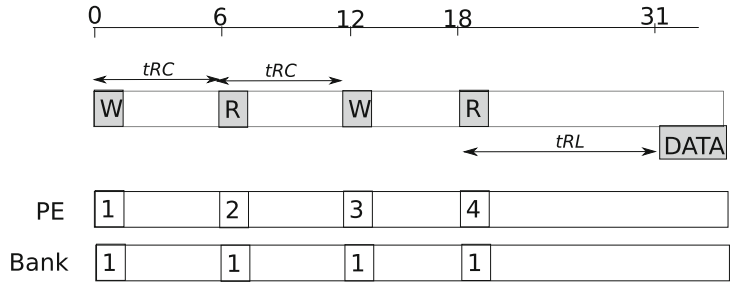


Fig. 13 WCL in a four-PE sysem with bank-sharing RLDC

5.2 Predictable arbitration

The proposed controller deploys Round Robin (RR) arbitration (RR Arbiter in Fig. 12) amongst requests at the head of each processor buffer (perPE Buffers). RR is a dynamic predictable arbitration mechanism that facilitates the latency analysis without sacrificing average-case performance. At the beginning of each cycle, the arbiter checks if the PE with the current slot in the RR schedule has a ready request to be sent to the RLDRAM. Timing Checker block decides if the request at the head of the queue of this processor satisfies the timing constraints of the RLDRAM and can be immediately serviced. This is conducted by maintaining a counter for each timing constraint. To exemplify, if RLDC issued a R to a bank, the tRC counter of that bank is initialized by the tRC constraint value. Hence, the Timing Checker ensures that no other command is issued to that bank before the tRC counter reaches zero. If the request is ready, the arbiter issues it to the RLDRAM in the form of a command (either R or W), a bank address, and request address. These are the cmd, ba, and addr signals in Fig. 12 at the interface between the controller and the RLDRAM. If the request is not ready due to timing violations, the arbiter checks the next PE in the schedule.

5.3 Latency analysis

We derive both worst- and best-case values for the total memory latency (Definition 2) incurred by any request to the proposed RLDRAM solution. The analysis is conducted for a multi-processor system. Although the proposed solution works for any pipeline architecture, we conduct the analysis assuming in-order PEs as they better represent PEs used for real-time systems. In addition, it is the commonly assumed pipeline type in predictable DDRx solutions including the ones we compare against (e.g. Hassan et al. 2015; Wu et al. 2013). For sake of generality, we derive the worst-case total latency for the two supported memory layouts: (1) bank sharing, where any PE has access to all banks (Lemma 1), and (2) bank partitioning, where banks are privately assigned to PEs (Lemma 2). Moreover, to enable the analytical calculation of the variability window, Lemma 3 provides the best-case total latency, which is the same for both bank partitioning and bank sharing mechanisms.

Lemma 1 *The worst-case total latency of any request to RLDC in a system with N PEs and a bank sharing layout can be calculated as follows (where tCL is tRL if Req_i is a read and tWL if Req_i is a write):*

$$WCL^{share} = (N-1) \times tRC + tCL.$$

Proof Recall that RLDC implements RR arbitration among PEs. As a consequence, the PE under analysis in the worst case waits for all other $N-1$ PEs before it is granted access. Additionally, since bank partitioning is not deployed, requests from different PEs can target any bank. In the worst case, requests from all PEs target the same bank such that only one command is serviced every tRC cycles. Accordingly, the PE under analysis has to wait for $(N-1) \times tRC$ cycles before it gains access to the RLDRAM. Once the command of the request under analysis is issued to the memory (R or W), the data transfer will start after tRL or tWL with respect to the type.

Figure 13 delineates this scenario for $N = 4$, where the WCL is $3 \cdot tRC + tRL = 3 \cdot 6 + 13 = 31$ cycles. □

It is worth noting that Lemma 1 applies for both addressing modes with the only difference that tCL is measured differently as explained in Sect. 4.1.

Lemma 2 *The worst-case total latency of any request to RLDC in a system with N PEs, a bank partitioning layout, and a non-multiplexed addressing mode can be calculated as:*

$$WCL^{part}_{noMUX} = \lceil \frac{N-1}{2} \rceil \times MAX\big((tWL - tRL + BL/2), 1\big) + \lfloor \frac{N-1}{2} \rfloor \times MAX\big((tRL - tWL + BL/2), 1\big) + tCL.$$

Proof From proof of Lemma 1, the request under analysis in the worst case waits for a request from each other $N-1$ PEs before it is granted access. Since bank partitioning is deployed, requests from PEs are guaranteed to access different banks (assuming that N is less than the number of banks). Accordingly, the only constraint is to separate every two successive commands in the RR schedule by the minimum delay required to avoid data bus collisions. Three cases are possible for any two successive commands to RLDRAM. (1) Both commands are of same type (either R or W). In this case, the minimum delay between these two commands is $BL/2$. Figure 9c, f represent this case. (2) The two commands are a write followed by a read. In this case, the minimum W-to-R delay is $tWL - tRL + BL/2$. Figure 9e represent this case. (3) The two commands are a read followed by a write. From Fig. 9d, these two commands have to be separated by a R-to-W delay of $tRL - tWL + tBUS$ cycles.

In the worst case, a data bus switching occurs between every two successive requests. Furthermore, since tWL is larger than tRL (Table 3), the W-to-R delay is larger than the R-to-W delay. Thus, the worst-case number of W-to-R switches is equal to or larger than R-to-W switches, which justifies the ceiling and flooring operations in Lemma 2. The MAX operations ensures that no bus collision occurs for the case

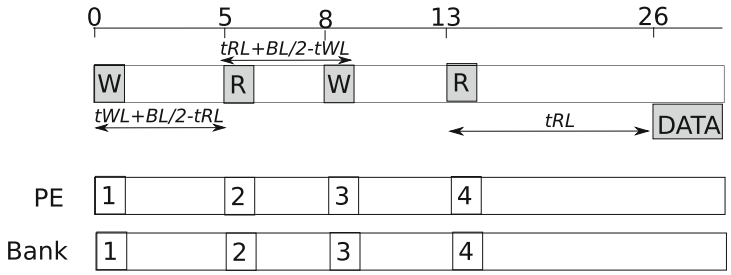


Fig. 14 WCL in a four-PE sysem with bank-partitioning RLDC

of $BL = 2$ since in $BL = 2$, $tRL - tWL + BL/2 = 0$ for RLDRAM devices with $tRL = tWL - 1$.

Figure 14 delineates this worst case scenario for $N = 4$ and $BL = 8$, which equals $2 \cdot (tWL - tRL + BL/2) + (tRL - tWL + BL/2) + tRL = 2 \cdot 5 + 3 + 13 = 26$ cycles. □

Corollary 1 *The worst-case total latency of any request to RLDC in a system with N PEs, a bank partitioning layout, and a multiplexed addressing mode can be calculated as:*

$$WCL_{MUX}^{part} = \left\lceil \frac{N-1}{2} \right\rceil \times MAX\big((tWL - tRL + BL/2), 2\big) + \left\lfloor \frac{N-1}{2} \right\rfloor \times MAX\big((tRL - tWL + BL/2), 2\big) + tCL.$$

Proof The proof directly follows from the proof of Lemma 2 by substituting the 1 in the MAX operations by 2. This is because in multiplexed addressing mode, any two commands has to be separated by a minimum of two cycles as detailed in Sect. 4.1. □

Lemma 3 *The best-case total latency of any request to RLDC, BCL, in a system with N PEs is calculated as:*

$$BCL = tCL.$$

Proof In best case, the request under analysis does not suffer any interference latency from other requests. Accordingly, its command is ready to execute upon arrival. Since there is a minimum of *tRL* (*tWL*) cycles between the R (W) command and the start of its data transfer, *BCL* is as calculated in Lemma 3. □

Similar to Lemma 1, *BCL* derived in Lemma 3 applies for both addressing modes with the only difference that *tCL* is measured differently as explained in Sect. 4.1.

From Lemmas 1–3 and Corollary 1, the variability window for bank sharing (under both addressing modes) is calculated in Eq. 4, while the variability windows for bank partitioning RLDC with non-multiplexed and multiplexed addressing modes are cal-

Table 5 Simulation environment configurations

PEs	4 PEs, in-order pipeline, a private 16KB L1 and a shared (but partitioned) 1 MB L2 cache
Main memory	Either RLDRAM or DDR
RLDRAM	RLDRAM3-1600 (Micron Technology Inc. 2018a) with timing constraints in Table 3, while the proposed RLDC manages accesses to RLDRAM
DDRx	DDR3-1600 with timing constraints with timing constraints in Table 1, while AMC, PMC, RTMem, DCmc, ORP, MCMC, ROC, or ReOrder manages access to DDR3
Bank Management	We experiment with both bank partitioning and bank sharing among PEs for RLDC

culated in Eqs. 5 and 6, respectively.

$$VW^{share} = \frac{(N-1) \times tRC}{tCL} \times 100 \tag{4}$$

$$VW^{part}_{noMUX} = \left(\lceil \frac{N-1}{2} \rceil \times MAX\big((tWL - tRL + BL/2), 1\big) + \lfloor \frac{N-1}{2} \rfloor \times MAX\big((tRL - tWL + BL/2), 1\big) \right) \times \frac{100}{tCL} \tag{5}$$

$$VW^{part}_{MUX} = \left(\lceil \frac{N-1}{2} \rceil \times MAX\big((tWL - tRL + BL/2), 2\big) + \lfloor \frac{N-1}{2} \rfloor \times MAX\big((tRL - tWL + BL/2), 2\big) \right) \times \frac{100}{tCL} \tag{6}$$

6 Evaluation

To evaluate the effectiveness of the proposed predictable RLDRAM solution, we use MacSim (Kim et al. 2012), a multi-processor architectural simulator integrated with DRAMSim2 (Rosenfeld et al. 2011) as the main memory system. We extend DRAMSim2 to faithfully model the RLDRAM operation and implement the proposed RLDC to manage accesses to the RLDRAM. We compare the proposed solution with eight of the state-of-the-art predictable DDRx controllers: AMC (Paolieri et al. 2009), PMC (Hassan et al. 2015, 2016), RTMem (Li et al. 2014), DCmc (Jalle et al. 2014), ORP (Wu et al. 2013), MCMC (Ecco et al. 2014), ROC (Krishnapillai et al. 2014), and ReOrder (Ecco and Ernst 2015; Ecco et al. 2016). On the integration of these controllers, we reuse the open-source implementation provided by Guo et al. (2018)

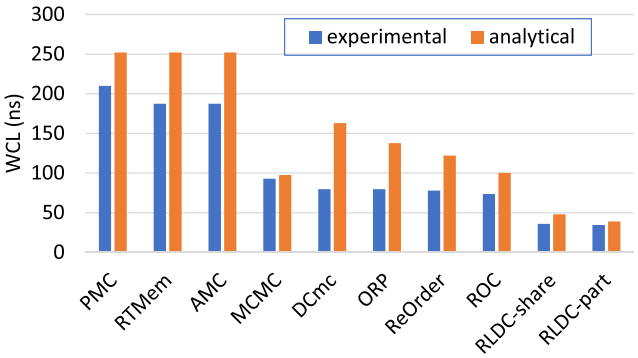


Fig. 15 Worst-case experimental and analytical latencies (ordered ascendingly by experimental WCL)

and Guo and Pellizzoni (2016). Table 5 tabulates the important system information. Since this work focuses on the off-chip memory delays, we deliberately configure the system to minimize interference (other than off-chip memory) among tasks. We assign each application a dedicated core that it solely uses until completion. In addition, we use cache partitioning to resolve the interference on the shared cache ($L2$ in this case). Cache partitioning is a common solution to cache interference (Gracioli et al. 2015). We use Benchmarks from the EEMBC-auto suite (Poovey et al. 2009), which includes representative applications from the embedded automotive domain. We use *a2time* benchmark as the application under analysis running on one PE. In our experiments, *a2time* has the following characteristics. It has a total of 2846 off-chip memory requests with a row hit ratio of 35%. The three other PEs are executing interfering applications. For these interfering PEs, we pick the three most memory extensive benchmarks from the EEMBC-auto suite: *matrix, aifftr, and aiifft*.

6.1 Worst-case latency

Figure 15 delineates both the experimental and the analytical WCLs for the DDRx and RLDRAM systems used in the experiments. Experimental WCL is the maximum total latency suffered by a request from the PE under analysis to the main memory. It is measured from the arrival time instance of the request into the controller to the time instance when the corresponding data of this request start transferring on the data bus. Analytical WCL of RLDRAM is the latency bound derived by the timing analysis conducted in Sect. 5.3.

For DDRx, we use the latency bounds derived in Sect. 3.3. RLDC-part in Fig. 15 indicates that RLDC is configured to use the bank partitioning mechanism, while RLDC-share indicates RLDC with a bank sharing mechanism.

6.1.1 Observations

1. All experimental WCLs are less than their corresponding bounds, which confirms that the derived bounds are safe.

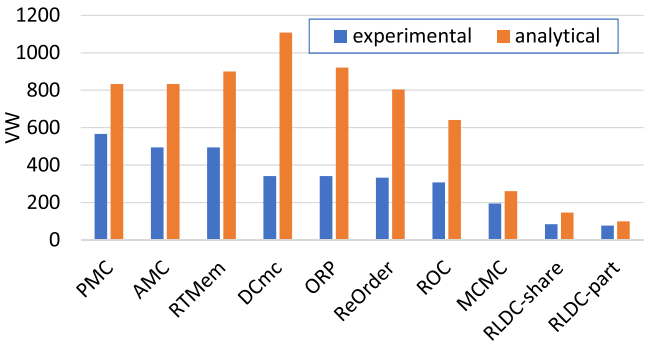


Fig. 16 Variability window (ordered ascendingly by experimental VW)

2. Clearly, the proposed RLDRAM solution provides a considerable less WCL compared to DDRx both experimentally and analytically. For instance, under bank partitioning mechanism, RLDC provides a WCL bound of 39 ns. On the other hand, WCL of DDRx varies from 97.5 ns for MCMC to 252 ns for PMC, RTMem, and AMC controllers. This is 2.5× and 6.46× higher than RLDC's WCL, respectively. Similar results are observed experimentally. The WCL of RLDC with bank partitioning is 34.5 ns. Minimum DDRx WCL of 73.5 ns is observed for ROC (2.13× RLDC's WCL), while the maximum WCL is observed for PMC and equals to 210 ns (6.09× RLDC's WCL). It is worth noting that this relatively low WCL of MCMC, ROC, and ReOrder as compared to other DDRx controllers relies on the existence of four DDRx ranks in the system. For a single-rank DDRx, those multi-rank controllers lose this advantage.
3. RLDC with bank partitioning provides tighter WCLs compared to bank sharing. This is because bank partitioning allows each PE to obtain an exclusive access to specific bank(s), which reduces the interference. This comes at the expense of lack of flexibility. For instance, unlike bank sharing, partitioning does not allow data sharing between PEs.
4. The gap between the analytical and experimental WCL is excessively higher for most of the DDRx controllers. This is because DDRx has larger number of commands and timing constraints between them. This complexity of the DDRx leads to nondeterministic behavior with wide variability window, which we study in the next experiment in more details.

6.2 Variability in total request latency

Figure 16 plots the experimentally observed variability window in the total memory latency for RLDRAM and DDRx using the same setup as in Sect. 6.1. The experimental variability window for each controller is calculated based on the observed best- and worst-case total latencies of this controller. For sake of comparison, we also plot the analytical variability window from Sect. 3.3.

6.2.1 Observations

1. Results show that a request to DDRx suffers from a significant variability in its latency and the variability window (Definition 3) is above 300% for seven of the eight considered DDRx controllers. The eighth controller (MCMC) has a variability window of 195.2%.
2. Contrarily, RLDC provides a considerable reduction in the variability window: 84.6% for bank sharing and 76.9% for bank partitioning. The explanation for this is that variability window is the relative difference between BCL and WCL. BCL occurs when the request suffers no interference at all form other requests. So its command execute immediately upon arrival. For both RLDRAM and DDRx, a request in best case consists of a single command (R or W). Accordingly, the BCL is either *tRL* or *tWL* for a read or write request, respectively. Since these two constraints are less in DDR3 than in RLDRAM3 (From Tables 1 and 3), DDR3 in fact has less BCL. On the other hand, because of the complexity of the DDRx protocol, WCL for DDRx is larger than that of RLDRAM as observed in Sect. 6.1. Accordingly, the variability window of DDRx is expected to be larger than RLDRAM3. This motivates the adoption of RLDRAM3 in real-time systems with its lower WCL and less variability.

6.3 Execution time

Figure 17 shows the overall execution time for each controller using the same setup as in Sects. 6.1 and 6.2. All results in Fig. 17 are normalized to the ROC controller since it is the one with best (least) observed execution time among considered DDRx controllers.

6.3.1 Observations

1. As Fig. 17 illustrates, improving WCLs reflects on the overall execution. RLDC achieves better execution time compared to all DDRx controllers, where the improvement ranges from 7% (compared to ROC) to 23% (compared to PMC).
2. It is worth noting that the gap between RLDC and DDRx controllers in execution time is notably less than the gap in the memory behavior (WCL and VW). This is because the total execution time composes of both computation time spent on processors and memory time spent on accessing data. RLDC only improves the off-chip memory access time. Therefore, the total effect on the overall execution of an application depends on the memory intensity of that application. We found that for this experiment setup, only 6% to 27% (depending on the controller) of the application time is consumed in off-chip memory access. Accordingly, we conclude that reducing overall execution time by 7% to 23% for such non-memory intensive application is a promising improvement.
3. We also observe that bank partitioning for RLDC offers a little improvement on the over all execution time than bank sharing in our experiment, with only 1% difference between RLDC-share and RLDC-part in Fig. 17.

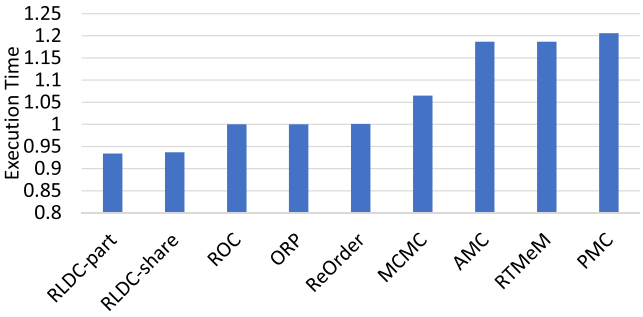


Fig. 17 Overall execution time. All results are normalized to ROC controller

6.4 Scalability: sensitivity to number of interfering PEs

In this experiment, we study the effect of varying the number of PEs in the system on both the analytical memory latency bounds as well as experimental WCL. Figure 18 depicts our findings.

6.4.1 Observations

1. Increasing the number of PEs, the gap in latency between RLDC and the majority of the considered DDRx controllers immensely increases. This is mainly because of the timing constraints that dominate the latency bounds, which also reflects the physical limitations of the DDRx memories. As explained in Sect. 4, RLDRAM does not suffer from the high latency of activation and precharging stages; thus, it has a lower *tRC* delay. In addition, accesses to RLDRAM does not suffer from the high data bus switching penalties that exist in DDRx memories as the switching delay in RLDRAM is only one cycle.
2. Some DDRx controllers show better scalability (less increase in WCL with the increase in number of PEs) than others. DDRx controllers in Fig. 18 can be classified into three categories. (a) Controllers with bank sharing mechanisms: AMC, PMC, and RTMem, which suffer the maximum increase in WCL when increasing the number of PEs. (b) Controllers with bank partitioning and multi-rank support: ReOrder, ROC, and MCMC, which incur the least WCL increase. This is because these controllers reduce interference among PEs by combining two techniques. First, they partition banks to eliminate the row conflict interference, which mitigates the long *tRC* delay. Second, they use the multi-rank support to amortize the data bus switching delays. (c) Controllers with only bank partitioning: ORP and DCmc, which exhibit intermediate increase in WCL.
3. Comparing all the aforementioned categories of DDRx controllers with RLDC (including the ones with bank partitioning and multi-rank requirement), highlights the advantages of RLDRAM for real-time systems. RLDC (whether with bank partitioning or sharing) encounters the least WCL across all PEs. This means that using RLDC, real-time systems enjoy tighter WCL with less sensitivity to

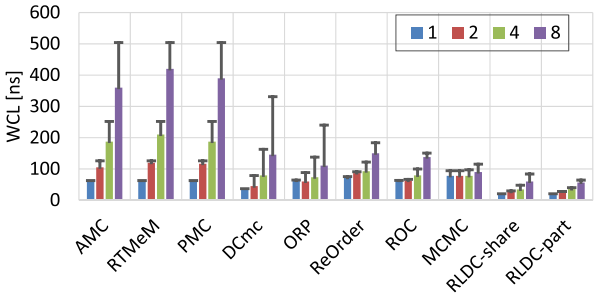


Fig. 18 Effect of varying number of PEs in the system. T-shaped bars are the analytical bounds and solid bars are the experimental WCLs

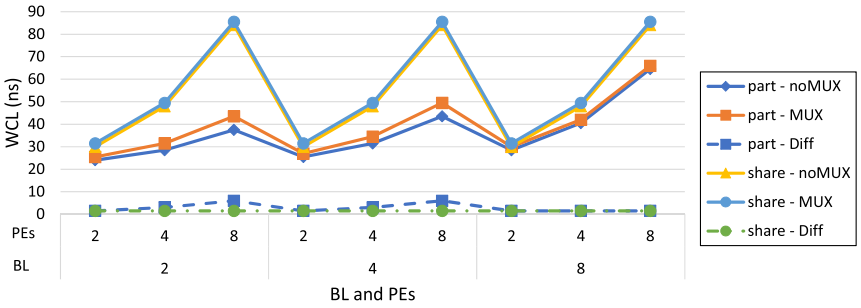


Fig. 19 Effect of addressing mode and burst length for different number of PEs

the number of interfering PEs, while having the flexibility of sharing data among different PEs.

6.5 Effects of addressing mode and burst length

We investigate the effects of the configured addressing mode (multiplexed vs. non-multiplexed) as well as for different burst length configurations on the latency bounds. Figure 19 delineates the analytical WCL for $BL = 2, 4$, and 8 in a system with 2, 4, and 8 PEs. As the legend shows, the plots are for different bank management schemes: partitioning (part) and sharing (share) as well as for different addressing modes: multiplexing (MUX) and non-multiplexing (noMUX). Figure 19 also delineates the difference of WCL between multiplexing and non-multiplexing addressing modes (Diff). Analyzing the results in Fig. 19, we make the following observations.

6.5.1 Observations

1. Generally, RLDC with the non-multiplexed address mode has a less WCL than RLDC with the multiplexed address mode for all BL and PE values under both bank partitioning and bank sharing schemes. This is expected and aligns with the analysis in Sect. 5.3. However, the difference between both addressing modes in WCL is minimal under most cases as we detail in the remaining observations.
2. *Under bank sharing scheme* the gap between multiplexed and non-multiplexed addressing modes is independent of BL and number of PEs and is equal to one

cycle or 1.5 ns (share-Diff in Fig. 19). This can be explained by Lemma 1, which shows that the WCL under bank sharing is $WCL^{share} = (N-1) \times tRC + tCL$. Accordingly, WCL^{share} is independent of BL's value. In addition, tRC is the same for both multiplexed and non-multiplexed addressing modes, while tCL of the multiplexed mode is larger by one cycle. Hence, the difference between both addressing modes is always one cycle.

3. *Under bank partitioning and* $\mathbf{BL = 8}$ the gap between multiplexed and non-multiplexed addressing modes is independent of number of PEs and is equal to one cycle or 1.5 ns (part-Diff in Fig. 19). This is because for $BL = 8$, both the R-to-W ($tRL - tWL + BL/2$) and W-to-R ($tWL - tRL + BL/2$) delays are larger than 2. Accordingly, from Lemma 2 and Corollary 1, both WCL^{part}_{noMUX} and WCL^{part}_{MUX} will have the same expression. The only difference between them is in tCL's value, which is larger for multiplexed address mode by one cycle.
4. *Under bank partitioning and* $\mathbf{BL = 4, 2}$: Unlike previous cases, for $BL = 2$ and 4, Fig. 19 illustrates that the gap between both address modes is dependent on the number of PEs in the system. This can be explained as follows. For $BL = 2$ or 4, $tRL - tWL + BL/2 < 2$; thus, from Corollary 1, the term $\lfloor \frac{N-1}{2} \rfloor \times MAX\big((tRL - tWL + BL/2), 2\big)$ will reduce to $\lfloor \frac{N-1}{2} \rfloor \times 2$ for the multiplexed address mode. On the other hand, for the non multiplexed address mode, the counterpart term in Lemma 2 reduces to $\lfloor \frac{N-1}{2} \rfloor \times 1$. This intuitively explains the dependence of the gap on the number of PEs (N). From Fig. 19, we also observe that this gap has a maximum of 4 cycles (or 6ns), which occurs in the case of $PE = 8$.

7 Other considerations: a discussion

We discuss some of the practical considerations towards adopting RLDRAM in real-time systems.

7.1 Cost

Compared to DDRx DRAMs, Static RAMs (SRAMs) provide a significantly lower latency at the expense of a significantly higher cost. This high cost prohibits the deployment of SRAMs in systems that require large capacity, which leaves the high-latency DRAMs as the only possible option. RLDRAM addresses this challenge by offering a balanced solution that provides a comparable latency to SRAMs, with a comparable cost to DDRx (Micron Technology Inc. 2018b). The lower latency is achieved by the means explained in Sect. 4. On the other hand, the lower cost than SRAMs is achieved by preserving the internal structure of DDRx, which consists of a single transistor as opposed to 6 in the case of SRAMs.

7.2 Density

Currently, the maximum density supported by RLDRAM3 is $2.25GB$ [5], while DDR3 offers up to $8GB$. Nonetheless, for real-time systems that require more than $2.25GB$ of data, multiple RLDRAM channels may be used.

7.3 Adoption

RLDRAM is manufactured by Micron Technology Inc. (2018b)), one of the biggest suppliers of memory devices. This ensures its stability and future adoption. It is also already adopted in high-speed networking solutions (Toal et al. 2007; Micron Technology Inc. 2018b). Moreover, RLDRAM is supported by several industry players such as Intel (2018), Xilnix (2018), Lattice Semiconductors (2018), and Northwest Logic (2018). As a result, state-of-the-art FPGA-based boards support RLDRAM interfacing [e.g. Intel Arria 10 GX FPGA (Intel 2018) and Xilinx Virtex UltraScale VCU110 (Xilnix 2018) development kits]. We believe that this support is an appealing opportunity since it provides the necessary means to design, experiment and evaluate RLDRAM solutions for future real-time systems.

7.4 Task-level analysis

In this paper, we derive an upper bound on the WCL incurred by any request to the off-chip memory. We also evaluate this WCL for the proposed RLDRAM solution as well as a breadth of available predictable DDRx solutions. This request-level bound can be used to either derive the WCET for a real-time task, or to conduct a response time analysis in a multi-tasking environment (Kim et al. 2014; Mancuso et al. 2017). In deriving such bound, we assume no knowledge about the the computation or memory access pattern of any of the running applications. It is worth noting that it can be possible to reduce off-chip memory interference, and hence improve the memory latency bounds by either enforcing certain constraints on the memory pattern, or make assumptions about such pattern. For instance, a system can enforce accesses to the off-chip memory to be streamlined through a DMA [e.g. by using a software-managed Scratchpad Memory (SPM)] (Soliman and Pellizzoni 2017). Another example is to shape the memory access pattern of PEs through throttling (Yun et al. 2013). These solutions are orthogonal to this work and can help in improving the memory bound; however, this improvement comes at the expense of compositionality of the task analysis (Hassan and Pellizzoni 2018).

8 Conclusions

The real-time community has been focusing on DDRx DRAMs as a sole solution for off-chip memories in real-time systems. We highlight the limitations of DDRx memories towards providing predictability in these systems. Then, we show that the emerging RLDRAM memory provides a promising solution that meets real-time requirements

with tighter latency bounds and less variability. To enable this deployment, we provide a predictable RLDRAM memory controller supporting multi-processor systems and conduct timing analysis to bound the latency suffered by any memory request. We compare the proposed solution with competitive predictable DDRx controllers. Results show that the proposed RLDRAM solution provides up to 6.4× reduction in the worst case memory latency and up to 11× less latency variability.

References

Akesson B, Goossens K, Ringhofer M (2007) Predator: a predictable SDRAM memory controller. In: IEEE/ACM international conference on hardware/software codesign and system synthesis (CODES+ISSS)

Chatterjee N, Shevgoor M, Balasubramonian R, Davis A, Fang Z, Illikkal R, Iyer R (2012) Leveraging heterogeneity in DRAM main memories to accelerate critical word access. In: IEEE/ACM international symposium on microarchitecture (MICRO)

Ecco L, Ernst R (2015) Improved DRAM timing bounds for real-time DRAM controllers with read/write bundling. In: Real-time systems symposium, pp 53–64

Ecco L, Ernst R (2017) Tackling the bus turnaround overhead in real-time SDRAM controllers. IEEE Trans Comput 66(11):1961–1974

Ecco L, Tobuschat S, Saidi S, Ernst R (2014) A mixed critical memory controller using bank privatization and fixed priority scheduling. In: IEEE international conference on embedded and real-time computing systems and applications (RTCSA)

Ecco L, Kostrzewa A, Ernst R (2016) Minimizing DRAM rank switching overhead for improved timing bounds and performance. In: Euromicro conference on real-time systems (ECRTS)

Goossens S, Akesson B, Goossens K (2013) Conservative open-page policy for mixed time-criticality memory controllers. In: Proceedings of the conference on design, automation and test in Europe. EDA Consortium, pp 525–530

Gracioli G, Alhammad A, Mancuso R, Fröhlich AA, Pellizzoni R (2015) A survey on cache management mechanisms for real-time embedded systems. ACM Comput Surv (CSUR) 48(2):32

Guo D, Pellizzoni R (2016) DRAMController: a simulation framework for real-time DRAM controllers. https://ece.uwaterloo.ca/%7Erpellizz/techreps/dramcontroller.pdf. Retrieved 2018

Guo D, Pellizzoni R (2017) A requests bundling DRAM controller for mixed-criticality systems. In: IEEE real-time and embedded technology and applications symposium (RTAS)

Guo D, Hassan M, Pellizzoni R, Patel H (2018) A comparative study of predictable DRAM controllers. ACM Trans Embed Comput Syst (TECS) 17(2):53

Hassan M (2018) On the off-chip memory latency of real-time systems: is DDR dram really the best option? In: IEEE real-time systems symposium (RTSS)

Hassan M, Patel H (2017) MCXplore: automating the validation process of DRAM memory controller designs. IEEE Trans Comput Aided Des Integr Circuits Syst (TCAD) 37(5):1050–1063

Hassan M, Pellizzoni R (2018) Bounding DRAM interference in COTS heterogeneous MPSoCs for mixed criticality systems. In: ACM SIGBED international conference on embedded software (EMSOFT)

Hassan M, Patel H, Pellizzoni R (2015) A framework for scheduling DRAM memory accesses for multi-core mixed-time critical systems. In: Real-time and embedded technology and applications symposium (RTAS), pp 307–316

Hassan M, Patel H, Pellizzoni R (2016) PMC: a requirement-aware DRAM controller for multi-core mixed criticality systems. ACM Trans Embed Comput Syst (TECS) 16(4): Article 100

Intel (2018) Arria 10 FPGA development kit, user guide (2018). https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/ug/ug%5Fa10-fpga-prod-devkit.pdf. Accessed 12 Sept 2018

Jalle J, Quinones E, Abella J, Fossati L, Zulianello M, Cazorla FJ (2014) A dual-criticality memory controller (DCmc): proposal and evaluation of a space case study. In: IEEE real-time systems symposium (RTSS)

JEDEC DDR3 SDRAM (2018) JEDEC DDR3 SDRAM specifications jesd79-3d. http://www.jedec.org/standards-documents/docs/jesd-79-3d. Accessed 12 Sept 2018

Kim H, Lee J, Lakshminarayana NB, Sim J, Lim J, Pho T (2012) MacSim: a CPU-GPU heterogeneous simulation framework user guide. Georgia Institute of Technology, Atlanta

Kim H, De Niz D, Andersson B, Klein M, Mutlu O, Rajkumar R (2014) Bounding memory interference delay in COTS-based multi-core systems. In: IEEE real-time and embedded technology and applications symposium (RTAS)

Kim H, Broman D, Lee EA, Zimmer M, Shrivastava A, Oh J (2015) A predictable and command-level priority-based DRAM controller for mixed-criticality systems. In: Real-time and embedded technology and applications symposium (RTAS), pp 317–326

Krishnapillai Y, Wu ZP, Pellizoni R (2014) ROC: a rank-switching, open-row DRAM controller for time-predictable systems. In: Euromicro conference on real-time systems (ECRTS)

Lattice Semiconductors (2018) Developing high-speed memory interfaces: the LatticeSCM FPGA advantage, a white paper. http://www.latticesemi.com/view%5Fdocument?document%5Fid=18926. Accessed 12 Sept 2018

Li Y, Akesson B, Goossens K (2014) Dynamic command scheduling for real-time memory controllers. In: Euromicro conference on real-time systems (ECRTS), pp 3–14

Mancuso R, Pellizzoni R, Tokcan N, Caccamo M (2017) WCET derivation under single core equivalence with explicit memory budget assignment. In: Euromicro conference on real-time systems (ECRTS)

Micron Technology Inc. (2018a) Micron RLDRAM3 SDRAM part mt44k64m18rb 093e. https://www.micron.com/~/media/documents/products/data-sheet/dram/576mb_rldram3.pdf. Accessed 12 Sept 2018

Micron Technology Inc. (2018b) Micron RLDRAM3 SDRAM product flyer. https://www.micron.com/%7E/media/documents/products/product-flyer/rldram3%5Fproduct%5Fflyer.pdf. Accessed 12 Sept 2018

Mutlu O, Subramanian L (2014) Research problems and opportunities in memory systems. Supercomputing Front Innov 1(3):19–55

Northwest Logic (2018) RLDRAM 3 controller core. https://nwlogic.com/products/docs/RLDRAM%5F3%5FController%5FCore.pdf. Accessed 12 Sept 2018

Paolieri M, Quiñones E, Cazorla FJ, Valero M (2009) An analyzable memory controller for hard real-time CMPs. Embed Syst Lett (ESL) 1:86–90

Phadke S, Narayanasamy S (2011) MLP aware heterogeneous memory system. In: IEEE design, automation & test in Europe conference & exhibition (DATE)

Poovey JA, Conte TM, Levy M, Gal-On S (2009) A benchmark characterization of the EEMBC benchmark suite. IEEE Micro 29(5):18–29

Reineke J, Liu I, Patel HD, Kim S, Lee EA (2011) PRET DRAM controller: bank privatization for predictability and temporal isolation. In: IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis (CODES + ISSS)

Rosenfeld P, Cooper-Balis E, Jacob B (2011) DRAMSim2: a cycle accurate memory system simulator. IEEE Comput Architect Lett (CAL) 10(1):16–19

Soliman MR, Pellizzoni R (2017) WCET-driven dynamic data scratchpad management with compiler-directed prefetching. In: Euromicro conference on real-time systems (ECRTS)

Toal C, Burns D, McLaughlin K, Sezer S, O'Kane S (2007) An RLDRAM II implementation of a 10Gbps shared packet buffer for network processing. In: NASA/ESA conference on adaptive hardware and systems (AHS)

Valsan PK, Yun H (2015) MEDUSA: a predictable and high-performance DRAM controller for multicore based embedded systems. In: Cyber-physical systems, networks, and applications (CPSNA), pp 86–93

Wilhelm R, Engblom J, Ermedahl A, Holsti N, Thesing S, Whalley D, Bernat G, Ferdinand C, Heckmann R, Mitra T et al (2008) The worst-case execution-time problem-overview of methods and survey of tools. ACM Trans Embed Comput Syst (TECS) 7(3):1–53

Wu ZP, Krish Y, Pellizzoni R (2013) Worst case analysis of DRAM latency in multi-requestor systems. In: Real-time systems symposium (RTSS), pp 372–383

Xilnix (2018) VCU110 evaluation board, user guide. https://www.xilinx.com/support/documentation/boards%5Fand%5Fkits/vcu110/ug1073-vcu110-eval-bd.pdf. Accessed 12 Sept 2018

Yun H, Yao G, Pellizzoni R, Caccamo M, Sha L (2013) Memguard: memory bandwidth reservation system for efficient performance isolation in multi-core platforms. In: IEEE real-time and embedded technology and applications symposium (RTAS)

Yun H, Mancuso R, Wu ZP, Pellizzoni R (2014) PALLOC: DRAM bank-aware memory allocator for performance isolation on multicore platforms. In: IEEE real-time and embedded technology and applications symposium (RTAS)

Yun H, Pellizzon R, Valsan PK (2015) Parallelism-aware memory interference delay analysis for COTS multicore systems. In: IEEE Euromicro conference on real-time systems

Mohamed Hassan is an Assistant Professor in the Electrical and Computer Engineer Department at the McMaster University, Canada. Previously, he worked as an assistant professor at University of Guelph and as an R&D SoC Lead Engineer at Intel. He received his MSc from Cairo University in 2012 and his PhD from University of Waterloo in 2017. Mohamed's research interests include real-time embedded systems, systems-on-chip architectures, hardware validation and security.