A Lean, Low Power, Low Latency DRAM Memory Controller for Transprecision Computing

Chirag Sudarshan[1], Jan Lappas[1], Christian Weis[1], Deepak M. Mathew[1], Matthias Jung[2], and Norbert Wehn[1]

[1] Technische Universität Kaiserslautern, Germany
{sudarshan,lappas,weis,deepak,wehn}@eit.uni-kl.de
[2] Fraunhofer Institute for Experimental Software Engineering (IESE), Germany
matthias.jung@iese.fraunhofer.de

Abstract. Energy consumption is one of the major challenges for the advanced System on Chips (SoC). This is addressed by adopting heterogeneous and approximate computing techniques. One of the recent evolution in this context is transprecision computing paradigm. The idea of the transprecision computing is to consume adequate amount of energy for each operation by performing dynamic precision reduction. The impact of the memory subsystem plays a crucial role in such systems. Hence, the energy efficiency of a transprecision system can be further optimized by tailoring the memory subsystem to the transprecision computing. In this work, we present a lean, low power, low latency memory controller that is appropriate for transprecision methodology. The memory controller consumes an average power of 129.33 mW at a frequency of 500 MHz and has a total area of 4.71 mm^2 for UMC 65 nm process.

Keywords: DRAM · DDR3 · Memory Controller · Transprecision · PHY.

1 Introduction

Approximate computing has been recognized as an effective technique to overcome the energy scaling barrier of computing systems by compromising the accuracy of results. Inspired by approximate computing, *transprecision computing* [1] has emerged as a new computing paradigm that offers a dynamic precision reduction to the intermediate computation stages in order to achieve higher energy efficiency without inheriting any errors in the final output. In other words, the accuracy of the final result is same as that of a traditional full-precision computing. The dynamic precision reduction is achieved by spanning all the layers of the computing system (i.e. from algorithm to the specifically tuned hardware that supports a variety of precision settings) and offering multiple control loops across these layers. The objective of H2020 European project OPRECOMP is to implement transprecision computing platforms for applications ranging from

Internet-of-Things (mW Platforms - ASIC) to *High Performance Computing* (KW platforms - FPGA).

OPRECOMP project uses the PULP (An Open Parallel Ultra-Low-Power Processing) platform [2] that is implemented using RISC-V cores for energy efficient computing. RISC-V is an open Instruction Set Architecture (ISA) that has acquired a lot of recognition across industry and academia. The high degree of customization offered in RISC-V architecture enables the design of energy efficient transprecision computation units. However, the ASIC implementation of the PULP is bound to low memory density devices like Static Random Access Memories (SRAMs). But, many applications demand larger memory footprints i.e external memories. The most prominent external memories are the Dynamic Random Access Memories (DRAMs). DRAM require a specialized circuitry called memory controller to manage the complex protocol. These memory controllers are designed for general purpose and are not available as an open hardware architecture. Additionally, DRAMs consume a major portion of the overall system power and deteriorate the system performance due to its long latency. This is partially due the DRAM operations such as refresh, activation and precharge (refer Section 2) that contribute high energy [3] and results long latency for the data accesses [4]. The impact of the refresh further increases for the next generation high density DRAM devices (64Gb devices) [5, 6].

In this work we focus on the mW platform that has a memory channel consisting of a single DDR3 DRAM device ($\times 8$ device). We present a DDR3 memory controller that includes several advanced features and optimizes the memory subsystem for transprecision computing.

One of the fundamental feature is to adapt the idea of approximate computing to the DRAM (i.e *Approximate DRAM*) [7–9]. The key knobs of approximating the DRAM is to vary the refresh rate in order to enable the trade-off between energy efficiency, performance and reliability. This technique allows the processing units to store the application data in an appropriate refresh/reliability zone (ranging from no refresh to high refresh rate) without incurring any computation errors. For example, the data are stored in no refresh zone if the lifetime of the application is less than the required refresh period of the DRAM [10]. A typical approximate DRAM employs a fine granular refreshing technique in order to refresh different zones of reliability at varied rates. The most favourable approach to realize a fine granular refresh is using *Optimized Row Granular Refresh (ORGR)* methodology [11]. The authors of [11] present a reverse engineering method to determine the user unknown minimum DRAM timings to realize the fine granular refresh that is as effective but more flexible than the conventional DRAM *Auto-Refresh*.

The second important feature is to exploit the application knowledge. General purpose memory controllers are confined to online scheduling techniques that only have a local view on the executed application. However, numerous applications feature deterministic memory access patterns, which can be exploited to improve bandwidth and energy. The authors of [4] present the methodology to generate an *Application-Specific Address Mapping* (ASAM), which has a global

view on the application and exploits the application knowledge to optimally map the data to a DRAM location that decreases the number of row misses, i.e. the number of precharge and activate operations. The authors of [4] showed upto 9x and 8.6X improvements in bandwidth utilization and energy efficiency by employing this technique.

However, all the discussed previous works (i.e. Approximate DRAM, ORGR, ASAM) presented the proof of concepts mainly by simulations. To the best of our knowledge for the first time our memory controller (i.e. frontend + physical layer or PHY) combines all the previously discussed advanced techniques. The designed DDR3 memory controller will be integrated with the PULP cluster to demonstrate the advantage of transprecision computing for IoT/embedded applications (mW platform). The key features of our memory controller are as follows:

- Lean, low latency and low power, optimized for embedded systems that apply the transprecision computing methodology.
- Enables fine granular refresh control using ORGR.
- Supports the exploitation of application knowledge using ASAM.
- Scalable and robust PHY design with an All-Digital-DLL (AD-DLL) that uses glitch-free delay-lines without special filters.

The paper is structured as follows: Section 2 gives a brief overview on DRAM and its operation. We present our implementation of the transprecision memory controller and PHY in Section 3 and Section 4 respectively. Section 5 discusses the post layout results and power estimation. Finally the paper is concluded in Section 6.

2 DRAM Background

In this section we first introduce the basic terminology and the operation of a DRAM device. DRAM devices are organized as a set of memory banks (e.g. eight) that include memory arrays. The banks operate concurrently (bank parallelism) with some constraints on data access due to the shared data and command/address bus. Accessing data from the DRAM is a two step process. First, the *activate* command (ACT) must be issued to the row of a certain bank. Then, the column access (CAS) i.e. *read* (RD) or *write* command (WR) are executed to read or write data from/to the specific column. The ACT command opens an entire row of the memory array and buffers in the *Primary Sense Amplifiers* that mimics a small cache, often called as row buffer. If a memory access targets the same row as the currently cached row (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 currently activated row (called row miss), it results in higher latency and energy consumption. If a certain row in a bank is active it must be *precharged* (PRE) before activating another row in the same bank. Additionally, to the normal RD and WR commands, there exist CAS commands with an integrated auto-precharge (RDA, WRA). If auto-precharge is selected, the row

Table 1: Key Timings for a DDR3-800D Device

Name	**Explanation**	**Value**
tRCD	*Row to Column Delay:* The time interval between ACT and RD on the same bank.	5 clk
tRAS	*Row Active:* The minimum active time for a row.	15 clk
tRTP	*Read-to-Precharge Delay:*The time interval between RD and PRE command on the same bank.	4 clk
tWR	*Write Recovery:* The minimum time interval between the end of a WR burst and a PRE command.	6 clk
tRP	*Row Precharge:* The time interval between PRE and ACT on the same bank.	5 clk
tRRD	*Row-to-Row Delay:* The minimum time interval between 2 consecutive ACT command to different bank.	4 clk
tCCD	*Column-to-column Delay:* The minimum time interval between 2 consecutive WR or RD command.	4 clk
tWTR	*Write-to-Read:* The minimum time interval between the end of a WR burst and a RD command.	4 clk
RL	*Read Latency:* Delay between the RD command and the availability of the first RD data bursts on the DRAM data interface.	5 clk
WL	*Write Latency:* Delay between the WR command and the availability of the first WR data bursts on the DRAM data interface.	5 clk
tRTW	*Read-to-Write:* The minimum time interval between the end of a WR burst and a RD command. $tRTW = RL + tCCD + 2clk - WL$	6 clk
tRFC	*Refresh cycle time:* The minimum time interval between the refresh command and any valid command.	110 ns
tREFI	*Refresh Interval:* The minimum time interval between the consecutive refresh commands.	7.8 us

being accessed will be precharged at the end of the read or write access. Further, the DRAM device is issued an *Auto-Refresh* (AREF) command at every tREFI duration that internally performs refresh operation. Table 1 shows the key timings of DDR3 DRAM device and its values as defined in [12].

3 Memory Controller Architecture

Fig. 1 shows the architecture of the transprecision memory controller. In this section, we describe on the architecture of the frontend. It is designed to satisfy the mW platform requirements i.e. low power, low area and low latency. The frequency ratio between the PHY and the frontend is 1:4, similar to state of the art memory controllers, such as [13, 14]. This allows the frontend to be operated at a lower clock frequency, satisfying the timing constraints and consuming lower power. In order to compensate the frequency difference and avoid

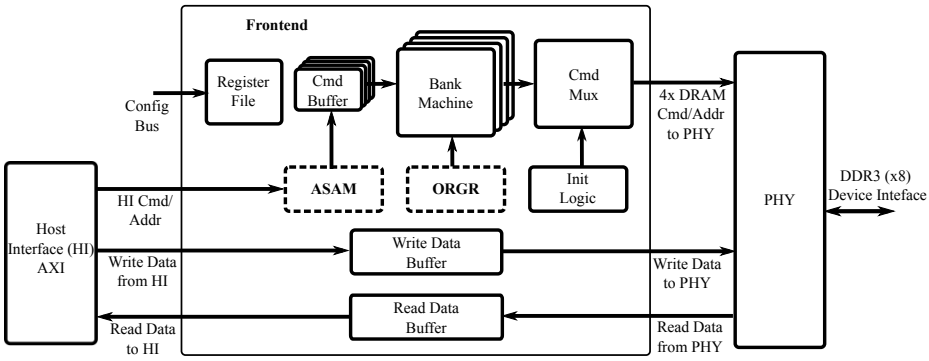


Fig. 1: Transprecision Memory Controller Architecture

stalling of the PHY, the frontend issues 4× DRAM commands/addresses (i.e. commands/addresses corresponding to next 4 PHY cycles) to the PHY.

As mentioned in Section 1, the ASAM technique presents better energy and bandwidth results as compared to an online scheduler for the applications with deterministic memory access pattern. Additionally, the online scheduler block requires large buffers for reordering the incoming requests and introduces a very high area overhead and latency penalty. Hence, this architecture does not integrate any online scheduler but rather employ ASAM block. The ASAM block is a dedicated address decoder that translates the incoming address bits from the *Host Interface* (HI) like AXI interface to an equivalent DRAM row, column and bank addresses as per the configured custom address map. The ASAM block incorporates a configurable address scrambling hardware as shown in Fig 2. The incoming logical address from the HI is typically 32 bit, out of which only 30 bits are valid since the maximum density of a DDR3 device is 8Gb. Note that the addresses from HI are byte addressable and the requests are in the granularity of a cache line (i.e. 8 bytes). The lower 3 bits of HI address are directly mapped to the lower 3 bits of the DRAM column address (C2-C0). The remaining 27 bits of the HI address are scrambled by the 27× 27 bit multiplexers to determine the DRAM addresses i.e. bank (B2-B0), row (R15-R0) and column (C10-C3). A typical general purpose memory controller also supports multiple HIs and has an *N-Port Arbiter* to prioritize the incoming request and delivers it to the scheduler. However, this arbiter would further add a lot of resources and latency to command processing. Thus, our controller is integrated with only one HI interface, which is sufficient for a typical embedded processors like mW platform.

The translated addresses and the corresponding HI command (i.e. read or write) are forwarded to one of the eight command buffers (FIFO) depending on the bank address. The *Bank Machines* (BM) consecutively process the incoming traffic associated with its respective DRAM bank. A BM keeps track of the current active row of its respective bank and translate the incoming transactions to a sequence of DRAM commands. The command sequence depends on

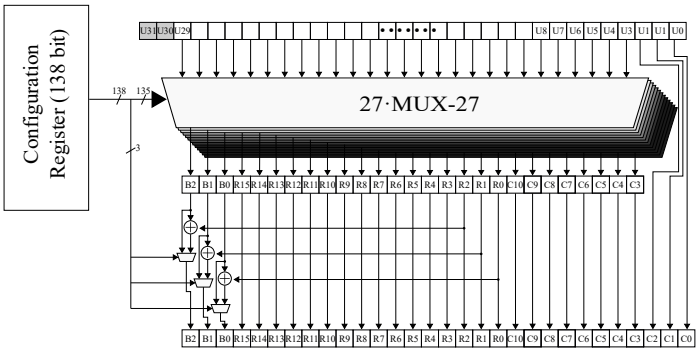


Fig. 2: ASAM Architecture

the current state of the bank and the target row of the incoming transaction. The BM also guarantees all bank specific timings such as tRCD, tRTP, tWR, tRAS and tRP. The *Command Multiplexer* (Cmd Mux) prioritizes the DRAM commands from multiple BMs, ensures that the bus related timings (inter-bank timing) like tWTR, tRTW, tRRD etc. are maintained, and packs 4× DRAM commands/addresses. The *Init* block handles the initialization sequence of the DRAM as specified in the DDR3 specification [12]. Until the initialization is finished the rest of the memory controller is stalled. The write data from the HI is stored in the write buffer and is forwarded to the PHY along with its corresponding write command. The read data that arrives RL clock cycles (i.e DRAM/PHY clock) later, is stored temporarily in read buffer and forwarded to processing unit via HI. The data bus width of the frontend is 64 bits i.e. DRAM Burst Length × DQ width (8 × 8). The configuration of the DRAM timings, mode register settings and other internal parameters required by the frontend and the PHY are done via the 8 bit *Configuration Bus* (Config Bus) that employs a custom protocol.

The *ORGR* block manages the DRAM refresh operation using optimized fine granular refresh technique. The ORGR block consists a set of counters to track the refresh interval of different DRAM zones of reliability. As the counter expires, the ORGR sends the corresponding BMs the row addresses to be refreshed. That appropriate BMs will stall its further transactions and services the ORGR request with the sequence of ACT and PRE commands (i.e. fine granular refresh) with reduced DRAM timings. These ACT and PRE commands are given the highest priority by the cmd mux. The reverse engineering methodology presented in [11] to identify the user unknown minimum timings is executed during the initialization. However, it is not triggered at every initialization due to the fact that the identified minimum timings in most cases remain unmodified for the entire course of operation of that device. Hence, it is triggered occasionally and for the rest of the initialization, the last known minimum timing values are configured via the config bus from the software. The config bus is also used to define the DRAM reliability zones and their respective refresh intervals.

4 DDR3 Physical Layer (PHY) Architecture

This new PHY is designed with the focus on simplicity and robustness. All full-custom atomic components are designed to be massively reused in the total design, to decrease design time, implementation time and time to test. The architecture of our DDR3 PHY is shown in Fig. 3. The PHY consists of two major blocks:

- Data Bus (see Fig.3 ①)
- Address/Command (ADDR/CMD) Bus (see Fig.3 ②)

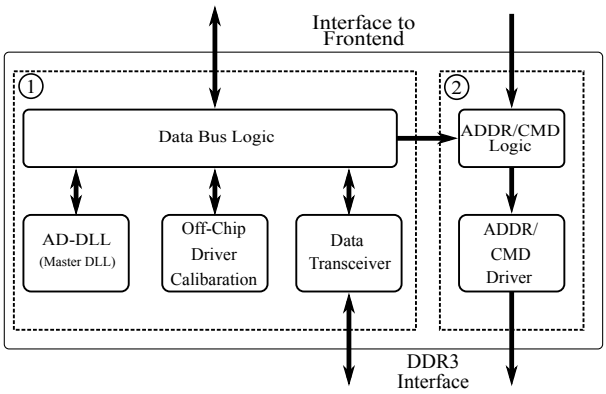
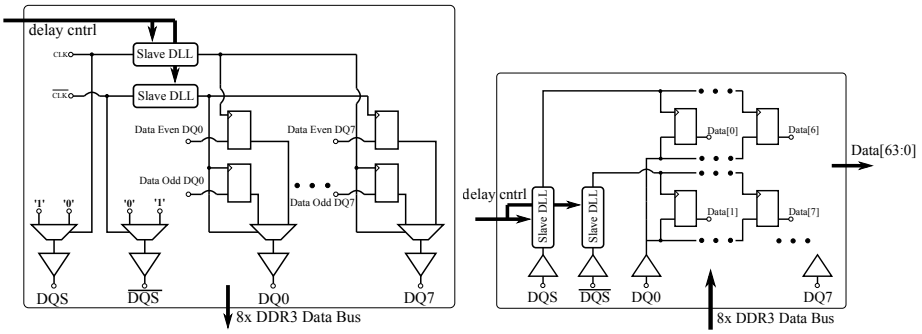


Fig. 3: DDR3-PHY Architecture

The bidirectional Data Bus consists of nine Data Transceivers, eight for data (DQ[7:0]) and one differential data strobe (DQS, nDQS). Each transceiver is implemented by seven parallel push-pull drivers where each of them has the pull-up (PMOS) and the pull-down (NMOS) path calibrated by the Off-Chip Driver Calibration Unit to $R_{dri} = 240\Omega$. The push-pull driver transistors are build with thick-oxide transistors (60Å gate oxide thickness). This allows a simple and robust ESD protection of the driver transistors and avoids complicated and fragile stacked transistor design with thin-oxide transistors as presented in [15]. This seven parallel push-pull drivers can be individual selected to set the required on-chip-termination value and to implement different driving strengths depending on write or read operations. The on-chip-termination impedance is then $(R_{dri}/N_{dri-sel})/2$ and off-chip-driver impedance $R_{dri}/N_{dri-sel}$. Where $N_{dri-sel}$ is the number of selected drivers. The overall driver architecture is based on [16].

The receivers for the data transceivers are single-ended receivers with their reference pin connected to the voltage V_{ref} (for DDR3 $V_{ref} = VDD/2 = 0.75V$). The data strobe receiver is implemented as a fully differential amplifier with a common-mode-voltage equal to V_{ref}. All receivers are biased locally to simplify the analog signal global routing. To reduce the power consumption bias circuits



(a) Write Path DDR3 PHY (b) Read Path DDR3 PHY

Fig. 4: DDR3-PHY Data Bus

are shared between two receivers. This PHY architecture implements also the 90° phase shift delay needed in DDR interfaces to center align the data strobe signals (DQS, nDQS) to the data signals (DQ) with a master-slave DLL configuration. After power up the Master DLL does a 360° lock to the internal clock. After locking the Data Bus Logic broadcasts the new configuration values from the Master DLL to the Slave DLLs. The Slave DLLs are built using a replica delay line from the Master DLL that represents a quarter of the Master DLLs delay. The Slave DLLs are placed inside the read and write path of the Data Bus (see Fig.4). The ADDR/CMD Bus consists of 26 single data rate drivers (ADDR, nCS, nWE...) and two clock drivers (CLK, nCLK). These drivers are using the same driver topology architecture as the Data Bus drivers but in a single data rate configuration. This allows to reuse the same impedance control values that are broadcasted by the Data Bus. The ADDR/CMD Bus is direct controlled by frontend of the memory controller. The ADDR/CMD Logic is only a thin abstraction layer that implements the serialization of the inputs and the configuration of the drivers.

4.1 All Digital Delay Locked Loop (AD-DLL)

An AD-DLL is selected for this DDR3 PHY due to its robustness and good scaling in deep-sub-micrometer CMOS processes., This enable fast design time and lower complexity compare to the analog counterparts [17]. The AD-DLL is composed of the following main components (see Fig.5):

- Phase Frequency Detector (PFD),
- DLL-Controller,
- four digital controlled delay lines (DCDL) with fine and coarse delay control.

An abstracted version of the Phase Frequency Detector used in this DLL is shown in Fig. 7. The Phase Frequency Detector (PFD) compares the leading edges of the reference clock (clk_in) with the delayed clock (clock coming from

the digital controlled delay lines). Depending if clk_in or the delay_clk is leading a small pulse on the internal up_int or down_int port will be generated. To enable the digital DLL Controller to detect these ports a RS-NAND latch is used to hold the last status. This kind of PFDs are common for All-Digital DLLs, but they have a major drawback to be very susceptible to glitches at their input. Missing clock edges due to glitches cause fault detection. This AD-DLL solves this problem by using digital controlled delay lines (DCDL) that suppress to generate glitches when switched to a different delay values. The coarse delay of the DCDL is constructed by using 32 glitch-free NAND-based delay element (DE) structure in a special three step switching scheme proposed by [18] (see Fig. 6a). The fine delay is implemented by two standard invertes and a RC-Delay where the capacitor is trimmable with a 4bit resolution. The trimmable capacitor is build out of MOSFETS were drain and source are connected to the signal. By switching the gate to VDD or to VSS the capacitor changes its value.

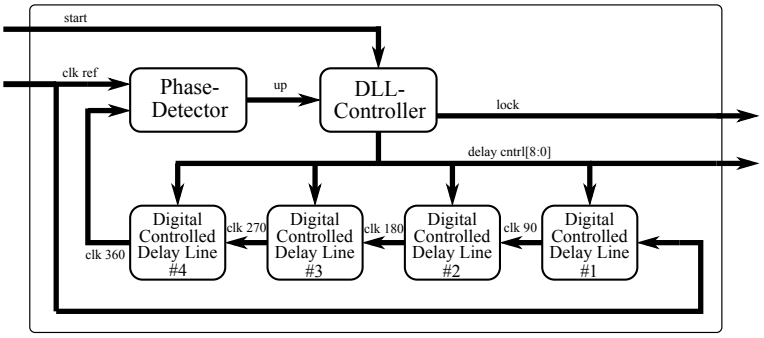
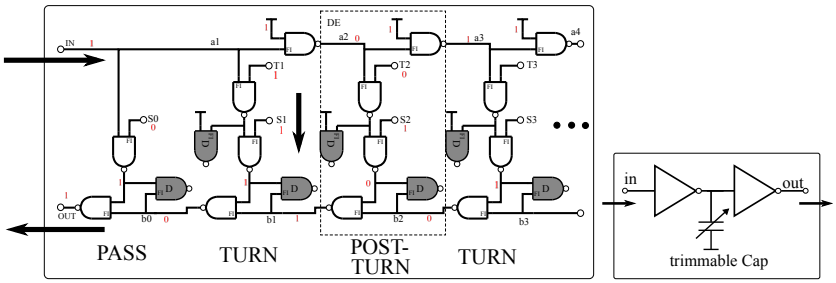


Fig. 5: All Digital Delay Locked Loop Architecture



(a) Coarse delay [18] (b) Fine delay

Fig. 6: Coarse and fine delay elements

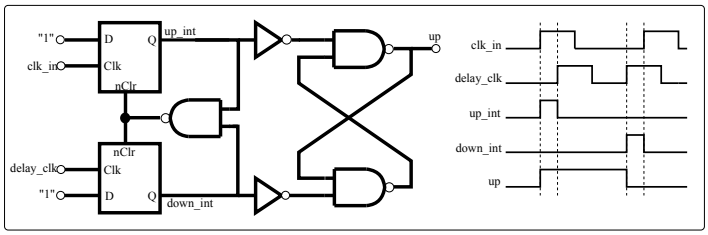


Fig. 7: Phase Frequency Detector (PFD) with Latch Output

5 Results

The memory controller is implemented using the UMC 65 nm Low-Leakage CMOS bulk technology. The synthesis, place and route, and timing/power analysis of the digital logic are carried out using Synopsis' DesignCompiler, IC Compiler and PrimeTime, respectively. The circuit level simulations and the full custom layout of the analog blocks of the PHY are done using Cadence Spectre and Virtuoso. The power estimation of the DRAM device is performed using Micron's Power Calculator [19]. The DRAM model provided by multiple vendors and the bit true model of our PHY are used for the functional verification (post-layout) of the DDR3 controller and the PHY. Fig 8 shows the floor plan and layout of our memory controller designed for mW platform. The pin pitch of the data IOs is 200 μm and the ADDR/CMD IOs have a pin pitch of 100 μm. The area distribution of the memory controller consisting of frontend, PHY digital logic and PHY IO transceiver is shown in the Table 2. The core of the memory controller i.e. frontend is is extensively lean, consuming only 2% of the total chip area.

Table 2: Area Distribution of the Controller and PHY

Component	*Area*
PHY-IO transceiver	$3.820\,mm^2$
PHY-Digital	$0.551\,mm^2$
Frontend	$0.339\,mm^2$

The post-layout results show that the DDR3 controller achieves a performance of 533 MHz (PHY clock) leading to a data rate of 1066 Mbit/pin/s under worst case condition (i.e slow process, low VDD and high temperature). The peak frequency of the design is limited due to the package (QFN64) used for the mW demonstrator. The controller has a very low latency of 3 frontend cycles plus 1 PHY clock for processing the host interface (AXI4) requests and delivering the associated commands to the DRAM.

The power consumption of a single I/O driver when subjected to 100% toggling rate for the frequencies 166 MHz and 500 MHz are 6.0 mW and 18.0 mW,

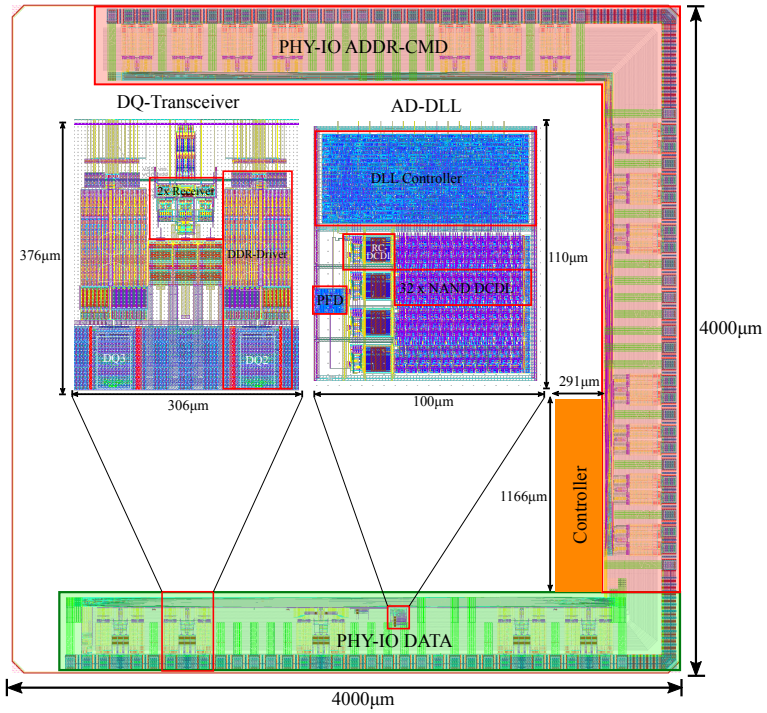


Fig. 8: Floor Plan

respectively. Similarly, a single data receiver consumes 3.75 mW and 5.5 mW power for the aforementioned frequencies. The power estimation of the memory controller is done using a random trace (for worst case estimation) with equal number of reads and writes as an input to the controller that resulted in a data bus utilization of 60%. Table 3 shows the distribution of the estimated power for the frequencies 166 MHz and 500 MHz. The power consumed by the ADDR/CMD block has no substantial difference with the DATA block power and the DRAM device power. This is true only for a single DRAM device memory subsystem. However, in a typical SO-DIMM architecture the DRAM power and PHY-IO DATA power will be predominant. The contribution of the DRAM power to the total power is low (or not the major contributor) due to the relatively good data bus utilization of 60% (less overhead - low number of row misses etc.) and that only a single DRAM device was used. Note that the vendors of commercial available DDR3 controllers do not disclose power and performance values of their IPs.

Table 3: Power Distribution of the Controller and PHY

Component	*Power at 166 MHz*	*Power at 500 MHz*
Frontend + PHY Digital	6.312 mW	18.764 mW
PHY-IO ADDR/CMD block	19.98 mW	59.96 mW
PHY-IO DATA block	15.96 mW	50.61 mW
DRAM Device (2 Gb)	24.0 mW	67.0 mW

6 Conclusion

In this work, we presented a memory controller that is tailored for IoT/embedded applications that leverage the transprecision computing methodology. This memory controller adapted several advanced techniques, such as approximate DRAM, sophisticated refresh policy, and optimal address mapping and data placement by exploiting application knowledge. These techniques allow the energy and performance optimization of DRAM subsystems. Experimental results show that the memory controller's design is lean, low latency and low power. Furthermore the presented design of the DDR3 PHY is scalable, low-complexity and robust even under worst case corner conditions (i.e slow process, low VDD and high temperature). Finally, with this memory controller design we enable open hardware platforms, such as RISC-V, to integrate external DRAM devices.

Acknowledgment

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). This work was also supported by the Fraunhofer High Performance Center for Simulation- and Software-based Innovation.

References

1. A. C. I. Malossi, et al. *The transprecision computing paradigm: Concept, design, and applications.* In 2018 Design, Automation Test in Europe Conference Exhibition (DATE), pages 1105–1110, March 2018.
2. D. Rossi, et al. *Energy-Efficient Near-Threshold Parallel Computing: The PULPv2 Cluster.* IEEE Micro, 37(5):20–31, Sep. 2017.
3. Christian Weis, et al. *DRAMSpec: A High-Level DRAM Timing, Power and Area Exploration Tool.* International Journal of Parallel Programming, 45(6):1566–1591, Dec 2017.
4. Matthias Jung, et al. *ConGen: An Application Specific DRAM Memory Controller Generator.* In Proceedings of the Second International Symposium on Memory Systems, MEMSYS '16, pages 257–267, New York, NY, USA, 2016. ACM.
5. Ishwar Bhati, et al. *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.

6. Jamie Liu, et al. *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.
7. Matthias Jung, et al. *Approximate Computing with Partially Unreliable Dynamic Random Access Memory - Approximate DRAM.* In Proceedings of the 53rd Annual Design Automation Conference, DAC '16, pages 100:1–100:4, New York, NY, USA, 2016. ACM.
8. Jan Lucas, et al. *Sparkk: Quality-Scalable Approximate Storage in DRAM.* In The Memory Forum, June 2014.
9. Song Liu, et al. *Flikker: Saving DRAM Refresh-power Through Critical Data Partitioning.* SIGPLAN Not., 46(3):213–224, March 2011.
10. Matthias Jung, et al. *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.
11. Deepak M. Mathew, et al. *Using Run-Time Reverse-Engineering to Optimize DRAM Refresh.* In International Symposium on Memory Systems (MEMSYS17), 2017.
12. Jedec Solid State Technology Association. *DDR3 SDRAM (JESD 79-3)*, 2012.
13. Cadence Inc. *Cadence Denali DDR Memory IP.* http://ip.cadence.com/ipportfolio/ip-portfolio-overview/memory-ip/ddr-lpddr, October 2014, last access 18.02.2015.
14. Synopsys, Inc. *DesignWare DDR IP.* http://www.synopsys.com/IP/ InterfaceIP/DDRn/Pages/, 2015, Last Access: 18.02.2015.
15. X. Fan et al. *ESD protection circuit schemes for DDR3 DQ drivers.* In Electrical Overstress/Electrostatic Discharge Symposium Proceedings 2010, pages 1–6, Oct 2010.
16. C. Yoo, et al. *A 1.8 V 700 Mb/s/pin 512 Mb DDR-II SDRAM with on-die termination and off-chip driver calibration.* In 2003 IEEE International Solid-State Circuits Conference, 2003. Digest of Technical Papers. ISSCC., pages 312–496 vol.1, Feb 2003.
17. S. Chen, et al. *An all-digital delay-locked loop for high-speed memory interface applications.* In Technical Papers of 2014 International Symposium on VLSI Design, Automation and Test, pages 1–4, April 2014.
18. D. De Caro. *Glitch-Free NAND-Based Digitally Controlled Delay-Lines.* IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 21(1):55–66, Jan 2013.
19. Micron. *DDR3 SDRAM System Power Calculator*, July 2011. last access 2014-07-03.