ECE 485/585
Winter 2016
Final Project

Simulate a DDR3 DRAM memory controller. Your model should assume a single requestor (CPU) and a single memory channel. Incoming memory requests are placed in a queue of as many as 16 pending requests. When the queue is full, no further memory requests can be accepted until a memory request is completed.

All memory addresses are 32-bits. Assume the DIMM has 32K rows, 8 banks, and 2K columns.

You can use any programming language you like to implement the simulation. Note that you are not doing a synthesizable design, though you must keep track of cycle counts.

Grading

There are a variety of policies a memory controller can employ in an attempt to improve performance (usually bandwidth, though sometimes worst-case latency or power).

The highest possible score you can receive depends upon the complexity of the policies that you implement. The table below summarizes them.

In-order, no access scheduling, closed page policy	80
In-order, no access scheduling, open page policy	90
First ready, access scheduling	95
Out-of-order, access scheduling	100

Note that your score can be lower (possibly significantly) if you have bugs: incorrect results, DRAM timing violations, etc. or your code is poorly written (difficult to understand, unreliable, difficult to maintain, hard to re-use).

You can receive extra credit for implementing refresh operations, further exploration of the design space (e.g. implementing and measuring performance of different out-of-order scheduling policies, request buffer size).

Your simulation does not need to model or support mode register programming, low power modes, or additive latency.

Scheduling policies

The in-order scheduling, closed page policy issues DRAM commands for memory requests in the order they are received and explicitly closes each page after every column access. It must maintain a buffer of pending memory requests because they may arrive faster than DRAM commands can be issued. When the buffer is full, no

further memory requests can be acknowledged until a pending memory operation completes and the request is removed from the queue.

The in-order scheduling, open page policy also processes requests in order. However, it doesn't explicitly close a page unless the queue is non-empty and the next in-order reference is to a different row in the same bank.

First ready, first access scheduling still processes requests in the order received. However, it can schedule individual DRAM commands from more than one in-flight request so that, for example, after issuing an activate command for one memory reference it can issue the activate command to another bank for another request before issuing the read command for the earlier reference.

Out-of-order policies can process memory requests out of order to optimize the DRAM command schedule. There are many possibilities. One example might be to schedule a memory read ahead of one that arrived earlier if it's to the currently open row in the same bank.

Academic honesty

You are free to consult the literature for ideas provided you cite all sources in your final report/presentation. You cannot make use of existing code or have anyone outside your team write code for you. Doing so will result in a code of conduct violation complaint and a score of 0 on the assignment.

Competition

All teams doing more than the minimum in-order, no access scheduling (closed or open page policy) can participate in a competition to achieve the best bandwidth on a benchmark that I will provide.

Teams with top three performance will be recognized and earn extra credit. The team with the top-performing predictor will win a small prize and be able to smugly mock their unworthy competitors.

Presentation and demonstration

Each team will demo their final project in a 30 minute time slot during finals week. You will provide a brief written report of your project as well as all documentation, testcases, and source code along with either a makefile (with default target) or README file with instructions for building and running your project. These should be packaged in a single zip file and uploaded to D2L 24 hours prior to your demo slot.

Verification (testing) is important. Be sure to include a description of your test strategy and testcases in your report.

No more than 16 pending DRAM requests can be held in the queue at one time.

Document all assumptions and design decisions

Inputs

You will read trace files of memory requests in the following format

```
0x2000D5C0 IFETCH  30
0x1FF96FC0 WRITE   160
0x2000D600 IFETCH  165
0x1FF97000 READ    192
0x2000A340 READ    278
```

Where the first column is the hexadecimal address of the memory reference, the second column is the type of operation (instruction fetch, data write, data read). You can treat both instruction fetch and data read operations as simply memory read operations. The third column is the "time" of the request in CPU clock cycles from the beginning of the program's execution.

Outputs

The output of your model will be a text file with a trace of all DRAM commands issued. The format of the DRAM trace file is as follows:

<CPU clock ticks> <DRAM command>

where <CPU clock tick> is an integer number of clock cycles from the start of execution and <DRAM command> is one of the following:

ACT <bank> <row>
PRE <bank>
RD <bank> <column>
RDAP <bank> <column>
WR <bank> <column>
WRAP <bank> <column>
REF

Assume the DRAM clock runs at ¼ the rate of the CPU clock.

Timing constraints

Your memory controller simulation must correctly schedule DRAM commands to comply with the following DDR-3 timing values. All values are given in DRAM clock cycles.

Parameter	Value
tRC	50
tRAS	36
tRRD	6
tRP	14
tRFC	172
tCWL (tCWD)	10
tCAS (CL)	14
tRCD	14
tWR	16
tRTP	8
tCCD	4
tBURST	4
tWTR	8

The meaning and use of these parameters can be found in the course lecture slides and in datasheets for DDR3 devices such as the Micron 2Gb MT41J512M.