ECE 485/585
Fall 2023
Final Project Description

Your team is responsible for the simulation of the scheduler portion of a memory controller capable serving a 12-core 4.8 GHz processor employing a single 16GB PC5-38400 DIMM. The system uses a relaxed consistency (XC) model. The controller uses the two channels of the DIMM independently. The DIMM is constructed with memory chips organized as x8 devices with a 1KB page size and 40-39-39-76 timing. These devices have 8 bank groups of 4 banks each. The DIMM uses 1N mode for commands requiring two cycles. There is no system-level ECC.

You can use any programming language (e.g. C, C++, Java) or hardware description language (e.g. SystemVerilog) you like to implement the simulation. Note that you are *not* doing a synthesizable design, though you must keep track of DRAM cycle counts.

Timing constraints

Your memory controller simulation must correctly schedule DRAM commands to comply with the following DDR-5 timing values. Unless otherwise indicated, all values are given in DRAM clock cycles.

Parameter	Value
tRC	115
tRAS	76
tRRD_L	12
tRRD_S	8
tRP	39
tRFC	295ns
CWL (tCWD)	38
tCAS (CL)	40
tRCD	39
tWR	30
tRTP	18
tCCD_L	12
tCCD_S	8
tCCD_L_WR	48
tCCD_S_WR	8
tBURST	8
tCCD_L_RTW	16
tCCD_S_RTW	16
tCCD_L_WTR	70
tCCD_S_WTR	52

The meaning and use of these parameters can be found in the course lecture slides and in Micron's datasheets for DDR5 devices such as https://media-www.micron.com/-/media/client/global/documents/products/data-sheet/dram/ddr5/ddr5_sdram_core.pdf

Do not assume that the hypothetical devices populating the DIMM in this assignment are the same in speed, size, organization, or timing to the device(s) described in this or other datasheets. This datasheet is provided only so that you can see how to interpret the parameters given above.

You do not need to support refresh, memory initialization or mode register programming. You can assume all banks start in the precharged state. Do not use auto-precharge. There is no additive latency.

The DRAM controller runs at the same clock speed as the processor. The interface to the controller serializes the requests from the cores so that the controller receives at most one request per clock cycle. The controller can potentially issue a DRAM command to each channel on every DRAM clock.

Input format

Your program must accept input trace files with the following format:

time core operation address

where *time* is the time of the request in (absolute) *CPU* clock cycles from the beginning of the trace execution, *core* is an integer between 0 and 11 indicating which core is making the request, *operation* is either 0, 1, or 2 to indicate the type of memory request being made (data read, data write, instruction fetch, respectively), and *address* is the hexadecimal address being referenced by the operation. Fields are separated by one or more spaces or tabs. All addresses will be 8-byte aligned. The last level caches employ critical word first and early restart.

For example:

```
30 0 2    01FF97000
31 1 2    01FF97080
32 2 0    10FFFFF00
40 0 1    10FFFFF80
```

The name of the input file should be specified at runtime. You should default to trace.txt if no filename is specified.

Output format

Your program must generate as output a text file with a trace of all DRAM commands issued using the following format:

time channel command

where *time* is the time in (absolute) elapsed *CPU* cycles when the DRAM command is issued, *channel* is the channel number the command is issued on, and *command* is one of the following:

```
ACT0 BG BA row
ACT1 BG BA row
PRE  BG BA
RD0  BG BA column
RD1  BG BA column
WR0  BG BA column
WR1  BG BA column
REF
```

The suffix 0 and 1 denote the first and second "halves" of commands that (because of multiplexing command and address pins) require two cycles. Although the BG and BA are sent with the first half and the row or column sent with the second half, I've included both on both halves to simplify your work.

BG and *BA* are bank group and bank respectively and are decimal integers. The *row* and *column* values are displayed in hexadecimal. All fields are separated by one or more spaces or tabs. For readability, you should display the time in a long fixed-width field so that the commands align. You may want to display the commands in fixed-width fields as well.

For example:
```
100   0 PRE   0 0
200   0 ACT0  0 0 03FF
204   0 ACT1  0 0 03FF
300   0 RD0   0 0 EF
304   0 RD1   0 0 EF
```

The output filename should be specified at runtime. You should default to dram.txt if none is specified.

GitHub

Your team must use GitHub to host your code. If you are unfamiliar with GitHub there are several very good tutorials at guides.github.com.

Grading

The project is worth 100 points. Your grade will be based upon:

- External specification
- Completeness of the solution (adherence to the requirements above)
- Correctness of the solution
- Quality and readability of the project report (e.g. specifications, design decisions)
- Validity of design decisions
- Quality of implementation
- Structure and clarity of design
- Readability
- Maintainability
- Testing
- Presentation of results and ability to explain your project and the underlying concepts and issues.

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).

Scheduling Algorithms

Assume the DIMM is already initialized and all banks are in the pre-charged state. The controller should maintain a queue of 16 outstanding memory requests. The memory controller *must* support channel parallelism. You can choose the level of sophistication (and difficulty). The number of points possible increases with the level.

- Level 0: no bank level parallelism, closed page policy (0 extra points)
- Level 1: no bank level parallelism, open page policy (10 extra points)
- Level 2: bank level parallelism, open page policy (20 extra points)
- Level 3: bank level parallelism, open page policy, out of order (25 extra points)

You can earn more credit by simulating more aggressive scheduling policies. If you do this, you must provide a run-time switch to enable/disable them. Examples of optional scheduling polices include: adaptive open page policy and out-of-order scheduling (e.g. to prioritize accesses to open row in the same bank or reads before writes, minimize read-to-write and write-to-read transitions). You can also receive extra credit for implementing refresh.

Level 1: The in-order scheduling, open page policy processes requests in order. 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.

Level 2: This 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/bank group for another request before issuing the read command for the earlier reference.

Level 3: Out-of-order policies can process memory requests out of order to optimize the DRAM command schedule. There are many possibilities. The simplest would be to allow a memory reference to a bank that could accommodate a request immediately even if it arrived after another memory reference (to another bank) that can't be started. Another 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. Another might schedule a read ahead of a write (provided they are not to the same address!). Out-of-order policies must incorporate an "aging"/timeout process so that a request isn't indefinitely passed over by later requests.

Academic honesty

You are free to consult the literature (e.g. papers in conferences and journals) for *ideas* provided you cite all sources in your final report/presentation. You cannot make use of any 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 for all team members.

Competition

All teams doing more than the minimum (in-order, no access scheduling, no bank parallelism, closed 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 controller 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 Canvas 24 hours prior to your demo slot.

The report should specify the address mapping you used and the scheduling algorithm and policies you implemented Verification (testing) is very important. Be sure to include a description of your test strategy and your testcases in your report.

Checkpoints

Because of the complexity of the project it's important that you get started early and make steady progress throughout the quarter. I've devised a series of early and intermediate checkpoints to help you with this. It has the added advantage that any major misunderstandings can be caught early when you can still correct them. Your team's performance during these checkpoints is part of your grade. The dates for each checkpoint will be shown on Canvas. Your team will need demonstrate to the course TA or instructor the capability indicated. I may revise the assignment and modify details of the checkpoints as the term progresses.

Checkpoint 1:

- Working code to read and parse an input trace file (the name of which is specified by the user) with correct default if none specified
- Debugging code (that can be enabled/disabled at runtime or compile time) to display the parsed values for each line
- Use of github repository
- Diagram showing physical to topological address (that is: how address bits are mapped to required fields (e.g. row, column, bank, bank group, channel)

Checkpoint 2:

- Working code to insert items into the queue(s)
- Stub to remove items from queue(s)
- Demonstrate maintaining correct (simulated) time
- Code generates correct output format for DRAM commands.
- What information needs to be maintained in the queue

Checkpoint 3:

- Present draft testplan
- Brief description of approach to testing
- Outline of tests to be performed
- Example test (and expected results)