Team 3 - ECE 585 Project - Final Report

Braden Harwood, Michael Weston, Stephen Short, Drew Seidel

https://github.com/ewerd/ECE585_Team3

December 6, 2021 - V1.0

Objective:

The purpose of this project was to simulate a memory controller as a part of a four core 3.2GHz processor employing a single channel memory. The memory controller communicates with a single rank 8GB PC4-25600 DIMM. For its most basic implementation, the memory controller utilizes bank parallelism and an open-page policy. Given an input text file of CPU requests (specifying time of request, type of request, and address needed of request), the simulator produces an output text file of the corresponding DRAM actions (precharge, activate, read or write), the CPU time at which they occurred, and corresponding DRAM location data when applicable (bank, bank group, row, column).

The memory controller will be implemented using a queue that can hold up to 16 outstanding requests. For the basic implementation, the memory controller's main objective is to prioritize the oldest request in the queue, while exploiting bank parallelism to peak at the remaining outstanding requests to determine what valid commands can be issued without violating timing constraints imposed by the DIMM. In order to make our memory controller the most efficient provided the given policies, we first had to make a few design choices. Then, in making sure our simulator was servicing CPU requests as efficiently as possible, we took into account DIMM timing constraints and tested methodically to ensure our memory controller was not violating constraints.

Design and Policies:

The first task was to determine how we wanted to efficiently map the CPU address to the DIMM. That is, the bits changing the most, should be near the LSB, and the bits that change the least should be towards the MSB. Operating under an open page policy with single channel memory, and after reviewing the timing constraints, we decided that the optimal memory mapping for servicing a DIMM while utilizing open page policy and bank parallelism is the following:

MSB: Rows[14:0] > Columns[10:3] > Banks[1:0] > Bank Groups[1:0] > Columns[2:0] > Byte Select[2:0]

The reasons for this choice are:

- Spatial locality - sequential code and data is likely to be fetched and accessed.
- Bank parallelism allows access to open pages while performing a precharge or activation on another. It is most efficient to spread these commands across bank groups, rather than banks within the same bank group when possible.

Another design decision was implemented across all versions, to mitigate bus conflict. We discovered that there was potential bus conflict when reading and then writing, but there was no designated read to write timing constraint.

Take the scenario where the memory controller issues a “read” to an open row in bank group 0. Based on the column read latency timing parameter, the DIMM will have the first bit of the read request available on the bus in 24 DRAM clock cycles. However, let’s say that 4 DRAM clock cycles after initiating the read we issue a write to open row in another bank group (satisfying tCCD_S). Based on the column write latency timing parameter, the DIMM will begin latching data from the bus 20 DRAM clock cycles later. This means that if we were to start this example at arbitrary DRAM time 0, the DIMM would be responsible for both writing to the bus, and latching data from the bus at DRAM time 24. To avoid this conflict all together, we created tRTW (read to write timing constraint) equal to 4 DRAM clock cycles (the burst length).

In the simulator itself, we aimed to have two main implementation builds. The simulators are as follows:

1. Default build: In order scheduling, open-page policy, bank parallelism.
2. Optimal build: Prioritize instructions over reads, and reads over writes. Utilize open page policy, and bank parallelism by reordering commands to take advantage of open rows. To **avoid starvation** there are CPU clock cycle thresholds for CPU requests of each type (fetch instruction, read data, write data), such that after each request is accepted into the queue, after a certain amount of clock cycles the request becomes a high priority request. The default thresholds are set to 500 CPU clock cycles for fetches, 1000 CPU clock cycles for reads, and 2000 CPU clock cycles for writes.These thresholds can be adjusted in the command like with -fch, -rd, -wr flags. If there are no requests over their thresholds, priority then goes to fetches to open rows, followed by reads to open rows, followed by writes to open rows. (-opt)
3. Strict build. Strict in order scheduling. As in the default build, the memory controller will exploit bank parallelism, but rather it will only service activate and precharge commands out of order, if efficient. Reads and writes will only execute in the order they came from the core. (-strict)

All three implementations are available at runtime via command line arguments.

Testing:
In order to ensure a properly functioning simulator, the default version was tested methodically and exhaustively. Testing began by ensuring that there were no syntax errors in our output, and

no errors in our parser. Before testing that our simulator was adhering to timing constraints, we had to first highlight all the various cases that the simulator could encounter, and what their timing constraints were. To do so, see the following outline of decision diagrams, constraints, and test cases:

Timing Constraints Between DRAM Commands

1st command	2nd command	To	Timing Constraint Internal to these two instructions (DRAM)	Notes (Page Reference to: DDR4 Datasheet)
Precharge	Precharge	Same bank group, same bank	tRC = 76 = tRAS + tRP (how quickly after activating that you can issue and complete a precharge command)	
		Same bank group, different bank	0 constraint (1 DRAM cycle) (row must have satisfied tRAS)	
		Different bank group	0 constraint (1 DRAM cycle) (row must have satisfied tRAS)	
Precharge	Act	Same bank group, same bank	tRP = 24	
		Same bank group, different bank	0 constraint (1 DRAM cycle) (row must have satisfied tRAS)	
		Different bank group	0 constraint (1 DRAM cycle) (row must have satisfied tRAS)	
Precharge	Read	Same bank group, same bank	NOT POSSIBLE	
		Same bank group, different bank	0 constraint (1 DRAM cycle) (row must have satisfied tRAS)	
		Different bank group	0 constraint (1 DRAM cycle) (row must have satisfied tRAS)	
Precharge	Write	Same bank group, same bank	NOT POSSIBLE	
		Same bank group, different bank	0 constraint (1 DRAM cycle) (row must have satisfied tRAS)	
		Different bank group	0 constraint (1 DRAM cycle) (row must have satisfied tRAS)	
Act	Precharge	Same bank group, same bank	tRAS = 52	
		Same bank group, different bank	0 constraint (1 DRAM cycles)	
		Different bank group	0 constraint (1 DRAM cycles)	
Act	Act	Same bank group, same bank	NOT POSSIBLE	Page 139
		Same bank group, different bank	tRRD_L = 6	
		Different bank group	tRRD_S = 4	

Act	Read	Same bank group, same bank	tRCD = 24	
		Same bank group, different bank	0 constraint (1 DRAM cycles)	
		Different bank group	0 constraint (1 DRAM cycles)	
Act	Write	Same bank group, same bank	tRCD = 24	
		Same bank group, different bank	0 constraint (1 DRAM cycles)	
		Different bank group	0 constraint (1 DRAM cycles)	
Read	Precharge	Same bank group, same bank	RTP = 12, must also satisfy time between Act and Precharge (tRAS)	Page 209
		Same bank group, different bank	RTP = 12	
		Different bank group	RTP = 12	
Read	Act	Same bank group, same bank	NOT POSSIBLE. Read will either lead into another read or write, or precharge	
		Same bank group, different bank	0 constraint (1 DRAM cycles)	
		Different bank group	0 constraint (1 DRAM cycles)	
Read	Read	Same bank group, same bank	tCCD_L = 8	Page 198
		Same bank group, different bank	tCCD_L = 8	
		Different bank group	tCCD_S = 4	
Read	Write	Same bank group, same bank	tCAS + tBURST - CWL = 8	Page 203
		Same bank group, different bank	tCCD_L = 8	
		Different bank group	tCCD_S + tWTR = 8	We invented tWTR to prevent bus conflict
Write	Precharge	Same bank group, same bank	tCWL + tBURST + tWR = 20 + 4 + 20 = 44	Page 237
		Same bank group, different bank	0 constraint (1 DRAM cycles)	
		Different bank group	0 constraint (1 DRAM cycles)	

Write	Act	Same bank group, same bank	NOT POSSIBLE	
		Same bank group, different bank	0 constraint (1 DRAM cycles)	
		Different bank group	0 constraint (1 DRAM cycles)	
Write	Read	Same bank group, same bank	tCWL + tBURST + tWTR_L = 20 + 4 + 12 = 36	Page 233
		Same bank group, different bank	tCWL + tBURST + tWTR_L = 20 + 4 + 4 = 36	
		Different bank group	tCWL + tBURST + tWTR_S = 20 + 4 + 4 = 28	
Write	Write	Same bank group, same bank	tCCD_L = 8	Page 227
		Same bank group, different bank	tCCD_L = 8	
		Different bank group	tCCD_S = 4	

Table(1) - Timing constraints for all cases of DRAM command orderings

Upon determining the necessary timing constraints we must abide by, we wanted to run many tests that verified that no constraints were being violated.

Initially, we tested all of the base cases that we speculated could have the potential to either break the program completely, or reveal timing violations or other violations our program was making. Once we were able to hand verify that our basic implementation was producing correct output results, the testing process was automated to efficiently ensure that as we developed more complex processes for our advanced implementations, that we were not making violations in the process. The automation was able to compare current tests outputs with the previously verified outputs to verify that continually adhered to all constraints.

After sufficiently testing the basic cases targeting specific timing constraints, the testing process was taken a bit further. We developed our own utility program to quickly generate large traces to test in the simulator. This methodology provided a means for streamlining some of the isolated cases before into test files that would be closer to what a hardware implementation of a memory controller would be dealing with.

Parsing Tests

Potential Problem	**Should We Test?**	**Test Trace Filename**	**Solution**	**Test Case**
Combination of Spaces/Tabs between the time, command, and address	Yes	extraWhitespace_test.txt	Using fscanf to gather the arguments and ignore the whitespace.	Have input trace use different spacing and tabs.
Empty Lines in Input Files	No		The design document describes very specific input formatting and describes the fields and that they are separated by one or more spaces or tabs. It does not indicate that the user can include empty lines in their input, so we assume that this should generate an error.	
Comments in Trace File?	No		The design document describes very specific input formatting and describes the fields and that they are separated by one or more spaces or tabs. It does not indicate that the user can include comment lines or comment out traces, so we assume that this should generate an error.	
Out of range addresses? This could be an issue if the row is too large. The address is given in hexadecimal, we can pull out all of the organization bits, but the row could be too large where it would reference an invalid address. We can either ignore this and mask the extra bits, or end the simulation and provide an error.	Yes	rowOutOfRange_test.txt	Add a check to the parser where it looks to see if the upper address bits that would be assigned to the 15 bits of the row address are out of range (larger than 0x7FFF or 32767) as this would be larger than the number of rows in the memory. Exits with an	A simple test case has an address of 0x11FFFFFFF which is larger than the possible address.

			error.	
Negative time inputs	No		Dr. Faust said that the trace file will be monotonically increasing and never negative, so we don't need to test for this condition.	
Time overflow	Yes	timeOverflow_test.txt	Dr. Faust said that we can decide what occurs if the elapsed CPU time overflows the size we have given it which is an unsigned long long int. We have decided that this should generate an error if the time overflows.	A simple test case tries to read at time 18446744073709551616 which is larger than an unsigned long long integer can hold. This should trigger an end simulation message.

Table(2) - Testing for errors that could occur in parsing

Runtime Tests

Potential Problem	**Should We Test?**	**Test Trace Filename**	**Solution**	**Test Case**
Multiple commands received at the same CPU clock cycle	Yes	sameClockCycle_test.txt	Per Dr. Faust, we decided to have only one item be added to the queue per CPU clock cycle. So any duplicate times will be added on the following clock cycle and not before.	Multiple requests on the same clock cycle twice. Fill the queue, should continue operating and only insert these extra requests once a space has opened in the queue.
Access successive bank groups and banks in the same row and column	Yes	allBanksBankGroups_test.txt	Timing should have all of the possible bank groups and banks accessed sequentially and should have no precharge commands.	16 requests all on the same byte select, row, column, with only bank group and bank changing through all possible combinations.
Read	Yes	readSingle_test.txt	Basic read, checking timing. Should be Activate, Read.	Single read to confirm basic timing. Confirms tRCD = 24.
Write	Yes	writeSingle_test.txt	Basic write, checking timing. Should be Activate, Write.	Single write to confirm basic timing. Confirms tRCD = 24.
Instruction Fetch	Yes	ifetchSingle_test.txt	Basic instruction fetch, should be the same as the Read.	Single instruction fetch to confirm basic timing. Confirms tRCD = 24.
Read from an Open Page Later	Yes	readOpenRowLater_test.txt	Basic read and then another read later from that open row	Two reads, one that happens much later from the open page. Tests open page policy.
Write from an Open Page Later	Yes	writeOpenRowLater_test.txt	Basic write and then another write later from	Two writes, one that happens much later

			that open row	from the open page. Tests open page policy.
Two successive Reads from the same Bank and BG	Yes	readSameBGB_test.txt	Read from the same bank group's bank twice.	Should confirm that delay is happening correctly when accessing the same row twice. Tests tCCD_L = 8.
Two successive Writes from the same Bank and BG	Yes	writeSameBGB_test.txt	Write from the same bank group's bank twice.	Should confirm that delay is happening correctly when accessing the same row twice. Tests tCCD_L = 8.
Two successive iFetches from the same Bank and BG	No			Same as reading
Two successive Reads from different Bank Groups	Yes	readDiffBG_test.txt	Read from two different bank groups	Should be act, act, read, read. Tests tRRD_S = 4 and tCCD_S = 4.
Two successive Writes from different Bank Groups	Yes	writeDiffBG_test.txt	Write from two different bank groups	Should be act, act, write, write Tests tRRD_S = 4 and tCCD_S = 4.
Two successive iFetches from different Bank Groups	No			Same as reading
Two successive Reads from the same BG different bank	Yes	readSameBGDiffB_test.txt	Read from the same bank group two different banks.	Should confirm that delay is happening correctly when trying to activate the same bank group again. Tests tRRD_L = 6 and tCCD_L = 8.

Two successive Writes from the same BG different bank	Yes	writeSameBGDiffB_test.txt	Write from the same bank group two different banks.	Should confirm that delay is happening correctly when trying to activate the same bank group again. Tests tRRD_L = 6 and tCCD_L = 8.
Two successive iFetches from the same BG different bank	No			Same as reading
Three successive Reads from the same Bank and BG but different rows	Yes	readThreeDiffRows_test.txt	Read from the same bank and bank group 3 times but different rows	Tests precharge to precharge delays. Tests tRAS = 52, tRP = 24 and tRCD = 24 and tRC = tRAS + tRP = 76
Three successive Writes from the same Bank and BG but different rows	Yes	writeThreeDiffRows_test.txt	Write from the same bank and bank group 3 times but different rows	Tests precharge to precharge delays. Tests tCWD(CWL) + tBurst + tWR = 20 + 4 + 20 = 44 which is the minimum time after a write that a precharge can be issued which is longer than tRAS. Also tests tRP = 24 and tRCD = 24.
Three successive iFetches from the same Bank and BG but different rows	No			Same as reading
Three successive reads. One, then a pause, second to same row, bank, bank group, then a third to same bank, bank group, but a different row.	Yes	readTwoLongWaitOneRTP_test.txt	Check to ensure tRTP is satisfied	Tests tRTP = 12 and tRP = 24. tRTP is a much shorter time period that needs to be satisfied than tRAS, but tRAS has already been satisfied because

				there is a long time before the second read.
Three successive writes. One, then a pause, second to same row, bank, bank group, then a third to same bank, bank group, but a different row.	Yes	writeTwoLongWaitOneRTP_test.txt	Check to ensure tRTP is satisfied	Tests tRTP = 12 and tRP = 24. tRTP is a much shorter time period that needs to be satisfied than tRAS, but tRAS has already been satisfied because there is a long time before the second write.
Read then write on same row	Yes	readWriteSameRow_test.txt	Check to ensure read then write timing.	Tests read to write latency differences between CL and WL where WL is smaller so there could be bus contention. Extra padding is done to ensure that tCCD_L is long enough to allow the read to complete before the write starts using the bus.
Read then write on different rows	Yes	readWriteDiffBGB_test.txt	Check to ensure read then write timing and that bus contention doesn't occur.	Tests read to write latency differences between CL and WL where WL is smaller so there could be bus contention. Extra padding is done to ensure that tCCD_L is long enough to allow the read to complete before the write starts using the bus.
Read then write then read on same row	Yes	readWriteReadSameRow_test.txt	Check to ensure read, write, read timing and that	Tests are the same as above and tWTR_L

			tWTR is satisfied.	+ tBurst + tCWD = 12 + 4 + 20 = 36. Can't read again on the same row until the write is finished and the write to read delay is satisfied.
Read then write then read on same row	Yes	readWriteReadDiffBGB_test.txt	Check to ensure read, write, read timing across banks and that tWTR is satisfied.	Tests are the same as above and tWTR_S + tBurst + tCWD = 4 + 4 + 20 = 28. Can't read again on the same row until the write is finished and the write to read delay is satisfied.
Read then write then read across all banks and bank groups.	Yes	readWriteReadAllBanksBankGroups_test.txt	Check to ensure read write read timing across banks all works.	Tests above read write read conditions over all banks and bank groups for validation

Table(3) - Runtime tests

Test Results:

Once the testing was validated, we wanted to check the statistics of our main simulator versus our optimized simulator. For the purposes of this report, three simple cases will be highlighted. We looked at performance over a randomly generated test file, a test file of consecutive misses (i.e. hitting different rows consecutively in the same bank), and a test file that yields all hits, cycling through bank groups repeatedly. We were particularly interested in how the performance improved with the ideal hit case, because it alluded to how successful our memory controller would be when a real life CPU was consistently accessing sequential memory.

The following tests measure performance of the memory controller simulator optimal mode versus performance in default mode.

```
Miss Test File:  Basic Run Statistics
--------------------STATISTICS--------------------
--IFETCHES:
Min:470
Max:2855
Average:2599.926
Median:2599.0
--------------------------------------------------
--READS:
Min:1106
Max:2823
Average:2600.457
Median:2599.0
--------------------------------------------------
--WRITES:
Min:96
Max:2847
Average:2592.828
Median:2591.0
--------------------------------------------------
----------------------TOTALS----------------------
Min:96
Max:2855
Average:2597.722
Median:2591.0
--------------------------------------------------
```

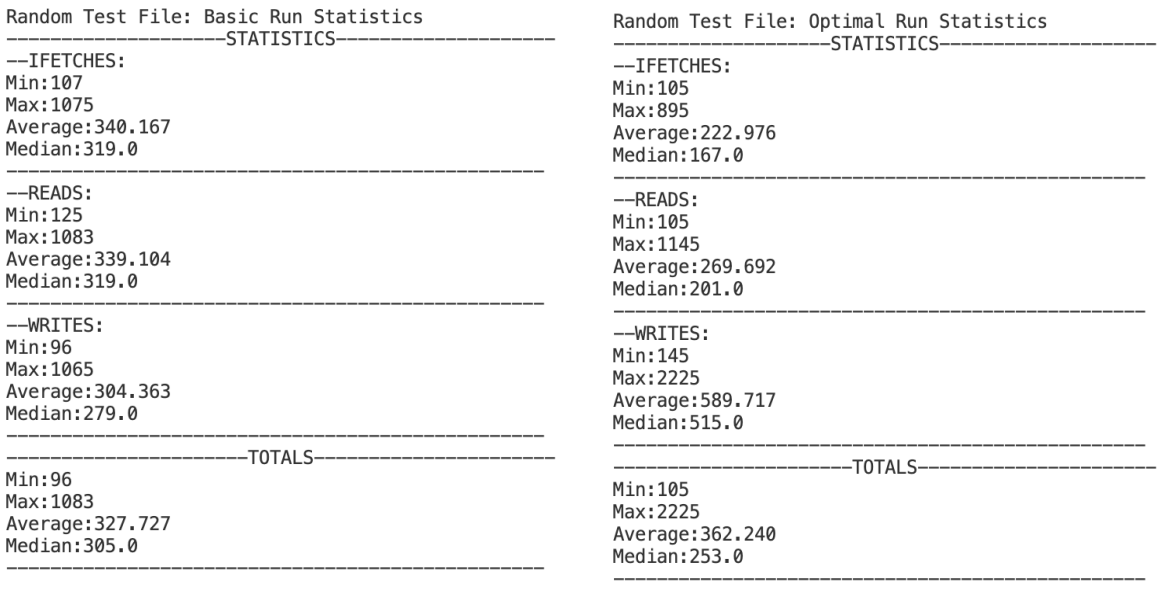
```
Miss Test File: Optimal Run Statistics
--------------------STATISTICS--------------------
--IFETCHES:
Min:286
Max:2623
Average:1977.410
Median:1983.0
--------------------------------------------------
--READS:
Min:1498
Max:3151
Average:2447.361
Median:2447.0
--------------------------------------------------
--WRITES:
Min:96
Max:4119
Average:3381.689
Median:3383.0
--------------------------------------------------
----------------------TOTALS----------------------
Min:96
Max:4119
Average:2597.706
Median:2447.0
--------------------------------------------------
```

Figure(1) - Measuring memory controller performance over a completely randomly generated input file. Default/basic mode vs. optimal mode performance.

```
Hit Test File: Basic Run Statistics
--------------------STATISTICS--------------------
--IFETCHES:
Min:166
Max:407
Average:326.515
Median:351.0
--------------------------------------------------
--READS:
Min:239
Max:407
Average:328.312
Median:341.0
--------------------------------------------------
--WRITES:
Min:96
Max:407
Average:289.611
Median:295.0
--------------------------------------------------
----------------------TOTALS----------------------
Min:96
Max:407
Average:313.931
Median:295.0
--------------------------------------------------
```

```
Hit Test File: Optimal Run Statistics
--------------------STATISTICS--------------------
--IFETCHES:
Min:57
Max:153
Average:100.818
Median:103.0
--------------------------------------------------
--READS:
Min:57
Max:178
Average:105.812
Median:111.0
--------------------------------------------------
--WRITES:
Min:169
Max:569
Average:397.611
Median:395.0
--------------------------------------------------
----------------------TOTALS----------------------
Min:57
Max:569
Average:208.188
Median:127.0
--------------------------------------------------
```

Figure(2) - Measuring memory controller performance over consecutive hits to open rows. As seen, the performance has increased dramatically. The average time a request is in the queue has been reduced by 33%. Default/basic mode vs. optimal mode performance.



Figure(3) - Measuring the memory controller performance given a file of consecutive misses (requests to a different row in the bank than the one that is open). Default/basic mode vs. optimal mode performance.

In this case performance given the optimal algorithm slightly worsened. However, this observed performance is trivial given that a case such as this is virtually impossible. We wanted to test the worst possible case in order to understand if the algorithm would drastically reduce performance beyond usability; the optimal algorithm passed this test, in that given an impossible case, reasonable performance was still given.

Conclusion:
At a high level, there are many arguments for the best and most efficient ways to handle data. That is because we rely on the assumption that spatial locality and temporal locality will be of the essence when using a computer. That being said, there are many policies that a memory controller can implement in order to optimize performance. The way in which we can theorize and test an algorithm, without immediately delving time and resources into chip manufacturing, is to simulate a design in software.

In our first version, we implemented a default edition of a memory controller exploiting an open-page policy and bank parallelism that utilized in-order scheduling but allowed the most efficient and ready command in the queue to be initiated to the DIMM. In optimization mode, we made the algorithm more efficient; prioritizing instructions over reads, and reads over writes. Further, priority was given to fetching, reading, and writing from/to open rows. As shown in the test results, this optimization made the CPU performance more efficient given a file that demonstrated spatial locality. Finally, a strict in order implementation was developed. The strict build is similar to the default build, but it will only service activate and precharge commands out

of order, if efficient. Reads and write commands will always be issued in the order in which they were received from the CPU core. This simulation is modular, and designed so that further algorithms down the line could be added without a need for overhauling the entire code. The project demonstrates that simulation can be used to test hypothetical memory controller accuracy and performance, all before devoting time, energy, and resources to hardware development.

References:
Micron DDR4 SDRAM Datasheet