ECE585 Final Project

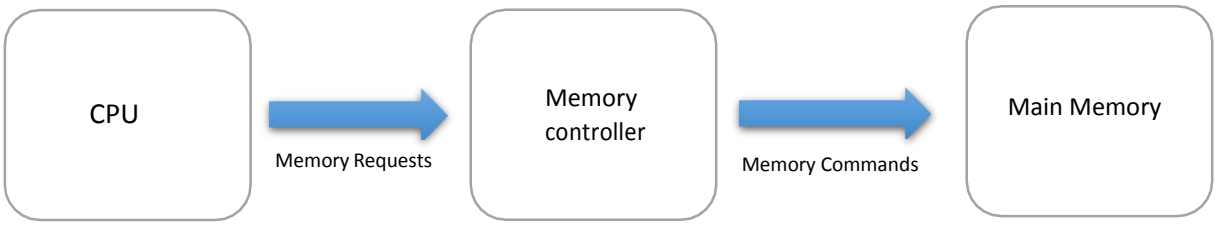
Simulation of a DDR3 DRAM Memory Controller

Surendra Maddula

Introduction:

Main memory is an essential part in a modern computer. Now a days DDR3 and DDR4 memory modules are generally used. To achieve higher bandwidths and to relieve the CPU from stress, we use memory controller for accessing main memory. This report gives a brief description about a DDR3 based memory controller and its implementation. This controller uses open-page policy with no access scheduling. Its implementation and operation are discussed below.

Block diagram:



Open-page policy:

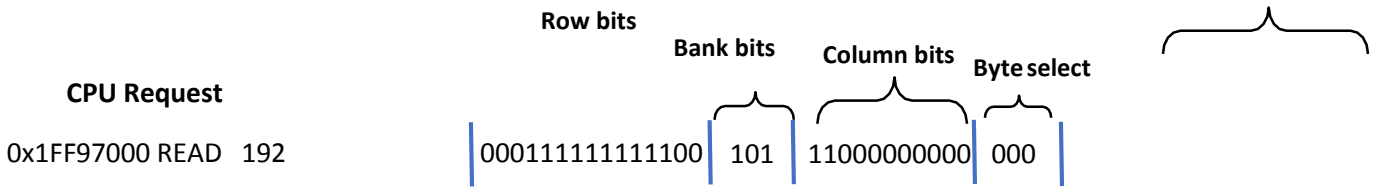
A well designed memory controller which uses an effective strategy can really give good performance and save lot of power. Inside a DRAM chip, there will be multiple Banks, (8 Banks in DDR3) and every bank contains thousands of rows and columns. In DRAM terminology, a row can be called as a 'page'. Here there is a fundamental limitation, i.e.; inside a bank, only one row can be active at any given time. Whenever CPU makes a request, memory controller turns that into a command and schedules it to DRAM. In this policy when CPU accesses a row in DRAM, it won't be closed (pre charged) immediately. Instead, we will proceed with the next command. If the command is for the same row then it is termed as a Hit, and we can save some time on accessing the data (As the row is already active). If that command is for another row in that same bank, then we need to pre charge the currently opened row and after that we need to activate the required row and should perform the operation. This is known as a Miss and wastes a whole lot of time.

Project Implementation:

This project simulates a DDR3 Ram Memory controller, which uses open page policy. A general outline on open page policy was given in the previous section. Our memory controller accepts requests from CPU and turns them into memory commands and supplies it to memory. Here, we assume there is only one requestor, and the commands will be taken from a text file. Output of memory controller will be another text file with all the timing parameters. This memory controller maintains a 16 element queue, in which requests are placed. Memory controller doesn't take any further requests when the queue is full. Controller removes an element in the que when it finishes operating on it.

Before executing any command, we first should decode the given command into byte select, column, bank, Row respectively.

Here, we have 32k rows, 8 banks, 2k columns and we use 3-bits for 'byte select'. So, we require 15bits for Row selection, 3 bits for bank selection, and 11 bits for column selection. In general, CPU clock will be always higher than DRAM clock. Here in this project, CPU clock is 4x faster than DRAM clock. The below description gives a clear view of the process.



The above request is having an address field, operation description, and the time stamp of the requestor. 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). We treated 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.

The first step in processing a request is dividing it into respective Row, bank, column, byte select and the type of request to be performed (e.g.: read/write/I fetch). After knowing the request to be performed, we can immediately jump ahead to issue the commands. But, issuing commands isn't a simple task. We need to consider several timing parameters before issuing a command.

This project implements open-page policy with no access scheduling. According to this policy, when a row in a bank is accessed, it won't be precharged immediately, instead we expect the next request for the currently active row. If the request is made for another row, then we need to precharge the currently active row, only then we can open the required row and process the request.

After having all this data, our program starts the required operation, but here we need to obey few timing constrains in order to get required operation done correctly. The below table lists all those parameters.

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

tRC: Row Cycle Time Determines Bandwidth
– Minimum Lme between successive row accesses
– tRC = tRAS + tRP

tRAS: Row Address Strobe
– Minimum Lme /RAS must be maintained

tRP: Row Pre-Charge Delay
– Minimum time to pre-charge so another /RAS can begin

tCAC: Column Access Time
– Delay from falling /CAS to valid data out

tRCD: Row Command Delay (RAS/CAS Delay)
– Minimum time between a row command and a column command

tRAC: Random Access Delay Determines Latency
– Minimum time from /RAS falling to valid data output
– Quoted as the speed of a DRAM
– tRAC = tRCD + tCAC

After completing the operation, program (memory controller) produces an output text file with DRAM commands in it. It looks like as below described.

<CPU clock ticks><DRAM command>

Where <CPU 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>

As a part of testing, various test inputs were given to the code, and the results were obtained. Those

results are as follows

Assumptions:

We assume that all banks are in precharged state initially.

Test cases & Results:

1) Consecutive reads for the same bank same row

INPUT:

0x2300D501 READ 1

0x2300D501 READ 2

OUTPUT:

4	ACT	3	0x1180
60	RD	3	0x2A0
132	RD	3	0x2A0

As we assume that, all banks are precharged, we started command sequence with ACT command, and after TRCD, we issued Read command, as the next command is for the same row, we can issue that right after completing the first request (Tccd is satisfied already).

2) Consecutive Write and read for the same bank and same row

INPUT:

0x2300D501 WRITE 20

0x2300D501 READ 25

OUTPUT:

20	ACT	3	0x1180
76	WR	3	0x2A0
164	RD	3	0x2A0

Here we first start with activate command, after TRCD, we issue WR command, after completing the write burst, we will wait for Twtr period then we will issue read command.

3) Consecutive read and write for the same bank and same row

INPUT:

0x2300D501 READ 20

0x2300D501 WRITE 25

OUTPUT:

20	ACT	3	0x1180
76	RD	3	0x2A0
148	WR	3	0x2A0

For this request, we will start with ACT command, then after TRCD we will issue RD command. Before issuing the Write command, we need to meet (Tcl+Tburst+2TCLK-TWL) timing value. For the given timing values, we can meet the requirement just after completing the RD command. So, we will issue this command right after RD+burst.

3) Now let us think that we have 4 requests this time, 4th request being the same first request.

INPUT:

0x2300D501 WRITE 20

0x2300D501 READ 25

0x23002501 READ 30

0x2300D501 WRITE 35

OUTPUT:

20	ACT	3	0x1180

76	WR	3	0x2A0
164	RD	3	0x2A0
236	ACT	0	0x1180
292	RD	0	0x4A0
364	WR	3	0x2A0

In this particular case, we tested, the prime motto of open page policy, i.e. keeping a row open for the consecutive requests. Because of this policy here we have saved clock cycles required for executive activate command for the 4th request.

4) 4th request to same bank as 1st request but for the different Row.

INPUT:

0x2300D501 WRITE 20

0x2300D501 READ 25

0x23002501 READ 30

0x2900D501 WRITE 35

OUTPUT:

20	ACT	3	0x1180
76	WR	3	0x2A0
164	RD	3	0x2A0
236	ACT	0	0x1180
292	RD	0	0x4A0
364	PRE	3	
420	ACT	3	0x1480
476	WR	3	0x2A0

This is same as the above case, instead this time our policy didn't help us because the 4th request was made for an unopened row in the bank

5) Same bank, different row first write then read.

INPUT:

0x2300D501 WRITE 20

0x2600D501 READ 25

0x23002501 READ 30

0x2900D501 WRITE 35

OUTPUT:

20	ACT	3	0x1180
76	WR	3	0x2A0
196	PRE	3	
252	ACT	3	0x1300
308	RD	3	0x2A0
380	ACT	0	0x1180
436	RD	0	0x4A0
508	PRE	3	
564	ACT	3	0x1480
620	WR	3	0x2A0

Here, the first and second requests were given for the same bank but different rows. So our policy didn't help here, After serving for the first request, we need to close(Precharge) that opened row and then we need to open another row for executing 2nd command.

6) Requests for the same bank but different row, read followed by write

INPUT:

0x2300D501 READ 20
0x2600D501 WRITE 25
0x23002501 READ 30
0x2900D501 WRITE 35

OUTPUT:

20	ACT	3	0x1180
76	RD	3	0x2A0
164	PRE	3	
220	ACT	3	0x1300
276	WR	3	0x2A0
332	ACT	0	0x1180
388	RD	0	0x4A0
460	PRE	3	
516	ACT	3	0x1480
572	WR	3	0x2A0

These request patterns were same as above ones except, the first two requests were changed. Remember that requests accessing different rows in the same bank requires a precharge command in between them.

7) Next request time < the previous request time
INPUT:
0x2300D501 READ 20
0x2600D501 WRITE 19
OUTPUT:

20	5	ACT	3	0x1180
76	19	RD	3	0x1180 0x2a0

This is an obvious case, as we can't travel into past, these kind of request timings considered illegal and we will abort the execution after displaying the following error.

(In Console) WARNING!
New request time cannot be less than previous request time, aborting.....

8) Next request time = previous request time
INPUT:
0x2300D501 READ 20
0x2600D501 WRITE 20
OUTPUT:

20	ACT	3	0x1180
76	RD	3	0x2A0
164	PRE	3	
220	ACT	3	0x1300
276	WR	3	0x2A0

In multicore systems, this kind of request pattern may be found, but our controller assumes Single requestor in this case. So, we print a warning during this condition and we will put the request into Que. Warning message will be like below.

(In Console) WARNING!
New request time is equal to previous request, can be used in multi core processors

9) Same bank, different rows – read, write, and read.
INPUT:
0x2200D501 READ 20
0x2300D501 WRITE 25
0x2400D501 READ 30
OUTPUT:

20	ACT	3	0x1100
76	RD	3	0x2A0
164	PRE	3	
220	ACT	3	0x1180
276	WR	3	0x2A0

396	PRE	3	
452	ACT	3	0x1200
508	RD	3	0x2A0

This example involves requests for the same bank but different row each time, So we need to precharge for every memory request in this case.

10) Different banks consecutive read, write, and read

INPUT:

0x2200D501 READ 100
0x23002501 WRITE 200
0x2400A501 READ 300

OUTPUT:

100	ACT	3	0x1100
156	RD	3	0x2A0
228	ACT	0	0x1180
284	WR	0	0x4A0
340	ACT	2	0x1200
396	RD	2	0x4A0

As DDR follows Bank architecture, this kind of input pattern will open a row in every bank, and we don't need to do any precharge because, there are no currently active rows in those opened banks.

11) Que is full and consists of 24 requests

INPUT:

0x2000D5C1 IFETCH 50
0x1FF96FC0 WRITE 160
0x2000D600 IFETCH 190
0x1FF97000 READ 200
0x123AA340 IFETCH 250
0x1FF96FC0 WRITE 300
0x1290D600 IFETCH 320
0xA1234000 READ 330
0x200D8340 READ 350
0x2000D234 IFETCH 360
0x1FF96ABC WRITE 370
0x2000D6CD IFETCH 375
0x1FF970EF READ 400
0x123AA3AC IFETCH 420
0x1FF96F0C WRITE 430
0x1290D123 IFETCH 440
0xA1234111 READ 450
0x200D8888 READ 480
0x2000D234 IFETCH 490
0x1FF96ABC WRITE 500
0x2000D6CD IFETCH 510

0x1FF970EF READ 520
0x123AA2DF IFETCH 530
0x1FF96ADA WRITE 540

OUTPUT:

52	ACT	3	0x1000
108	RD	3	0x2B8
180	ACT	5	0xFFC
236	WR	5	0x5F8
292	RD	3	0x2C0
364	RD	5	0x600
436	ACT	2	0x91C
492	RD	2	0x468
564	WR	5	0x5F8
620	PRE	3	
676	ACT	3	0x948
732	RD	3	0x2C0
804	ACT	7	0x3FFE
860	RD	7	0x7FF
932	ACT	6	0x1006
988	RD	6	0x68
1060	PRE	3	
1116	ACT	3	0x1000
1172	RD	3	0x246
1244	WR	5	0x557
1300	RD	3	0x2D9
1372	RD	5	0x61D
1444	RD	2	0x475
1516	WR	5	0x5E1
1572	PRE	3	

1628	ACT	3	0x948
1684	RD	3	0x224
1756	RD	7	0x7FF
1828	RD	6	0x111
1900	PRE	3	
1956	ACT	3	0x1000
2012	RD	3	0x246
2084	WR	5	0x557
2140	RD	3	0x2D9
2212	RD	5	0x61D
2284	RD	2	0x45B
2356	WR	5	0x55B

This input request pattern will test the Queue. Here we have given more than 16 requests, So that we can observe how our que can handle these kind of sequence. Our Queue took 16 requests at first, then after removing a processed request, it inserted another request. The same process was followed until there are no pending requests.