MICROPROCESSOR SYSTEM DESIGN

ECE 585

SIMULATION OF MEMORY CONTROLLER

PROJECT REPORT

INSTRUCTOR:

Prof. Mark Faust

Group No: 10

Team members: Bhoomika Mohan ([bhoomika@pdx.edu](mailto:bhoomika@pdx.edu))

Mansi Dhanrajani ([mansi2@pdx.edu](mailto:mansi2@pdx.edu))

Preksha Gandhi ([preksha@pdx.edu](mailto:preksha@pdx.edu))

Sanchita Leekha([sleekha@pdx.edu](mailto:sleekha@pdx.edu))

**Design Specification and Implementation:**

We have implemented a memory controller using SystemVerilog. Our code deals with in order, open page policy with no bank parallelism. Using $value$plusargs we are enabling or disabling the debug mode. If the debug is enabled, we are displaying the request being inserted in the queue and the request is being popped out of the queue. If the queue is empty, we are incrementing simulation time by 1, or if the queue is not empty the simulation time will be incremented to the time value of the next request. We are also checking if the request time of the next instruction is greater than or equal to the time of the previous request. We are also checking whether the instruction is read, write, or instruction fetch. If there is an operation other than that, it is invalid. If the queue is not empty and there are no pending requests, the code checks if the queue of 16 requests is full. If it is not, and there is no pending request, the program reads from the file and puts the next pending request into the queue. If there is a pending request and the queue is empty, the clock will move to the request time of the pending request. A request will be triggered for every DIMM tick. For the implementation of the open page policy, if the request is made to the same bank, bank group, and same row, there is no need for precharge and directly read/write command is issued. Each bank maintains the structure of the bank state to check if it has been previously accessed or not. Every subsequent access to the same bank will result in a precharge command unless it is to the same row which was accessed most recently.

Address mapping:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Rows (15 bits) | Higher Column  Bits (8 bits) | Banks (2 bits) | Bank Groups (2 bits) | Lower Column  Bits (3 bits) | Byte Select  (3 bits) |

In DIMM 8GB PC4 25600 clock rate is

25600/8=3200

3200/2=1600

Therefore, 1600 MHz

Page size is 2KB

The internal access width is 8, therefore the number of columns is 2^11 = 2K.

Number of column bits = 11

There are 8 chips of x8 device (as per the description) as the width of DIMM module is 64 bits.

Number of Banks = 4 therefore 2 address bits for banks.

Number of Bank Groups = 4 therefore 2 address bits for bank groups.

In our implementation, we have used the byte select bits as unused bits at the least significant 3 bits as the address is 8 bytes aligned. So we can ignore the last 3 bits. Next, we have low column bits, as we will give priority to the request (rd/wr/instruction fetch) to a different column in the same row so that fewer precharges and activates have to be done. The column bits are divided into higher and lower order bits as lower columns are accessed more frequently. Next, a bank group is put in our implementation as we prioritize access to different bank groups before giving access to the same bank. Access to different bank groups leads to a penalty of tCCD\_S and tRRD\_S. After the bank group, the bank is placed as in case of bank parallelism simultaneously another bank can be used for accessing a row. This precharge and access is preferable instead of doing a precharge to the current bank being accessed. Next higher order column bits are placed. The last block is for rows as rows incur the highest timing penalties.

Testing:

We have covered the following cases for testing our program.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Bank** | **BankG** | **Row** | **Col** | **Req1** | **Req2** | **Type** |
| **Same** | **Same** | **Same** | **Same** | **0x01FF97000** | **0x01FF97001** | **R to R** |
| **Same** | **Same** | **Diff** | **-** | **0x011115D51** | **0x01BF6A958** | **R to R** |
| **Same** | **Same** | **Same** | **Diff** | **0x020090981** | **0x02008A582** | **R to R** |
| **Diff** | **Same** | **-** | **-** | **0x56710F138** | **0xFFF10332A** | **R to R** |
| **Diff** | **Diff** | **-** | **-** | **0x1FFFB110A** | **0X1AF1102D8** | **R to R** |
| **Diff** | **Diff** | **-** | **-** | **0x1FFFB110A** | **0X1AF1102D8** | **W to R** |
| **Same** | **Same** | **-** | **-** | **0x011115D51** | **0x01BF6A958** | **W to R** |
| **Diff** | **Same** | **-** | **-** | **0x56710F138** | **0xFFF10332A** | **W to R** |
| **Same** | **Same** | **Same** | **Same** | **0x01FF97000** | **0x01FF97001** | **W to W** |
| **Same** | **Same** | **Diff** | **-** | **0x01FF97000** | **0x01FF97001** | **W to W** |
| **Same** | **Same** | **Same** | **Diff** | **0x020090981** | **0x02008A582** | **W to W** |
| **Diff** | **Diff** | **-** | **-** | **0x1FFFB110A** | **0X1AF1102D8** | **W to W** |
| **Diff** | **Same** | **-** | **-** | **0x56710F138** | **0xFFF10332A** | **W to W** |