DDR2 Controller Design Project
EE577B Fall 2009

Due Date:		Dec 11th 2009
Student Name:	Tommy Zhang	Andras Zambori
Student Number:	1922-9142-34	8961-5913-85

Project Description

This Project is intended to provide the students an opportunity to learn about DDR2 SDRAM devices and their source synchronous double data rate bus interface timing. A Verilog model provided by Denali for 512Mb (32Mb x16 four bank) DDR2 would be used in this project. Students are going to design and implement a DDR2 controller in Verilog HDL and simulate their designs along with Denali's DDR2 model using Cadence NC-Verilog. The DDR2 controller is going to provide a simple FIFO based front-end that would support write and read transactions like scalar, block and atomic to and from the DDR2 SDRAM. Students would refer to the JEDEC DDR2 SDRAM Standard (JESD79-2C) for all timing, bus interface and initialization specifications. The controller would initialize the DDR2 model (chip) with the given parameters like CAS latency and Burst length etc. Normal data transactions would start after a successful completion of the DDR2 initialization sequence.

- DDR2 part to be used for the project: Micron MT47H32M16BN-37E
  http://www.micron.com/products/partdetail?part=MT47H32M16BN-37E
- System clock frequency of DDR2 controller = 500MHz
- DDR2 clock to be run at 250 MHz
- Oklahoma State University's 0.18um standard cell library to be used for synthesis. The information on each standard cell in the library can be found at
  http://avatar.ecen.okstate.edu/projects/scells/tsmc018/indexframe.html

Figure 1 shows a cartoon of DDR2 controller. Students have full freedom to improve the design to achieve higher data bandwidth. The heart of the DDR2 controller core is the logic that would be responsible for the following tasks:

(1) Start initialization sequence logic for the DDR2. When INITDDR input is asserted (captured high at the positive edge of the clock) this logic would initialize the DDR2. It asserts an output signal READY when initialization sequence is completed.

(2) Fetch the command/data queued in the input FIFOs and complete the DDR2 Transaction with correct timing without wasting a single unnecessary cycle.

(3) Before de-queuing a read command from input Cmd/Addr FIFO the FSM must ensure that the output FIFO's would have enough space to accommodate the data received from DRAM in response to a read command.

(4) Match or generate correct return address for the data received from the DRAM.

(5) Refresh logic: This logic would cyclically issue DRAM refresh commands

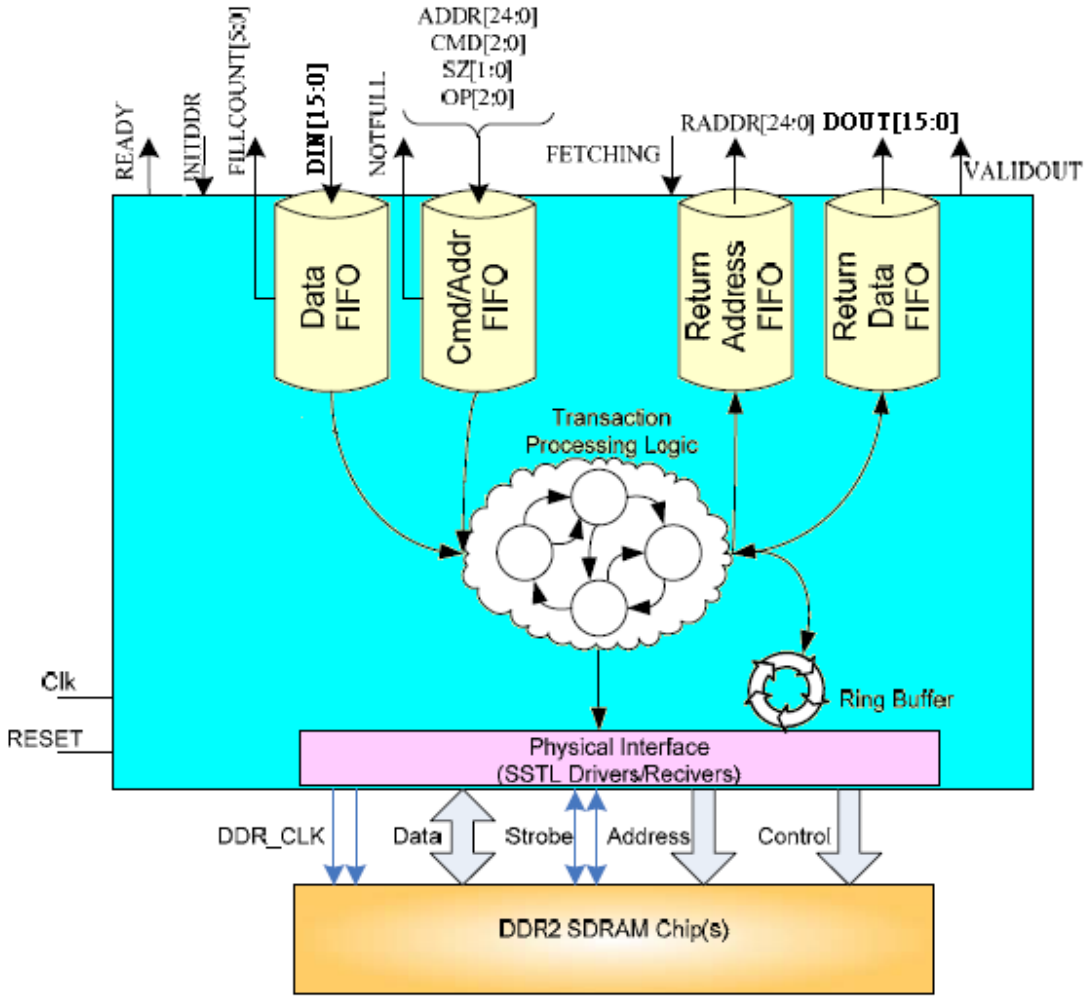


Figure 1. EE577b DDR2 Controller Cartoon

Design Functionality

The design of the DDR2 controller is shown to be working for both the RTL and synthesized netlist. The result of the output dump file matches that of the reference dump file for the provided short and medium test. Since there is not enough disk quota to run the full test, the output of the dump file for the full test is checked up to the point where the test stops running due to the simulation files exceeding the quota. The full test up to that point was found to also match with the provided reference file.

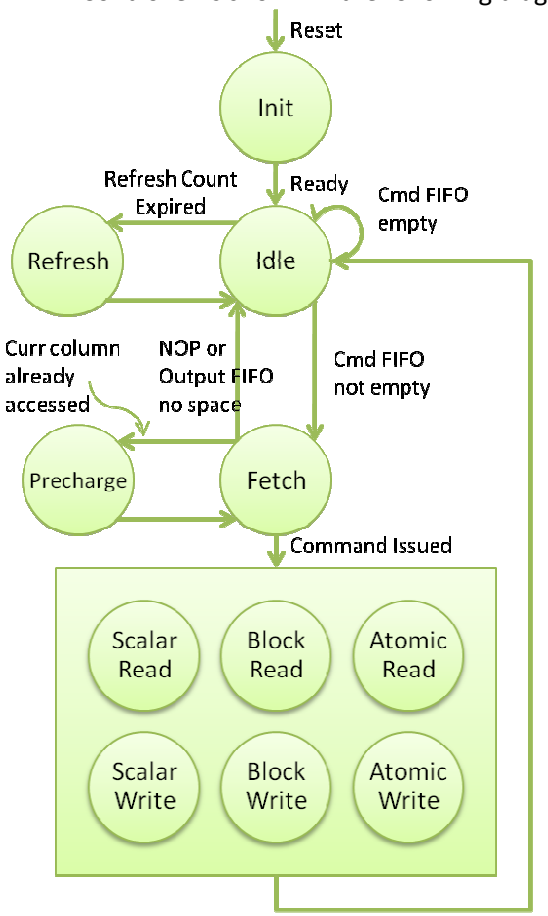
Design Architecture

Initialization Engine

The DDR2 controller initialization engine follows the DDR2 initialization procedure in the DDR2 memory specification sheet to start the DDR2 controller memory clock, ready all banks for operation and configure the various modes registers for the DDR2 operation. The state machine used in the DDR2 controller issues the correct commands in accordance with the timings and delays required in the specification sheet.

Controller State Machine

The state machine used in the DDR2 Controller is shown in the following diagram:



When the reset signal is applied, the controller enters the Init state and runs the DDR2 initialization procedure. The controller registers and counters are initialized and set to known default

values. The DDR2 controller continues to wait in the Init state until the ready signal is asserted by the initialization engine.

Once the ready signal is asserted, the controller transitions to the Idle state and polls the command/address FIFO to see if it is empty. If the command/address FIFO is empty, the controller remains in the Idle state and repeatedly polls the command/address FIFO to check if there is valid data within the FIFO. This process is repeated until the command/address FIFO contains valid data.

On each clock cycle, the refresh counter is incremented to keep track of the time since the last refresh operation was performed to make sure the controller issues the next refresh command before the data in the memory is lost. To prevent other states in the controller from violating the maximum refresh delay, the value at which the refresh counter should begin the refresh operation is the maximum delay between refreshes minus longest operation in the controller, since the controller does not know the command which will be run while it is in the idle state. This ensures that the maximum delay cannot be violated.

Once the command/address FIFO is no longer empty, the controller will transition from the Idle state to the Fetch state. In the Fetch state, the controller reads the command from the command/address FIFO and either returns to the Idle state if the command is a NOP or proceeds to the state to handle to corresponding operation. The transition from the Fetch state to the Idle state in the case of an NOP command has the effect of incurring a 1 clock cycle delay which handles delay for the NOP command. Before proceeding to the operation state for the selected command, if the operation is a read operation, the data output FIFO is checked to make sure that there is enough space to store the output data. If there is not enough space, the controller waits in the Fetch state until there is enough space. Since long wait times in the Fetch state may cause the controller to miss its next refresh cycle, if the controller is waiting in the Fetch state, it must check the refresh counter and initiate the refresh operation if necessary.

Ring Buffer

The ring buffer is necessary due to the fact that data returned from the DDR2 memory during a read access and its strobe signal are asynchronous and not aligned with the DDR2 controller or memory clocks. The ring buffer is needed to bridge the communication between the controller and memory during read operations by temporarily storing data coming from the DDR2 memory and then allowing the DDR2 controller to read the data at the controller clock edge.

Input and Output FIFOs

There are three FIFO's used in the DDR2 controller to handle data communication of the DDR2 controller with external devices. The FIFOs are used as a synchronizing element and also allow the DDR2 controller to buffer commands while the memory is busy with an existing operation. The three FIFO's used in the DDR2 controller are the input data FIFO, the input command/address FIFO and the output data FIFO. All FIFOs are 32 elements deep.

Supported Operations

Once the DDR2 controller transitions from the Fetch state to the Operation state, the following operations can be performed specified by the command issued in the command/address FIFO:
Scalar Read/Write, Block Read/Write and Atomic Read/Write
Each of the commands has its own sub-state machine which proceeds through the sequence of operations and using the timing specified by the DDR2 memory controller specification sheet.

Scalar Read/Write

The scalar read and write operation operates on a single word out of the accessed burst length of four words. The scalar read and write is needed when an operation is to be performed on a number of words less than the size of the burst length. For the scalar read command, the controller stores the very first value read to the output data FIFO. The other words in the burst length are ignored. For the scalar write command, only the first word is written to the memory and then the data mask is applied to prevent subsequent words from being overwritten

Block Read/Write

The block read and write operations allow the DDR2 controller to perform operations on a large set of data successively to achieve high throughput and perform faster than several smaller separate commands on the same amount of data. The block read and write commands have a SZ parameter which determines the number of words to be operated on. Since the SZ always specifies a number of words which is a multiple of the burst length, all accesses in a block operation are useful and no data is ignored or masked. The DDR2 controller uses the SZ parameter to select a sub-state machine to perform the row and column accesses. For larger block sizes, the DDR2 controller must simultaneously issue the row and column access commands and handle the memory DQ and DQS signals when the control for the two signals begin to overlap.

Atomic Read/Write

The atomic operations allow the device issuing the instruction to the DDR2 controller to guarantee data synchronization such that both the access to memory and the following logic operation will occur one after the other without being interrupted by unpredictable system scheduling. Atomic operations are necessary to implement multi-process operating systems as well as provide the basic synchronization mechanisms for locks, barriers and semaphores. For an atomic read operation, the data word provided to the controller is operated on by the data specified by the instruction, which is first read from the DDR2 memory. The result is computed and then stored back to the address specified by the instruction. The old data read from the memory is then sent to the output data FIFO. For an atomic write operation, the supplied data word is combined with the data word stored in the DDR2 memory and the specified operation is performed then written back to the original DDR2 memory address. The atomic operations use the Read-to-Write timings described in the DDR2 memory data sheet, which allows a write operation to follow a read operation to the same row and column without incurring a precharge and activate delay between the two operations. This speeds up the atomic operations considerably.

The atomic operations support the following logic operations:

NOT, AND, OR, XOR, ADD, SUB, SRA, SLA.

Optimizations

Posted CAS

The DDR2 controller uses the posted CAS mode to issue the row and column operations into order to allow the controller to quickly issue the entire operation sequence. The use of the posted CAS scheme increases the efficiency and throughput of the controller by taking advantage of the additive latency. In the configuration used for our project, the column latency is 4 cycles and the additive latency is 3 cycles. Thus if the column command is issued right after the row command, the operation will only incur the delay of the column latency. Issuing the entire sequence of row and column operations frees the controller to handle other operations on other banks. If the posted CAS scheme were not used, then the controller would have to issue the row command first and then wait and do nothing until it is time to

issue to the column operation or keep track of outstanding operations after it issues the row command so that it can issue the column command after the correct timing delay.

Bank Interleaving

Bank interleaving improves the throughput of the DDR2 memory by storing successive address multiples of the burst length in different banks. If an operation goes to the DDR2 controller requesting two burst sizes of data, the DDR2 controller can then issue the first half of the access to one bank and the second half of the access to a second bank. The two accesses can operate independently and in parallel since they are accessing data in different banks, increasing the throughput of the system. If bank interleaving were not used, then in the previous example, both accesses would have to go to the same bank. Since the bank can only handle one operation at a time, the two accesses would have to occur one after the other instead of in parallel.

Manual Precharge

To improve the performance of the DDR2 controller, manual precharge is done when required by an operation rather than auto precharging after each access. The benefits of manual precharge are quicker completion times for each operation since the operation does not have to precharge and close the active row after each access. The cost of this optimization is more logic complexity as well as a large array of registers to store the state of each column to mark if the column has been accessed. Since there are 2^{10} columns and four banks, the size of the array which keeps track of the state of the charge in the columns is 4096 bits. Each time a column is used in a read or write operation, the corresponding bit to that column in the array is changed from a value of 0 to 1 to indicate that the column has been used for this precharge cycle. If a column which has already been used is accessed again, the controller must then precharge all the columns in that bank and activate the row again before the access can proceed. The amount of performance gained by the manual precharge optimization is access dependent. The optimization works best if the accesses are random or sequential. However, if the access tends to reuse the same columns repeated, this optimization can result in lower performance and memory throughput because instead of precharging after each operation and paying the delay upfront, the controller must precharge when it discovers that the next operation requires access to a spent column, which delays the access for that operation.

Last Write Access Caching

To solve the problem of a repeated access to the same address which negates the performance benefits of manual precharge and results in a lower throughput of the DDR2 memory controller, the last write operation to the DDR2 memory is cached. The effect of this caching is if a write access to some address is followed by a read access to the same address, the cache can return the previously written data immediately without having to access the memory to access the data. The benefits to this scheme are twofold: First, the access is much faster since the memory does not have to be accessed. And second, this prevents the DDR2 memory from having to precharge and delay the current operation. The second benefit is especially advantageous if the access patterns are:

write addrA, read addrA
write addrB, read addrB
…
write addrZ, read addrZ

In the above case, manual precharge would perform worse than auto precharge because each pair of operations requires a precharge due to the same address being accessed. No performance is gained from manually controlling precharge even though there are 2^{10}-1 columns which are unused each time the controller precharges. With the last write caching scheme, we prevent the controller from

having to precharge each time since the result is returned without having to access the memory. The size of the write cache used is the word size of the largest operation, which is the block read of size 16. In addition to caching the data, we also need to cache to address so that it can be matched to the address of the access as well as a valid bit for each word to determine if any value has yet been cached. With the given word size of 16 bits: The size of the data cache is 16*16 = 256 bits, the address cache is 25*16 = 400 bits, the valid bits = 16. The total size of the last write cache is 672 bits.

The result of using manual precharging and last write caching is much better throughput from the DDR2 controller, since the two schemes complement each others' weaknesses. Manual precharge improves performance for random and sequential accesses to memory while last write caching protects the manual precharge from having to continuously precharge when the same location in memory is written then read. Manual precharging also complements last write caching, since last write caching by itself is useless when the accesses are random or sequential.

DDR2 Controller Simplifications

Due to lack of memory to perform synthesis on the entire controller with all optimizations, the manual precharge was changed from keeping track of the state of all columns to controlling only banks. This involved changing the controller from keeping track of all columns across all four banks to just keeping track of accesses to the same banks. This was done instead of keeping track of the columns since the size of the array and the logic operations to the 4096 bit array increased the logic complexity of the controller and caused the Synopsys synthesis tool to exceed the memory on the user account and crash at around 50% of the synthesis operation. The banking optimization resulted in better performance for the old mid sized test pattern. But for the current set of test patterns, keeping track of only banks results in less performance gain than just auto-precharging after each access. Thus this feature was not included in final submission. The final submission uses last write caching only.

Results

Area

```
Number of ports:              148
Number of nets:              7960
Number of cells:             7218
Number of references:          54
Combinational area:       614907.000000
Noncombinational area:    384488.000000
Total cell area:          999395.000000
```

Timing

```
Point                                     Incr       Path
  -----------------------------------------------------------
  clock CLK (rise edge)                    0.00       0.00
  clock network delay (ideal)              0.00       0.00
  inAddr_reg[2]/CLK (DFFPOSX1)             0.00 #     0.00 r
  inAddr_reg[2]/Q (DFFPOSX1)               0.19       0.19 f
  U5424/Y (INVX2)                          0.06       0.26 r
  U5179/Y (BUFX4)                          0.11       0.37 r
  U4904/Y (INVX4)                          0.09       0.46 f
  U7378/Y (XNOR2X1)                        0.11       0.57 f
  U7379/Y (NAND2X1)                        0.06       0.63 r
  U7380/Y (NOR2X1)                         0.06       0.69 f
  U7381/Y (NAND3X1)                        0.10       0.80 r
  U5308/Y (INVX1)                          0.08       0.87 f
  U5062/Y (AND2X2)                         0.11       0.99 f
  U7601/Y (NAND2X1)                        0.06       1.05 r
  U7602/Y (NOR2X1)                         0.08       1.13 f
  U7610/Y (AOI21X1)                        0.13       1.26 r
  U4976/Y (NAND2X1)                        0.06       1.32 f
  U7642/Y (NOR3X1)                         0.12       1.45 r
  U4867/Y (NAND2X1)                        0.05       1.50 f
  U5174/Y (AND2X2)                         0.13       1.63 f
  U8065/Y (NAND2X1)                        0.10       1.72 r
  U5141/Y (INVX2)                          0.10       1.83 f
  U8037/Y (OAI21X1)                        0.08       1.91 r
  readData_reg[0]/D (DFFSR)                0.00       1.91 r
  data arrival time                                   1.91

  clock CLK (rise edge)                    2.00       2.00
  clock network delay (ideal)              0.00       2.00
  readData_reg[0]/CLK (DFFSR)              0.00       2.00 r
  library setup time                      -0.09       1.91
  data required time                                  1.91
  -----------------------------------------------------------
  data required time                                  1.91
  data arrival time                                  -1.91
  -----------------------------------------------------------
  slack (MET)                                         0.00
```