**DDR Configure**

**Clock Generate**

**Data Capture**

**DDR**

**Controller**

**DDR**

**R/W\_BURST**

**DDR**

**Refresh/PWD**

**Output Alignment**

**Data In**

**Clock**

**Addr**

**Data**

**Ctrl Signals**

**Captured Clock**

**Diff Clock**

**Cmd**

**Data In/Out**

From Memory

Manager Unit

To DDR4 Memory

**Rdy**

**DDR4 Memory Controller (MemCtrl) Architecture**

The architecture is to describe set of the inputs, outputs and main block functions for simplified MemCtrl (we target x8 address mode, read,write, refresh )

1. **Inputs/Outputs to Memory Management unit:** The DDR4 MemCtrl will receive/output the following signals from Memory Controller unit:

* Input Clock Signal.
* Memory Address for read/write
* Write Data
* Control Signals: Read, Write request, Reset, Number of bytes for Read or Write.
* DDR Options to set up the option for the device.
* Rdy: It is a handshake signals to MMU to not receive any request while it is busy. So one request as a time.

1. **Input/ Outputs:** The DDR4 MemCtrl will output the following signals to DDR4 Memory:

* Differential Clock Signals
* Commands: groups of signals, i.g CKE (clock enable), CS\_n, ACT\_n and Bank Groups, Bank Address, Columns Address, etc.
* Data In/Out: Output data for write, Input data for read.

1. **Main Functions:**
2. **Clock Capture:** move to the Interface
3. **Data Capture:**

As Faust wants, we design the queue to store the R/W request (address, data, r/w) and implement the function to sort the queue to tbd algorithm or just let first come first serve.

Stop capture data when Controller in Initialization, Prefresh, Update.

1. **DDR Controller:**

The block is implemented to arbitrate between Initialization, Refresh, Update and Burst RD/WR as the FSM below. When DDR Controller receivers Update command, or Prefresh timer expires, the Controller should stop capture the data and let the R/W Burst finish all the data in queue.

1. **DDR Configuration:**

There is a straight forward, simple state machine. Just concern about the timing between the commands and capture the timing parameters set in the MRS register. Ignore the rest.

Implement the FSM for sequence of commands of Configuration DDR4 device (section 3.3 Step 5 to step 15). The state machine transitions are at the rising clock of the capture edge. The CS\_n asserts low during transmitting a command.

Note the following

After receiving Initialization Request, assert CKE high after 500us – Tis (not find Tis anywhere in standard – set to zero)

Wait for tXPR, then asserts the first Mode Register Set (MRS) (step 6 in standard).

Each MRS asserts for 1 clock cycle (as shown in diagram). The next MRS will be after tMRD. (There are DES commands in between MRS)

The spacing between ZQCL command and previous MRS is Tmod.

After the ZQCL command, the last command, wait for Tzqint, then assert Done signal to DDR Controller block.

Use the number clock cycle( = time/ clock period) as the time counter. The transition between states machine will be at the rising edge of the captured clock.

1. **DDR Pre-fresh/Power down:**

Sometime of the timer to counter down tREFi, when close to zero, assert the Controller to stop the data capture. And when in PreFresh Mode, asserts the refresh command.

1. **DDR Update:**

Receives the update command, the Controller will stop capture data and wait to become idle before enter this states. Send out MRS to change the parameters. Similar to Initialization process, just concern about tMRP and tMOD.

1. **DDR R/W Burst**

The following items are not implemented in projects:

* CS to Command Address latency (CAL mode): add latency between assert CS\_n to Address/Command. In the project, CS\_n asserts in same clock cycle as Command/Address.
* C/A Parity Latency: Disable. Not quite sure what function of this feature but will add more complexity to the state machine. So PL (Parity Latency = 0)
* Read DBI (Data Bus Inversion) as well as write CRC: The feature is to inverse the data bus when more than 4 0-bits in a byte in order to save power. It will add more complexity in the state machine.

**Timing Definition: (also for assertion)**

* tCCD : CAS to CAS delay (applied for consecutive read or write to different bank group or same bank group. Note not for the same bank). There are 2 numbers, one for different bank group (short one) and one for same group (long one). We uses the long one so don’t have to worry about bank group matter
* tRRD: Activate to Activate Command period. (note this is to activate different bank). Same as tCCD, we use the long one to cover both cases of bank group.
* tWRT: Write to Read delay. Also use the long one to cover both cases.
* tRTP: Read to PreCharge. Set in MR0[A12:9]
* tWR: Write to PreCharge (recover time). Set in MR0[A12:9]
* tRCD: Activate Command to CAS. (not listed in section 12 but in section 9 or maybe programmed as tCL or tCWL. need to verify)
* tCL: delay between CAS to data out for read mode. Control by MR0 [A6:2] for # clock cycles
* tCWL: delay between CAS to data ready for write. Control by MR2 [A5:3] for # of clock cycles
* AL : Additive Latency to increase number of clock cycle between CAS and data. Set in MR1[a4:3] to 0, CL-1 or CL-2.
* Burst Length: set in MR0[A1:A0] as BL8, 4 or on the Fly.
* Preamble: either 1tCK or 2 set in MR4[A12]

Burst READ/Write operation

while I was looking at the single burst read/write state machine with Interface design. I realized that If I implement as different way as one process for ACTIVE, one process for CAS, and one for DATA, then I can implement the DDR Controller with interleaving operation without much coding and simpler state machine. In the new implementation as shown below, we can have design true burst functing operating.(or interleaving).

The timing parameters for same bank group and different bank group, are sometimes are just few ns or 1, 2 clock cycles. We are going to use the greater number so we don’t have to check the bank group and simplify our design.

Some of the timing parameters are programed during initialization process or during the update. The rest can be initialized as parameters. Some functions are implemented to compute some parameters such as CL, CWL,etc. can be during initialization and update.

For the ACT:

The Controller DDR activates any bank, and leave it open until has another request to access this same bank with different row or refresh command activate.

The Controller receives new request. Check to see the bank activates. If not, send ACT. If already open, and same row address, send CAS (e.g R/W) command. Else, send PRE.

After tRRD, get the next request, start the sequence again.

(there is an fixed array to keep track which bank is activated. and have store bank group, bank address and row address, and clear after PRE)

The CAS has to satisfy both tCCD (CAS to CAS) and tRCD (ACT to CAS). However, the time delay between ACT, tRRD, is greater than tCCD. And the ACT process above will check new request at least at tRRD rate, then the tCCD can be ignored

the CAS process as following:

Wait for tRCD after ACT, asserting CAS. (1st after idle)

wait for tRRD, check for new ACT, if not then go back to idle

if same operation as previous request, then assert CAS.

If not wait for the data from previous request complete, than assert CAS after ?? clock cycle

There should be implemented function to compute the wait time between read to write or write to read.

if read-to-write: wait time is BL/2 + CWL+AL+ 1+ Preamble

if write-to-read: wait time is BL/2 + tWTR (write to read delay)

For data process for read/write data:

The data out is constrained by WL and CL number clock cycles from the CAS asserted. To implement for burst read/write operation, there are some cases of number of CAS commands asserted before the first data appear on the data bus.

I use a queue to keep track when the CAS occur relative to current counter. Basic to calculate how many wait clock cycles need after current data out.

For example, if CAS occurs at clock cycle t0, t3, t6, t9, and t12. The CL is 11 cycles. So the queue will store {8, 5, 2}. There are number cycles that each CAS waited in the counter for t0 CAS . When data out shows up at t11 cycle, counter is clear. The t3 CAS is now already waited for 8 cycles. So only CL - 8 = 3 cycles for t3 CAS data. And there queue is updated {8, 5} the cycles that CAS at 6th and 9th will waited after t3 CAS data out. At t12 cycle, another CAS, the queue is updated {8,5, 2}. 2 in the queue is cycles that t12 CAS will wait. At t14, data is shown up for t3 CAS, queue is update {8,5} and t6 CAS only needs CL-8 for its data. At t17, t6 CAS data out, queue is update to {8}. 8 is cycle that t12 cas waited util t9 CAS out.

There will be some handshake in the FSMs to avoid asserting commands in same clock cycles. Add delays

**Burst Read Operation**

The block is to implement the FSM for the simple single burst read command followed by Pre-Charge. Note the DQS (Strobe signals) must high impedance during the read.

**Burst Read state machine:**

tRC: min spacing from ACT to next ACT

tRAS: min spacing from ACT to Pre Charge Command

tCL: CAS Latency

tRCD: spacing between ACT to Read Command

tRP: spacing from Pre Charge to ACT. Program in MR0[A11:9]

All the parameters above defines in section 9.

t1: start at entering state READ\_NOP1

t2: start at entering state READ\_NOP2

t3: start at entering state READ\_NOP3

t4: start at entering state READ\_NOP4

Call the mapping function for Topological address as in HW3 before enter READ\_ACT

Send out Activate Command in READ\_ACT state as describe in Table 16

Nop Command in READ\_NOP1, 2,3, 4.

Send out Read Command in READ\_CMD. The Read Command is correspond to the setting in MR0[A1:A0] as BL8, BC4 or on the fly.

Assert Read signal to Data In in READ\_DATA state to start capture data from DDR device. Timing should include PreAmble read timing.

Assert PreCharge Command in READ\_PRECHARGE state.

Assert Done to DDR CONTROLLER block for the next request.

(Note: we are not going to implement any back to back reads or read/write for now until we have this design work at least. Include back to back read or read write substantially increase time in both design and test bench)

rising\_clk

t1 < tRCD

rising\_clk

rising\_clk

rising\_clk

rising\_clk

t2 <AL + CL

rising\_clk

rising\_clk

(t3<tRP).(t1<tRAS

rising\_clk

(t4 < tRP). (t1<tRC)

**Burst WRITE Operation**

The block is to implement the FSM for the simple single burst write followed by Pre-Charge.

**Burst Write state machine:**

tRC: min spacing from ACT to next ACT

tRAS: min spacing from ACT to Pre Charge Command

tCWL: CAS Write Latency (From Write to data). Program in MR2[A5:3]

tRCD: spacing between ACT to Read Command

tWR: Write Recover from next rising\_edge after last data to PRE. Program in MR0[A12:9]

All the parameters above defines in section 9.

t1: start at entering state WRITE\_NOP1

t2: start at entering state WRITE\_NOP2

t3: start at entering state WRITE\_DATA (half of burst length)

t4: start at entering state READ\_NOP4

Call the mapping function for Topological address as in HW3 before enter WIRTE\_ACT

Send out Activate Command in WRITE\_ACT state as describe in Table 16

Nop Command in WRITE\_NOP1, 2,3, 4.

Assert differential Strobe signals at WRITE\_STROBE state (one or 2 clock cycle ahead of data for preamble write)

Write data in WRITE\_DATA

Assert PreCharge Command in PRECHARGE state.

Omit the wait states for the next ACT (similar with last 2 states in read fsm)

(Note: we are not going to implement any back to back reads or read/write for now until we have this design work at least. Include back to back read or read write substantially increase time in both design and test bench)

t1 < tRCD

rising\_clk

rising\_clk

t2<AL + CWL -tCK

rising\_clk

rising\_clk

t3 < BL/2 (BL =4, or 8)

rising\_clk

t4< tWR

rising\_clk

1. **Output alignment:**

Multiplex the signals set by DDR Configure, Burst R/W and Prefresh block to output pins.

Alignment the signals to differential clock and strobe signals (DQS) output.

Configure the Data Bus and DQS as tri-state bus.

1. **Data In:** Capture the read data from DDR4 Memory when receives Read signal from Burst Read State Machine.