✏️

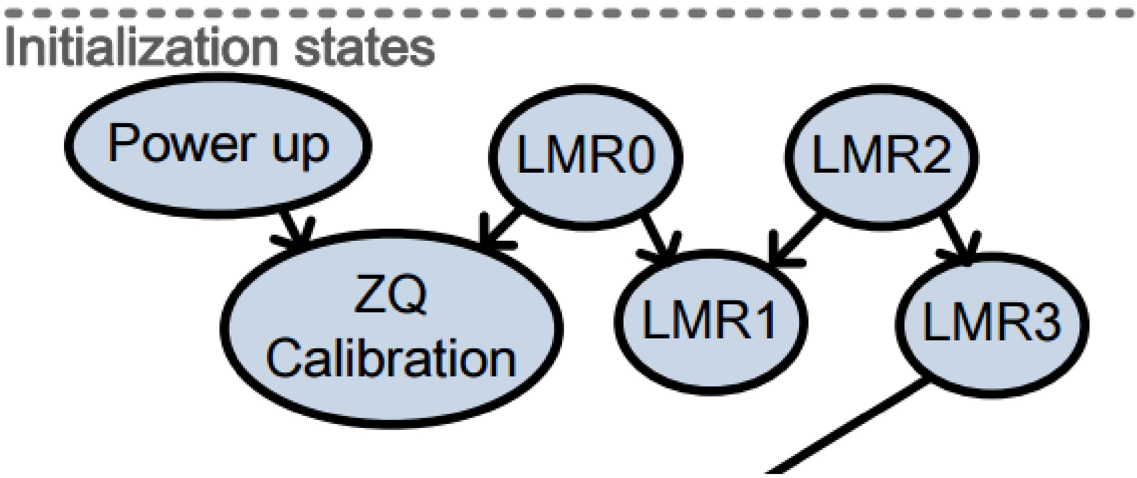
Senior’s DRAM Notes

Created	@January 3, 2025 5:05 PM
Class	

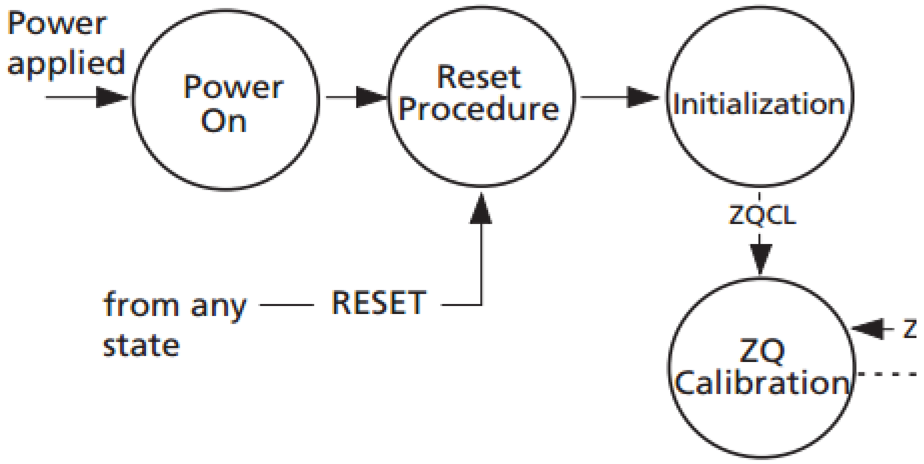
- List its triggers and controllers from state machine, data path, timing constraints counter and PHY layer interaction.

Initialization

Senior’s state machine for initialization



DDR3 specifications

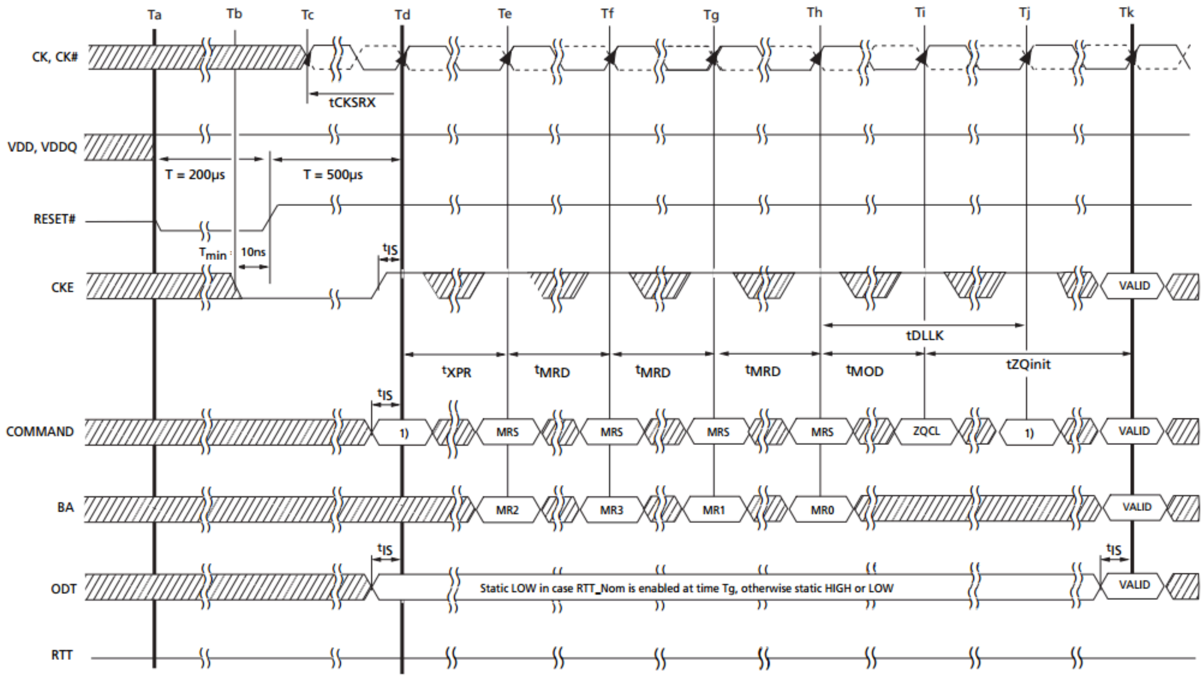


- The process of initialization is initialized through the initialization counter

Power up Procedure

3.3 RESET and Initialization Procedure (Cont'd)

3.3.1 Power-up Initialization Sequence (Cont'd)



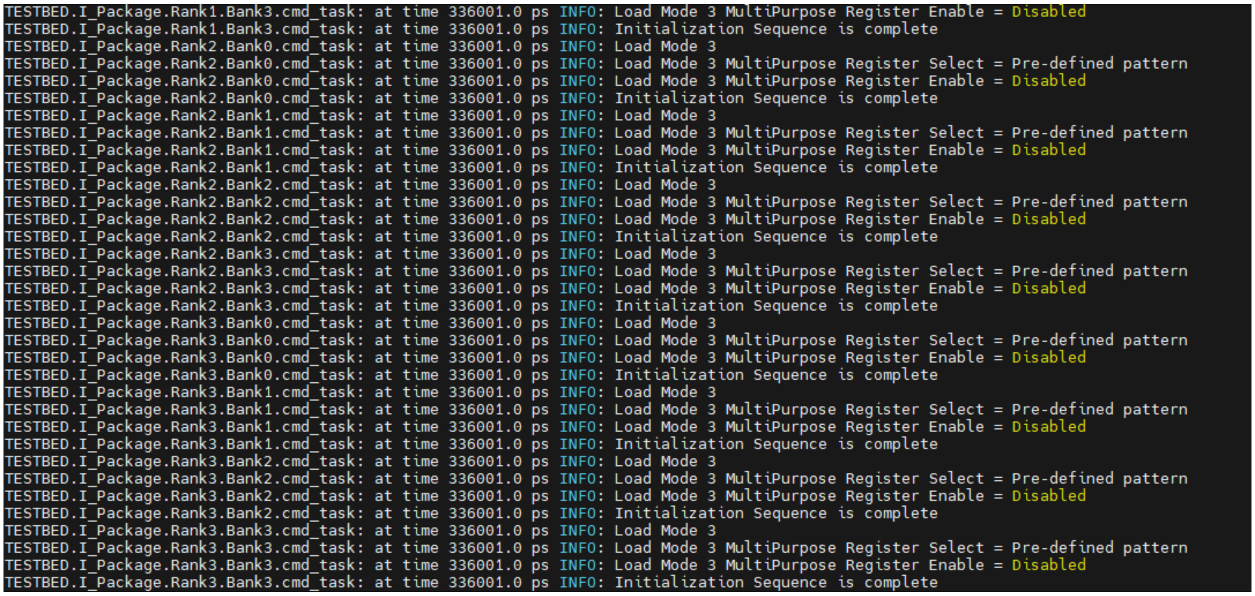
NOTE 1. From time point "Td" until "Tk" NOP or DES commands must be applied between MRS and ZQCL commands.

TIME BREAK ▨ DON'T CARE

Figure 5 — Reset and Initialization Sequence at Power-on Ramping

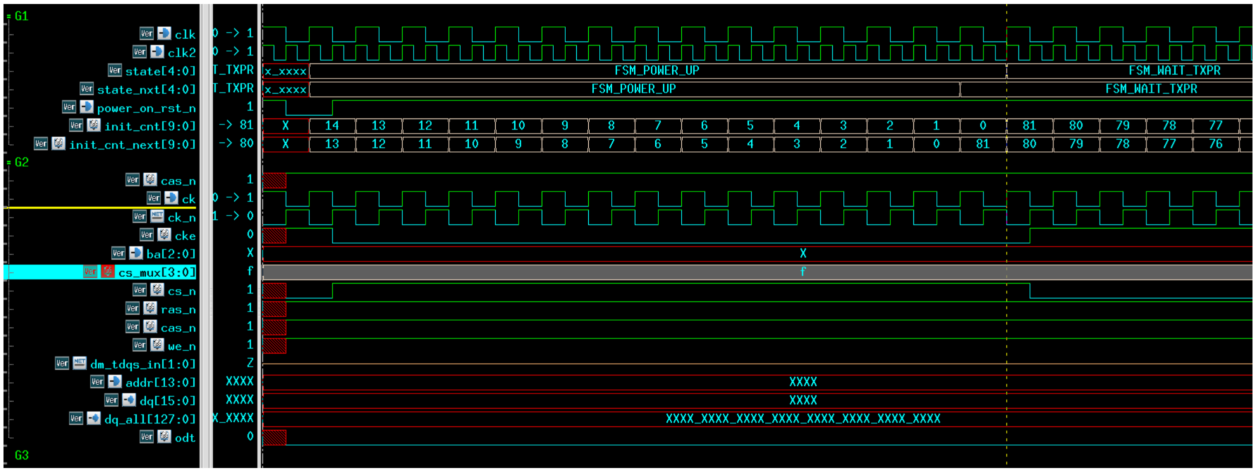
- CK,CK# are the clocks, needed timing constraints are tCKSRX,tXPR,tMRD,tMOD,tZQinit & tDLLK,

must first wait 200us later 500 us for the start up

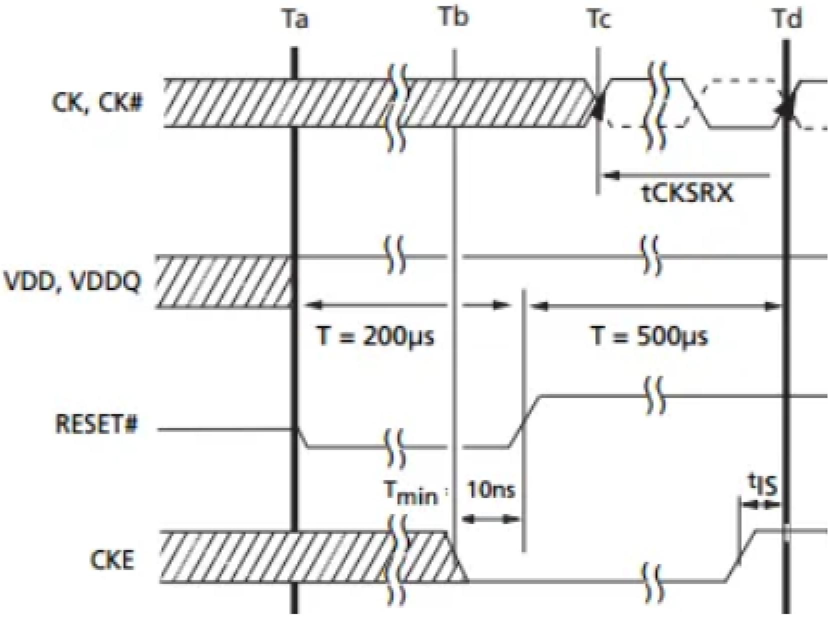


```
TESTBED.I_Package.Rank1.Bank3.cmd_task: at time 336001.0 ps INFO: Load Mode 3 MultiPurpose Register Enable = Disabled
TESTBED.I_Package.Rank1.Bank3.cmd_task: at time 336001.0 ps INFO: Initialization Sequence is complete
TESTBED.I_Package.Rank2.Bank0.cmd_task: at time 336001.0 ps INFO: Load Mode 3
TESTBED.I_Package.Rank2.Bank0.cmd_task: at time 336001.0 ps INFO: Load Mode 3 MultiPurpose Register Select = Pre-defined pattern
TESTBED.I_Package.Rank2.Bank0.cmd_task: at time 336001.0 ps INFO: Load Mode 3 MultiPurpose Register Enable = Disabled
TESTBED.I_Package.Rank2.Bank0.cmd_task: at time 336001.0 ps INFO: Initialization Sequence is complete
TESTBED.I_Package.Rank2.Bank1.cmd_task: at time 336001.0 ps INFO: Load Mode 3
TESTBED.I_Package.Rank2.Bank1.cmd_task: at time 336001.0 ps INFO: Load Mode 3 MultiPurpose Register Select = Pre-defined pattern
TESTBED.I_Package.Rank2.Bank1.cmd_task: at time 336001.0 ps INFO: Load Mode 3 MultiPurpose Register Enable = Disabled
TESTBED.I_Package.Rank2.Bank1.cmd_task: at time 336001.0 ps INFO: Initialization Sequence is complete
TESTBED.I_Package.Rank2.Bank2.cmd_task: at time 336001.0 ps INFO: Load Mode 3
TESTBED.I_Package.Rank2.Bank2.cmd_task: at time 336001.0 ps INFO: Load Mode 3 MultiPurpose Register Select = Pre-defined pattern
TESTBED.I_Package.Rank2.Bank2.cmd_task: at time 336001.0 ps INFO: Load Mode 3 MultiPurpose Register Enable = Disabled
TESTBED.I_Package.Rank2.Bank2.cmd_task: at time 336001.0 ps INFO: Initialization Sequence is complete
TESTBED.I_Package.Rank2.Bank3.cmd_task: at time 336001.0 ps INFO: Load Mode 3
TESTBED.I_Package.Rank2.Bank3.cmd_task: at time 336001.0 ps INFO: Load Mode 3 MultiPurpose Register Select = Pre-defined pattern
TESTBED.I_Package.Rank2.Bank3.cmd_task: at time 336001.0 ps INFO: Load Mode 3 MultiPurpose Register Enable = Disabled
TESTBED.I_Package.Rank2.Bank3.cmd_task: at time 336001.0 ps INFO: Initialization Sequence is complete
TESTBED.I_Package.Rank3.Bank0.cmd_task: at time 336001.0 ps INFO: Load Mode 3
TESTBED.I_Package.Rank3.Bank0.cmd_task: at time 336001.0 ps INFO: Load Mode 3 MultiPurpose Register Select = Pre-defined pattern
TESTBED.I_Package.Rank3.Bank0.cmd_task: at time 336001.0 ps INFO: Load Mode 3 MultiPurpose Register Enable = Disabled
TESTBED.I_Package.Rank3.Bank0.cmd_task: at time 336001.0 ps INFO: Initialization Sequence is complete
TESTBED.I_Package.Rank3.Bank1.cmd_task: at time 336001.0 ps INFO: Load Mode 3
TESTBED.I_Package.Rank3.Bank1.cmd_task: at time 336001.0 ps INFO: Load Mode 3 MultiPurpose Register Select = Pre-defined pattern
TESTBED.I_Package.Rank3.Bank1.cmd_task: at time 336001.0 ps INFO: Load Mode 3 MultiPurpose Register Enable = Disabled
TESTBED.I_Package.Rank3.Bank1.cmd_task: at time 336001.0 ps INFO: Initialization Sequence is complete
TESTBED.I_Package.Rank3.Bank2.cmd_task: at time 336001.0 ps INFO: Load Mode 3
TESTBED.I_Package.Rank3.Bank2.cmd_task: at time 336001.0 ps INFO: Load Mode 3 MultiPurpose Register Select = Pre-defined pattern
TESTBED.I_Package.Rank3.Bank2.cmd_task: at time 336001.0 ps INFO: Load Mode 3 MultiPurpose Register Enable = Disabled
TESTBED.I_Package.Rank3.Bank2.cmd_task: at time 336001.0 ps INFO: Initialization Sequence is complete
TESTBED.I_Package.Rank3.Bank3.cmd_task: at time 336001.0 ps INFO: Load Mode 3
TESTBED.I_Package.Rank3.Bank3.cmd_task: at time 336001.0 ps INFO: Load Mode 3 MultiPurpose Register Select = Pre-defined pattern
TESTBED.I_Package.Rank3.Bank3.cmd_task: at time 336001.0 ps INFO: Load Mode 3 MultiPurpose Register Enable = Disabled
TESTBED.I_Package.Rank3.Bank3.cmd_task: at time 336001.0 ps INFO: Initialization Sequence is complete
```

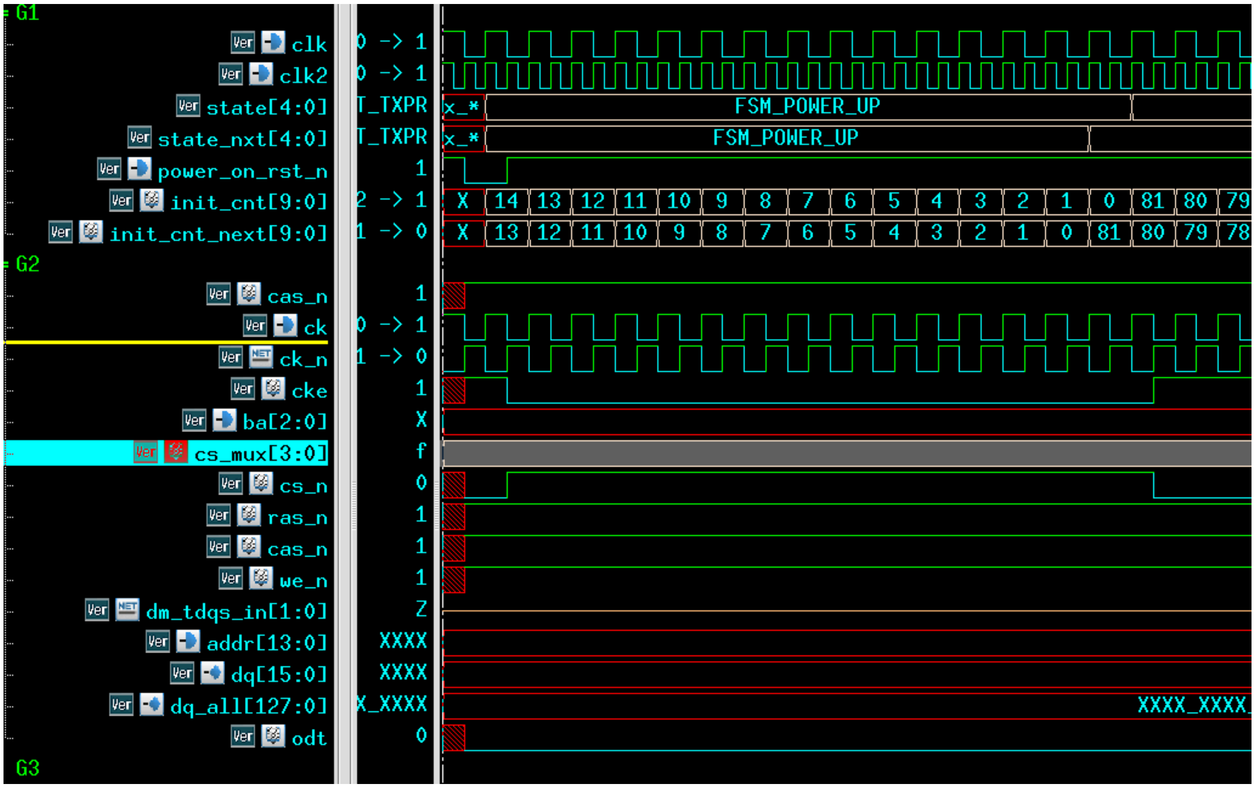
- This should be seen if the initialization process is correct!


G1
clk 0 -> 1
clk2 0 -> 1
state[4:0] T_TXPR x_xxxx FSM_POWER_UP FSM_WAIT_TXPR
state_nxt[4:0] T_TXPR x_xxxx FSM_POWER_UP FSM_WAIT_TXPR
power_on_rst_n 1
init_cnt[9:0] -> 81 X 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 81 80 79 78 77
init_cnt_next[9:0] -> 80 X 13 12 11 10 9 8 7 6 5 4 3 2 1 0 81 80 79 78 77 76
G2
cas_n 1
ck 0 -> 1
ck_n 1 -> 0
cke 0
ba[2:0] X X
cs_mux[3:0] f f
cs_n 1
ras_n 1
cas_n 1
we_n 1
dm_tdqs_in[1:0] Z
addr[13:0] XXXX XXXX
dq[15:0] XXXX XXXX
dq_all[127:0] X_XXXX XXXX_XXXX_XXXX_XXXX_XXXX_XXXX_XXXX_XXXX
odt 0
G3

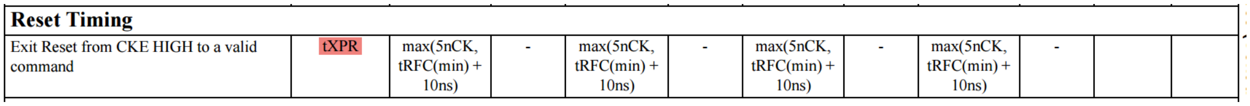
- Strange thing happens when the power up does not proceed for 200us, which is strange



- **The initialization process should be 200 us later 500 us, senior’s design does not consider this case. Which is very strange!**

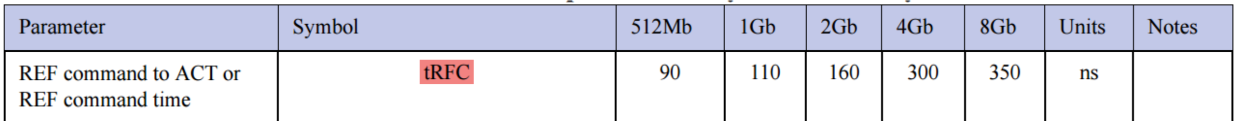

G1
clk 0 -> 1
clk2 0 -> 1
state[4:0] T_TXPR x_* FSM_POWER_UP
state_nxt[4:0] T_TXPR x_* FSM_POWER_UP
power_on_rst_n 1
init_cnt[9:0] 2 -> 1 X 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 81 80 79
init_cnt_next[9:0] 1 -> 0 X 13 12 11 10 9 8 7 6 5 4 3 2 1 0 81 80 79 78
G2
cas_n 1
ck 0 -> 1
ck_n 1 -> 0
cke 1
ba[2:0] X
cs_mux[3:0] f
cs_n 0
ras_n 1
cas_n 1
we_n 1
dm_tdqs_in[1:0] Z
addr[13:0] XXXX
dq[15:0] XXXX
dq_all[127:0] X_XXXX XXXX_XXXX_
odt 0
G3

- According to DRAM spec, a minimum of 200us + 500 us is required for powering up, this does not make sense.



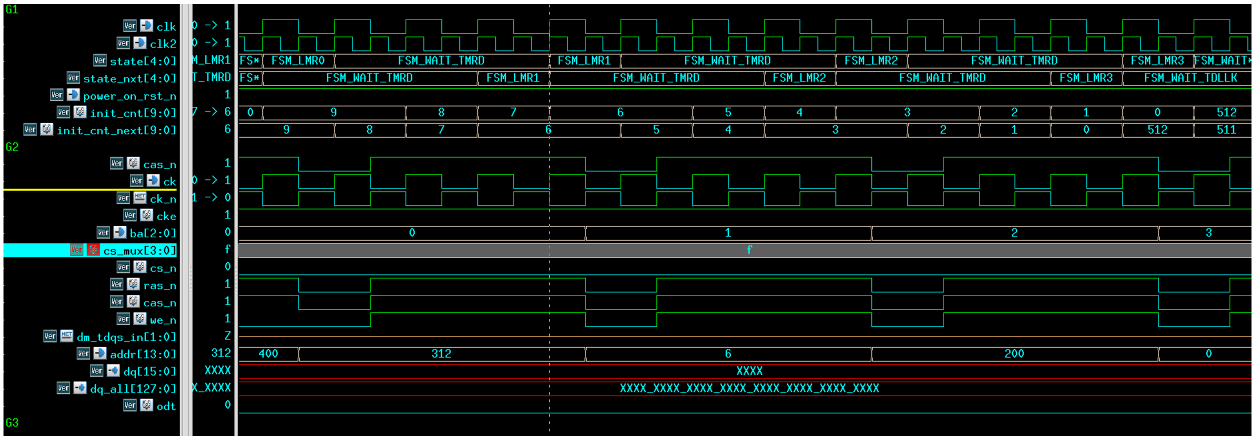
Reset Timing											
Exit Reset from CKE HIGH to a valid command	tXPR	max(5nCK, tRFC(min) + 10ns)	-	max(5nCK, tRFC(min) + 10ns)	-	max(5nCK, tRFC(min) + 10ns)	-	max(5nCK, tRFC(min) + 10ns)	-		

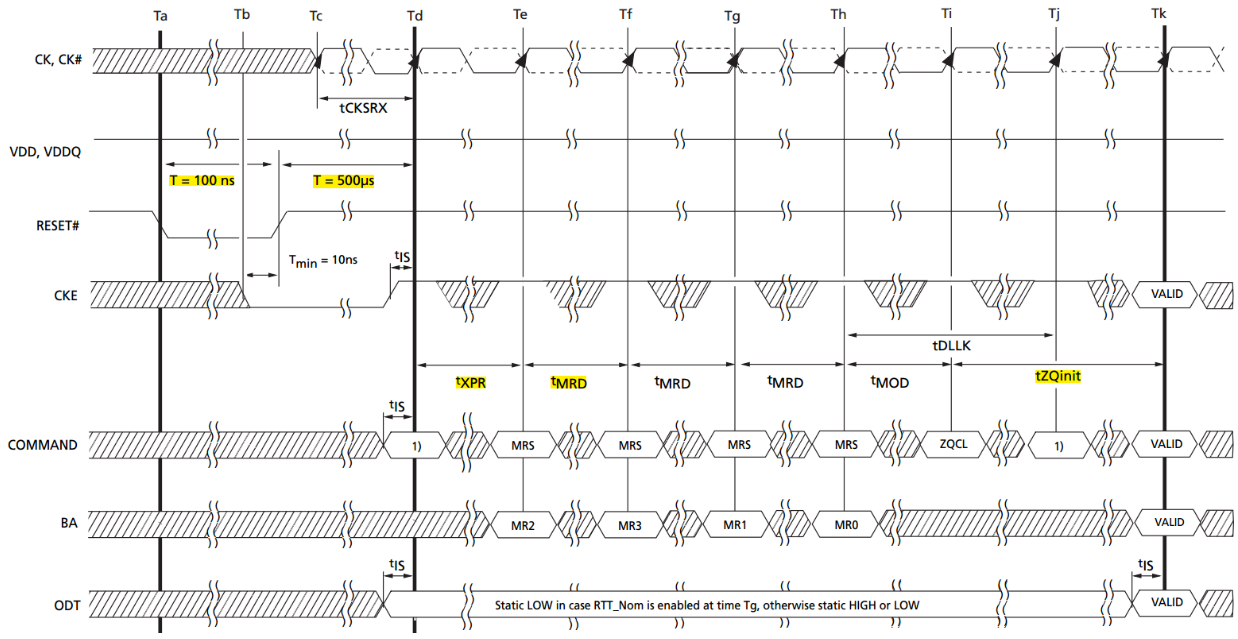
- The refest timing tXPR for powering up, the max of (110+10) or (5nCK)



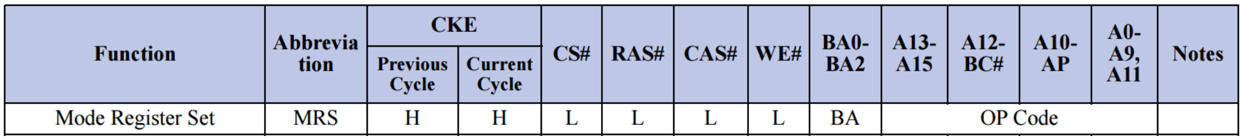
Parameter	Symbol	512Mb	1Gb	2Gb	4Gb	8Gb	Units	Notes
REF command to ACT or REF command time	tRFC	90	110	160	300	350	ns	

Set Mode Registers





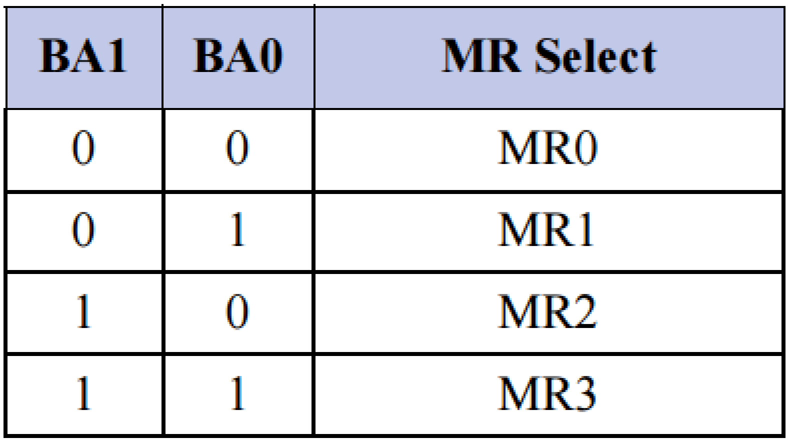
- Mode register COMMAND



Function	Abbreviation	CKE		CS#	RAS#	CAS#	WE#	BA0-BA2	A13-A15	A12-BC#	A10-AP	A0-A9, A11	Notes
		Previous Cycle	Current Cycle										
Mode Register Set	MRS	H	H	L	L	L	L	BA	OP Code				

- Notice that CKE must follow and previous cycle, current cycle routine to indicate that this is the correct signals and commands.

Accessing MRx



BA1	BA0	MR Select
0	0	MR0
0	1	MR1
1	0	MR2
1	1	MR3

- BA address number is used for MRx register selection

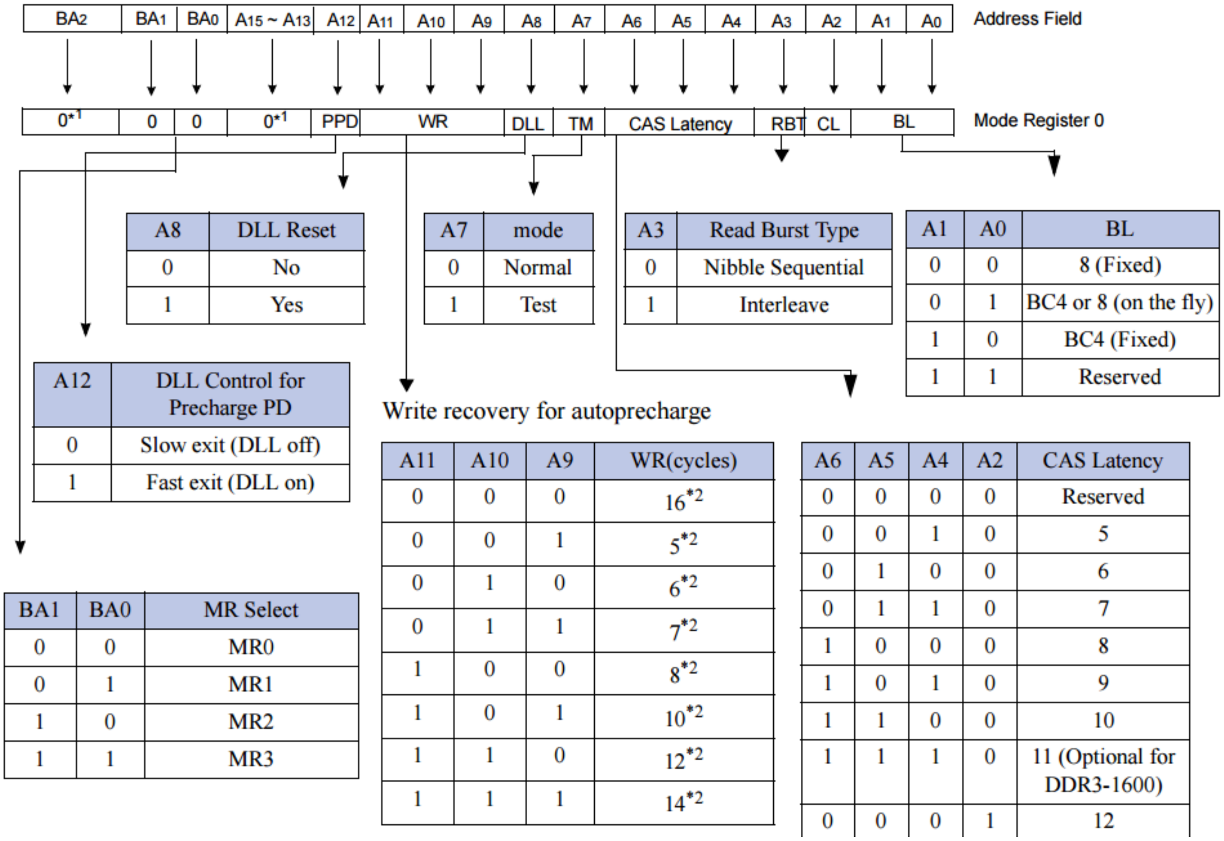
Mode Register 0

- Burst length, Burst mode, DLL initialization are set here.

3.4 Register Definition (Cont'd)

3.4.2 Mode Register MR0 (Cont'd)

while controlling the states of address pins according to Figure 9.



A8	DLL Reset
0	No
1	Yes

A7	mode
0	Normal
1	Test

A3	Read Burst Type
0	Nibble Sequential
1	Interleave

A1	A0	BL
0	0	8 (Fixed)
0	1	BC4 or 8 (on the fly)
1	0	BC4 (Fixed)
1	1	Reserved

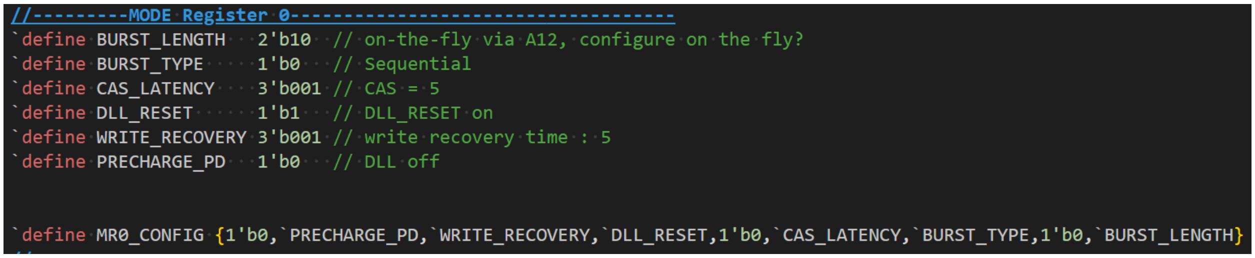
A12	DLL Control for Precharge PD
0	Slow exit (DLL off)
1	Fast exit (DLL on)

Write recovery for autoprecharge

A11	A10	A9	WR(cycles)
0	0	0	16*2
0	0	1	5*2
0	1	0	6*2
0	1	1	7*2
1	0	0	8*2
1	0	1	10*2
1	1	0	12*2
1	1	1	14*2

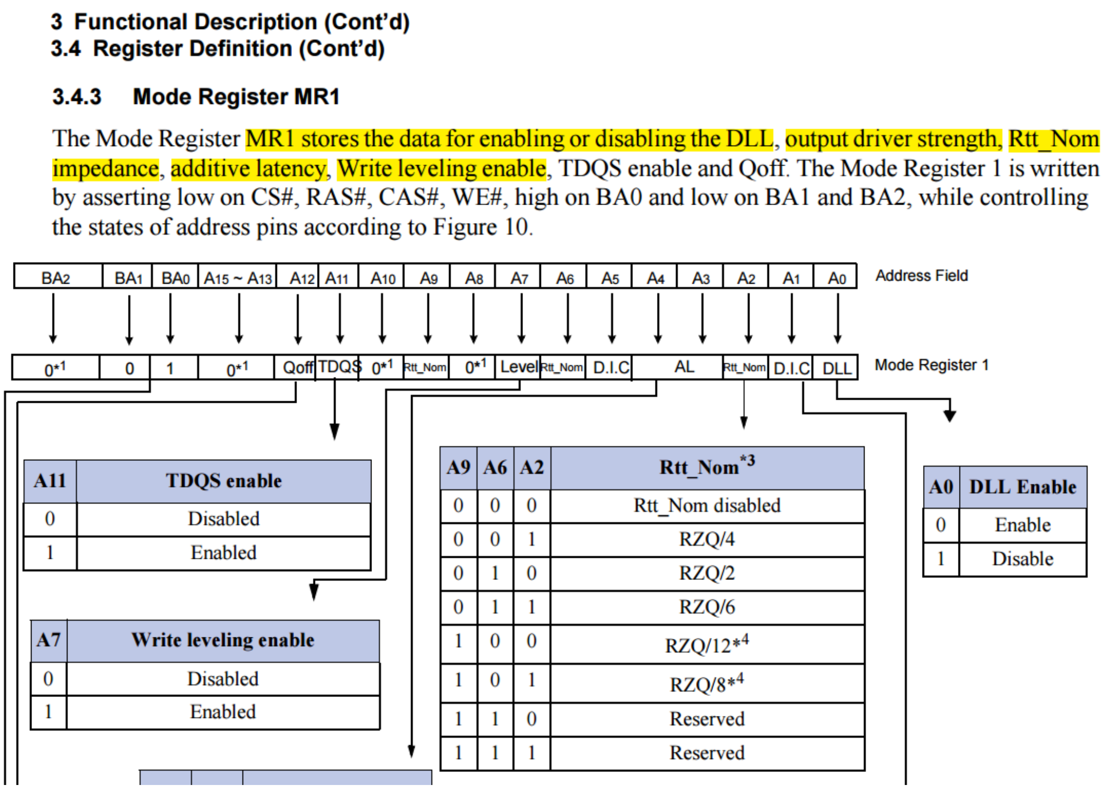
A6	A5	A4	A2	CAS Latency
0	0	0	0	Reserved
0	0	1	0	5
0	1	0	0	6
0	1	1	0	7
1	0	0	0	8
1	0	1	0	9
1	1	0	0	10
1	1	1	0	11 (Optional for DDR3-1600)
0	0	0	1	12

BA1	BA0	MR Select
0	0	MR0
0	1	MR1
1	0	MR2
1	1	MR3



```
//---------MODE Register 0-----------------------------------
`define BURST_LENGTH   2'b10  // on-the-fly via A12, configure on the fly?
`define BURST_TYPE     1'b0   // Sequential
`define CAS_LATENCY    3'b001 // CAS = 5
`define DLL_RESET      1'b1   // DLL_RESET on
`define WRITE_RECOVERY 3'b001 // write recovery time : 5
`define PRECHARGE_PD   1'b0   // DLL off


`define MR0_CONFIG {1'b0,`PRECHARGE_PD,`WRITE_RECOVERY,`DLL_RESET,1'b0,`CAS_LATENCY,`BURST_TYPE,1'b0,`BURST_LENGTH}
```

Mode Register 1



3 Functional Description (Cont'd)
3.4 Register Definition (Cont'd)

3.4.3 Mode Register MR1

The Mode Register MR1 stores the data for enabling or disabling the DLL, output driver strength, Rtt_Nom impedance, additive latency, Write leveling enable, TDQS enable and Qoff. The Mode Register 1 is written by asserting low on CS#, RAS#, CAS#, WE#, high on BA0 and low on BA1 and BA2, while controlling the states of address pins according to Figure 10.

BA2	BA1	BA0	A15 ~ A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	Address Field
0^{*1}	0	1	0^{*1}	Qoff	TDQS	0^{*1}	Rtt_Nom	0^{*1}	Level	Rtt_Nom	D.I.C	AL		Rtt_Nom	D.I.C	DLL	Mode Register 1

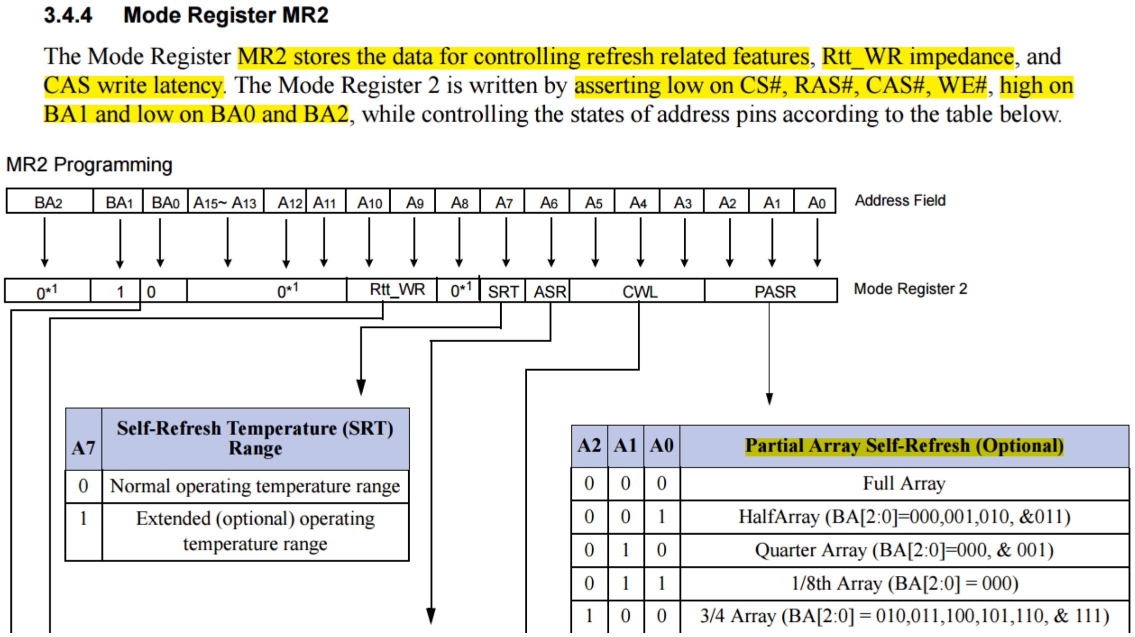
A11	TDQS enable
0	Disabled
1	Enabled

A7	Write leveling enable
0	Disabled
1	Enabled

A9	A6	A2	Rtt_Nom^{*3}
0	0	0	Rtt_Nom disabled
0	0	1	RZQ/4
0	1	0	RZQ/2
0	1	1	RZQ/6
1	0	0	RZQ/12^{*4}
1	0	1	RZQ/8^{*4}
1	1	0	Reserved
1	1	1	Reserved

A0	DLL Enable
0	Enable
1	Disable

MR3 Register



3.4.4 Mode Register MR2

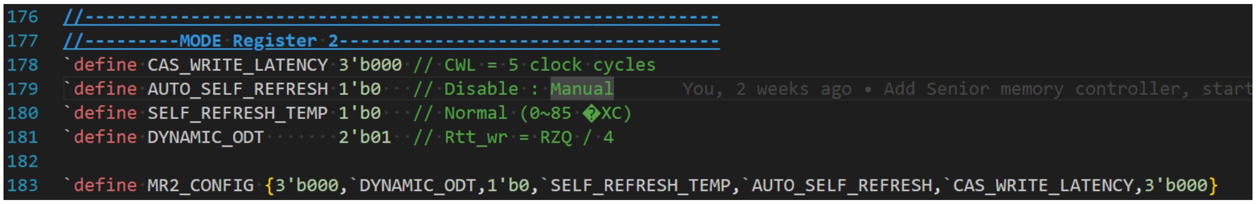
The Mode Register MR2 stores the data for controlling refresh related features, Rtt_WR impedance, and CAS write latency. The Mode Register 2 is written by asserting low on CS#, RAS#, CAS#, WE#, high on BA1 and low on BA0 and BA2, while controlling the states of address pins according to the table below.

MR2 Programming

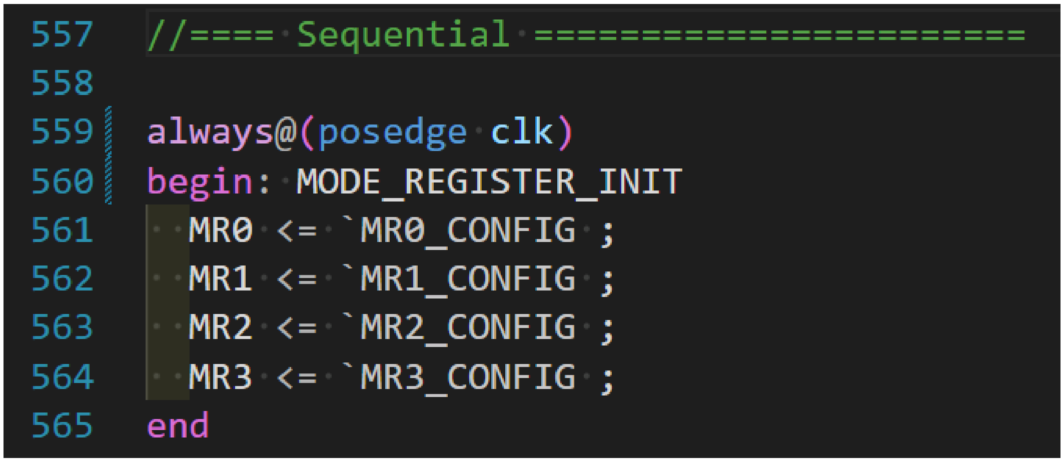
BA2	BA1	BA0	A15~ A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	Address Field
0^{*1}	1	0	0^{*1}			Rtt_WR		0^{*1}	SRT	ASR	CWL			PASR			Mode Register 2

A7	Self-Refresh Temperature (SRT) Range
0	Normal operating temperature range
1	Extended (optional) operating temperature range

A2	A1	A0	Partial Array Self-Refresh (Optional)
0	0	0	Full Array
0	0	1	HalfArray (BA[2:0]=000,001,010, &011)
0	1	0	Quarter Array (BA[2:0]=000, & 001)
0	1	1	1/8th Array (BA[2:0] = 000)
1	0	0	3/4 Array (BA[2:0] = 010,011,100,101,110, & 111)

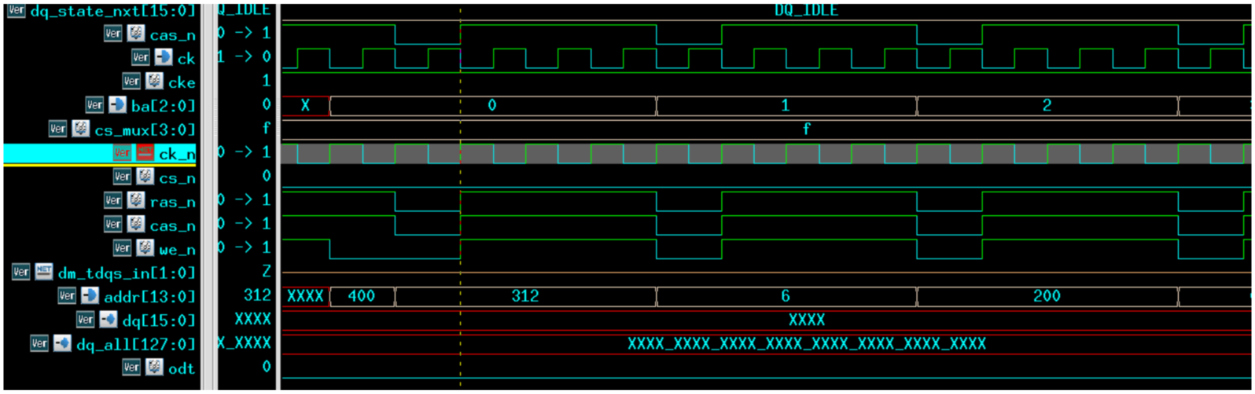


```verilog
//----------------------------------------------------------
//---------MODE Register 2-----------------------------------
`define CAS_WRITE_LATENCY 3'b000 // CWL = 5 clock cycles
`define AUTO_SELF_REFRESH 1'b0   // Disable : Manual      You, 2 weeks ago • Add Senior memory controller, start
`define SELF_REFRESH_TEMP 1'b0   // Normal (0~85 �XC)
`define DYNAMIC_ODT       2'b01  // Rtt_wr = RZQ / 4

`define MR2_CONFIG {3'b000,`DYNAMIC_ODT,1'b0,`SELF_REFRESH_TEMP,`AUTO_SELF_REFRESH,`CAS_WRITE_LATENCY,3'b000}
```

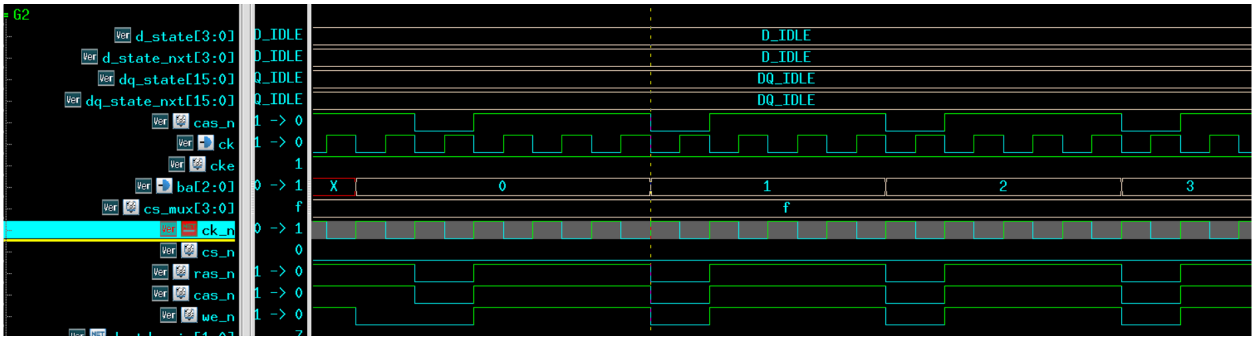


```verilog
//==== Sequential ========================

always@(posedge clk)
begin: MODE_REGISTER_INIT
  MR0 <= `MR0_CONFIG ;
  MR1 <= `MR1_CONFIG ;
  MR2 <= `MR2_CONFIG ;
  MR3 <= `MR3_CONFIG ;
end
```

Mode Register interaction with PHY layer



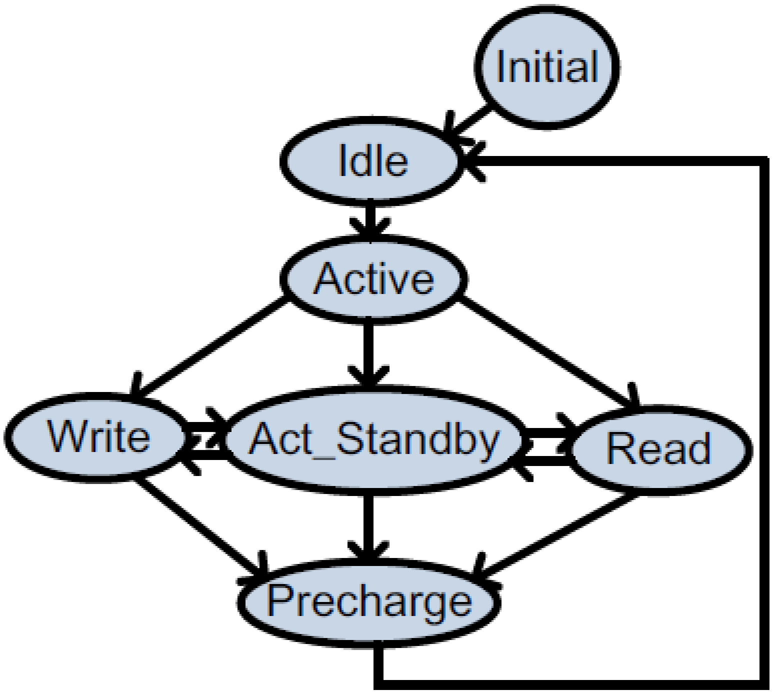
- Specially noted that ck and ck_n has a 90 degree phase shift, every signal trying to send to DRAM should have a 90 degree phase shift, thus uses negedge clk instead of posedge trigger



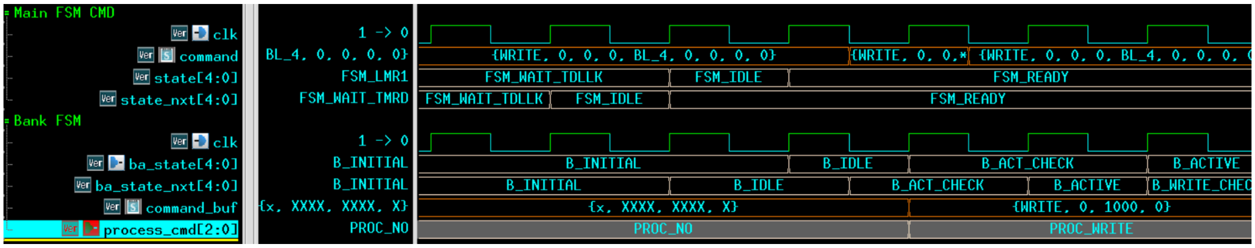
- Physical DDR domain uses ck_n posedge clk as reference for sampling

Write Command Procedure

- Notice that this procedure is different from what he described in his own paper, also what other people will do
- command → bankFSM→command Scheduler(This is where command gets Re-scheduled)→ issue_fifo



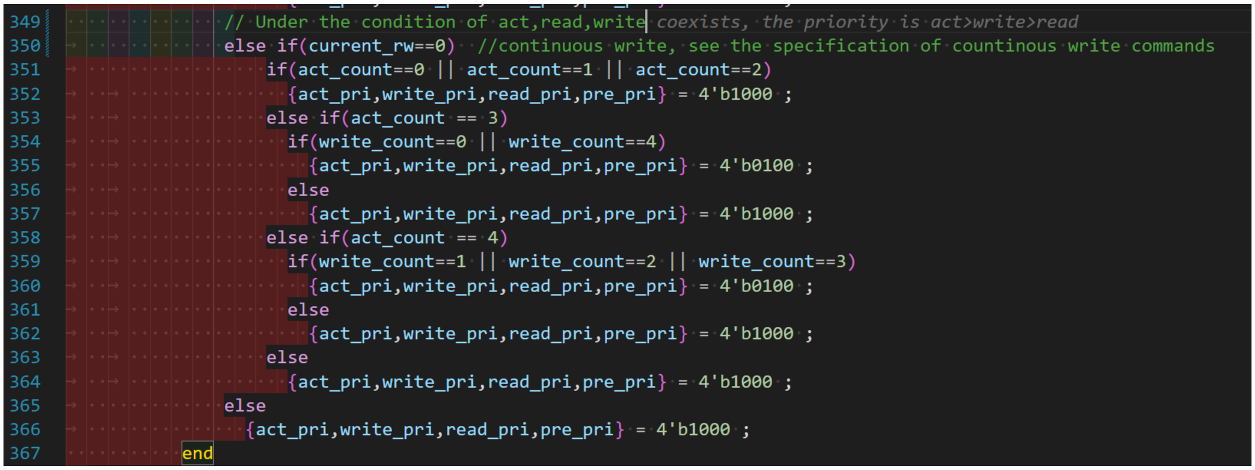
(a). Bank FSM



```
Main FSM CMD
                  clk   1 -> 0
              command   BL_4, 0, 0, 0, 0}   {WRITE, 0, 0, 0, BL_4, 0, 0, 0, 0}   {WRITE, 0, 0,*   {WRITE, 0, 0, 0, BL_4, 0, 0, 0, 0
           state[4:0]   FSM_LMR1            FSM_WAIT_TDLLK   FSM_IDLE   FSM_READY
       state_nxt[4:0]   FSM_WAIT_TMRD       FSM_WAIT_TDLLK   FSM_IDLE   FSM_READY
Bank FSM
                  clk   1 -> 0
        ba_state[4:0]   B_INITIAL           B_INITIAL   B_IDLE   B_ACT_CHECK   B_ACTIVE
    ba_state_nxt[4:0]   B_INITIAL           B_INITIAL   B_IDLE   B_ACT_CHECK   B_ACTIVE   B_WRITE_CHEC
          command_buf   {x, XXXX, XXXX, X}  {x, XXXX, XXXX, X}   {WRITE, 0, 1000, 0}
     process_cmd[2:0]   PROC_NO             PROC_NO   PROC_WRITE
```

- Note that for every bank fsm transition, there is a previous check state after that there is the act,write or pre command,

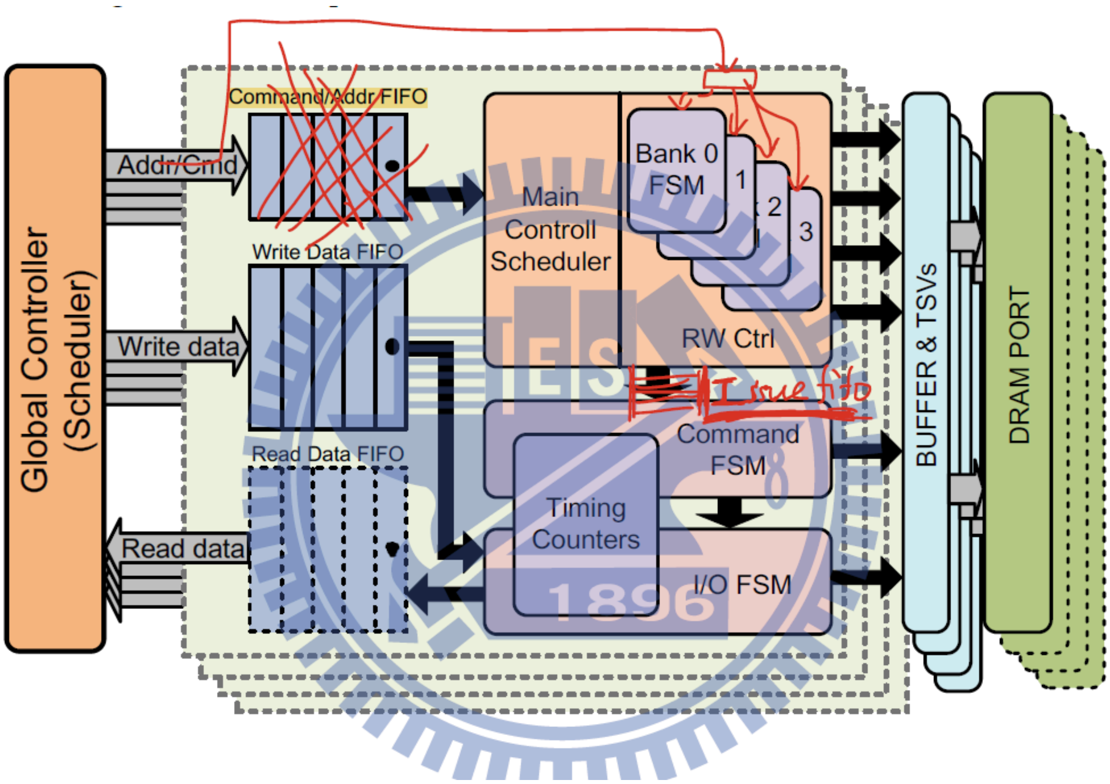
CMD scheduler scheduling priority

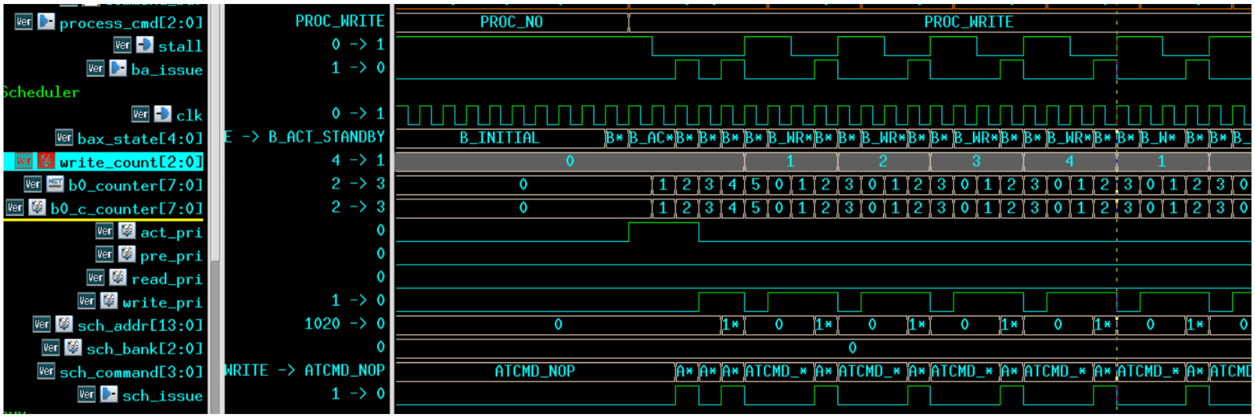


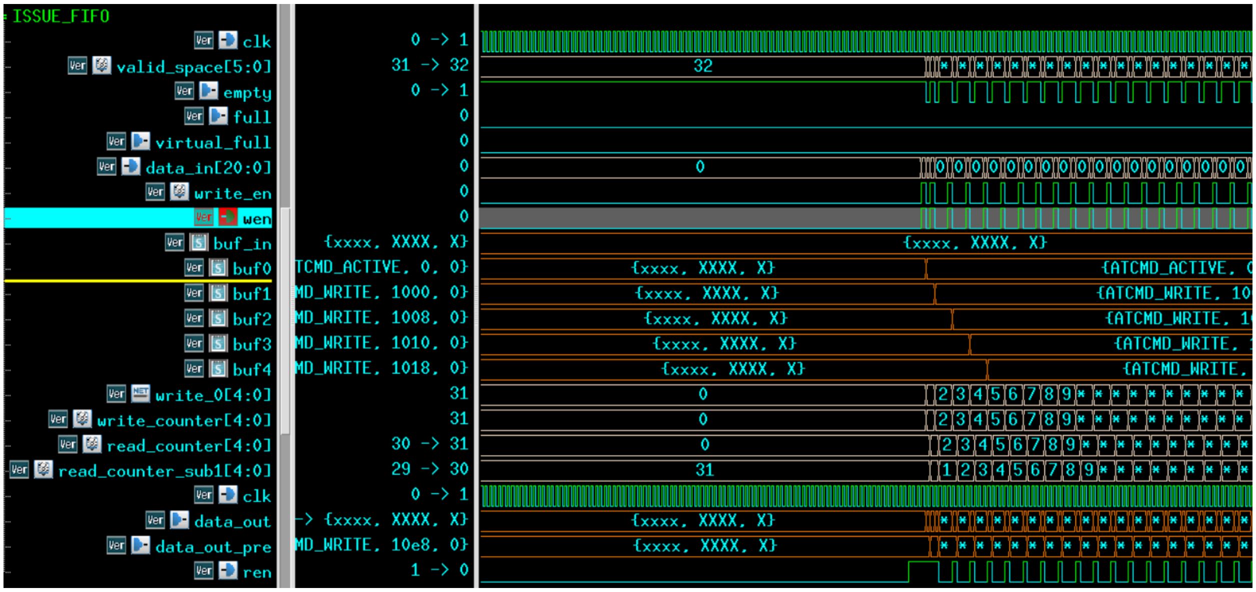
```verilog
  // Under the condition of act,read,write coexists, the priority is act>write>read
  else if(current_rw==0)  //continuous write, see the specification of countinous write commands
    if(act_count==0 || act_count==1 || act_count==2)
     {act_pri,write_pri,read_pri,pre_pri} = 4'b1000 ;
    else if(act_count == 3)
     if(write_count==0 || write_count==4)
      {act_pri,write_pri,read_pri,pre_pri} = 4'b0100 ;
     else
      {act_pri,write_pri,read_pri,pre_pri} = 4'b1000 ;
    else if(act_count == 4)
     if(write_count==1 || write_count==2 || write_count==3)
      {act_pri,write_pri,read_pri,pre_pri} = 4'b0100 ;
     else
      {act_pri,write_pri,read_pri,pre_pri} = 4'b1000 ;
    else
     {act_pri,write_pri,read_pri,pre_pri} = 4'b1000 ;
  else
   {act_pri,write_pri,read_pri,pre_pri} = 4'b1000 ;
end
```

- Since multiple banks are competing for resources, only one bank can be granted the access. A priority table must be constructed from command dependencies to ensure correct operation.

Write Continuous values







- Issue fifo is a circular fifo which accepts the scheduled DRAM commands from the command scheduler
- Notice that this is different from the diagram from Senior's thesis, thus the full DRAM controller model should be redrawn

DDR Spec Write & READ Write turnaround timing constraints

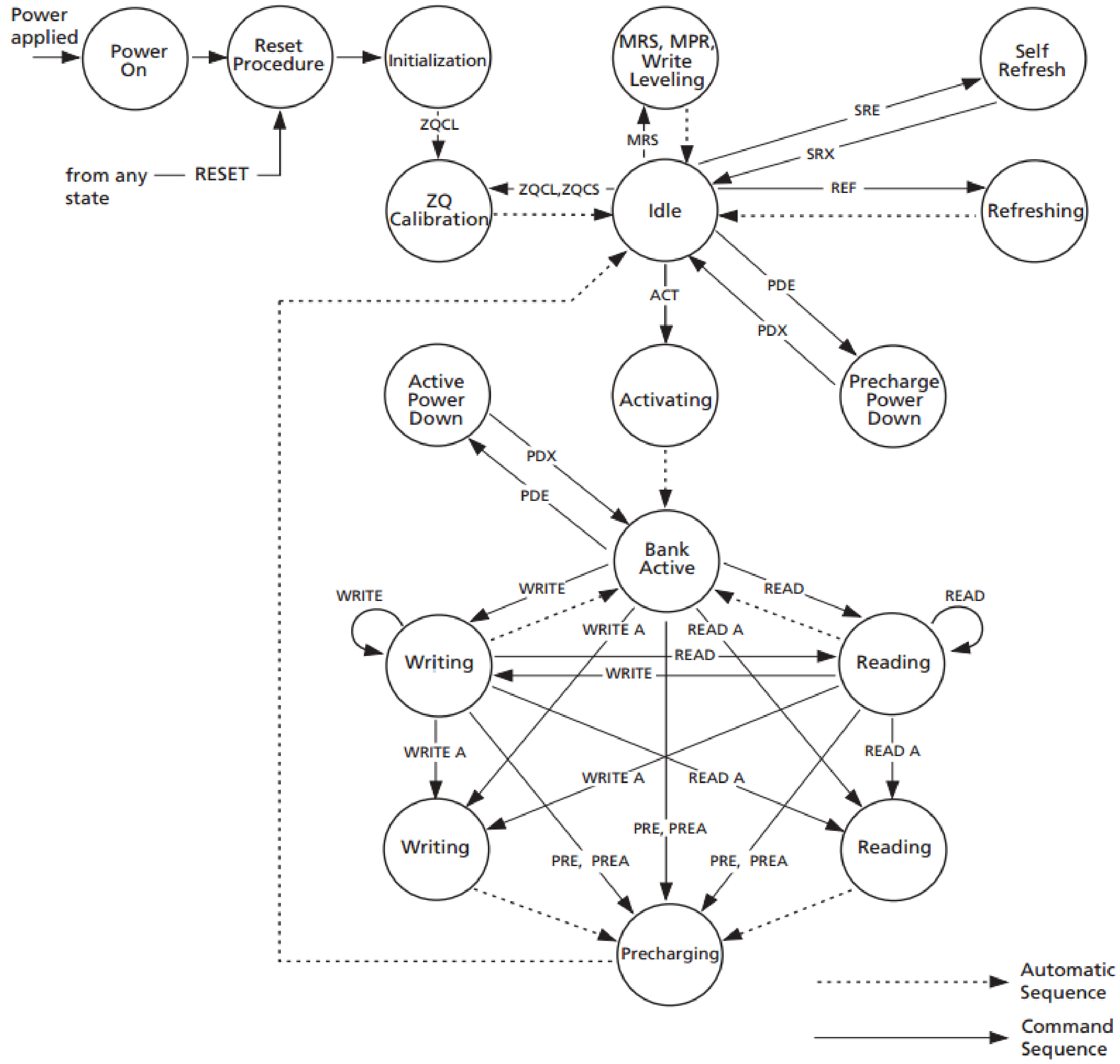


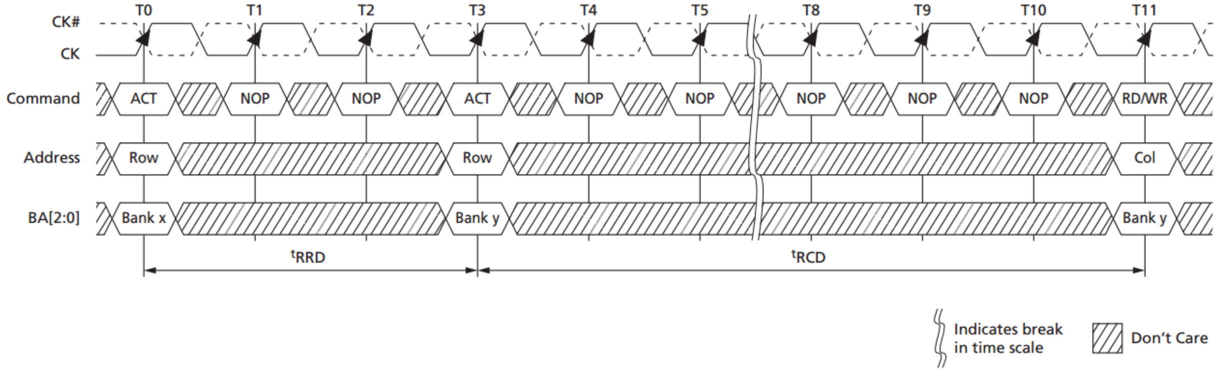
Figure 4 — Simplified State Diagram

- From the diagram we can know which timing constraints are needed

ACT

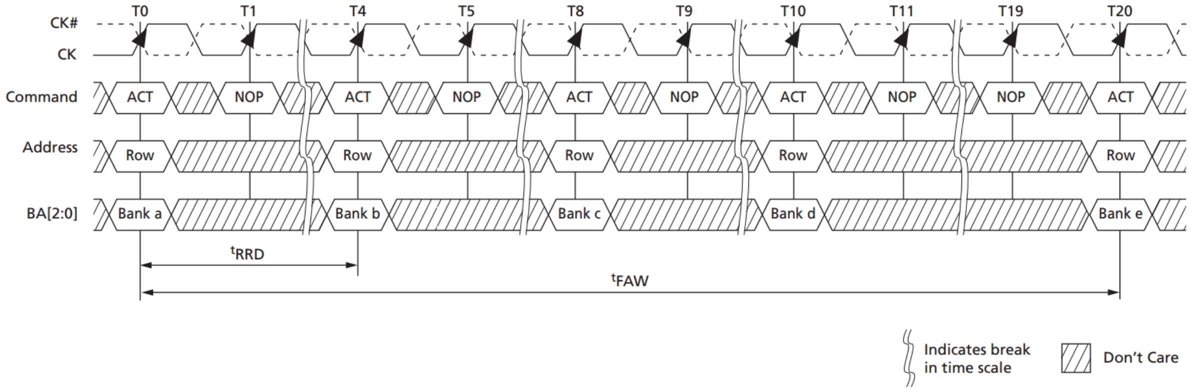
- Case of accessing different banks using ACT

Figure 59: Example: Meeting ^{t}RRD (MIN) and ^{t}RCD (MIN)



tFAW has to be held

Figure 60: Example: ^{t}FAW



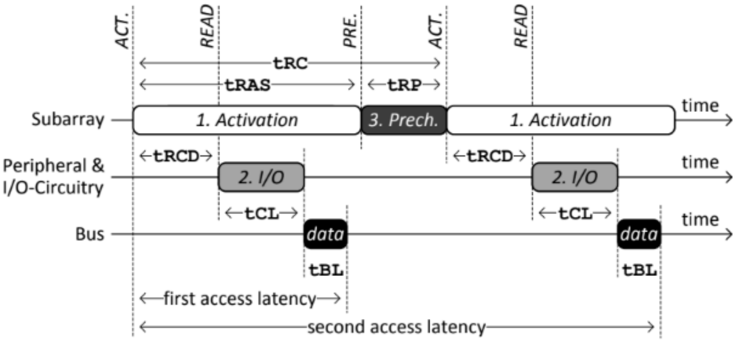
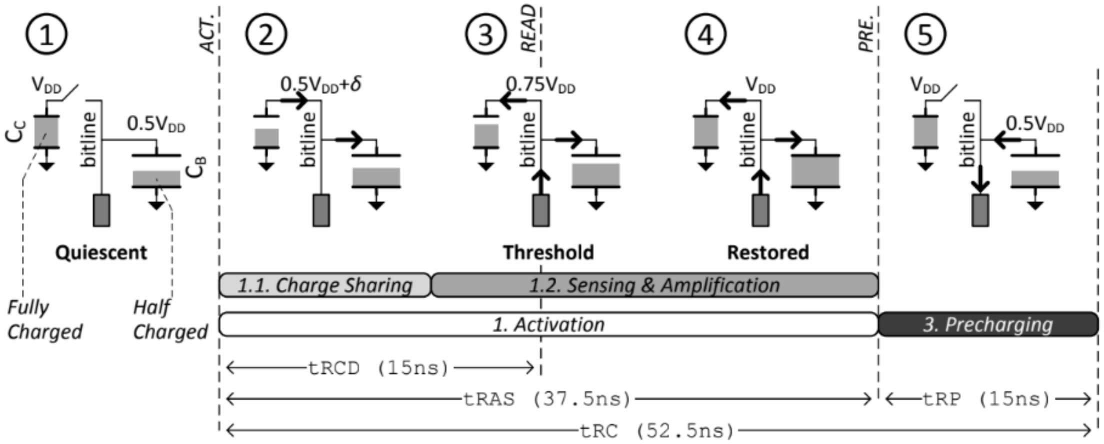


Figure 5. Three Phases of DRAM Access

Table 2. Timing Constraints (DDR3-1066) [43]

Phase	Commands	Name	Value
1	ACT → READ ACT → WRITE	tRCD	15ns
	ACT → PRE	tRAS	37.5ns
2	READ → *data* WRITE → *data*	tCL tCWL	15ns 11.25ns
	data burst	tBL	7.5ns
3	PRE → ACT	tRP	15ns
1 & 3	ACT → ACT	tRC (tRAS+tRP)	52.5ns



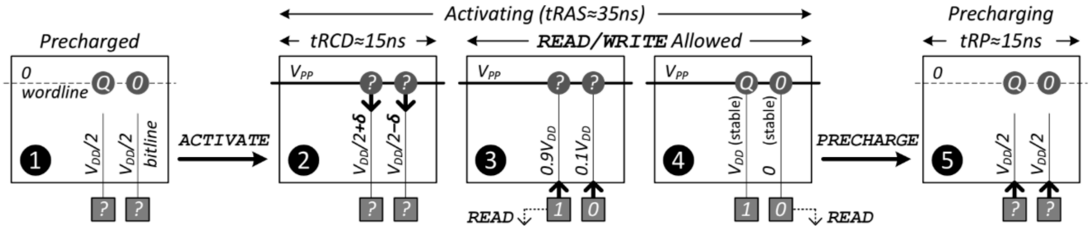
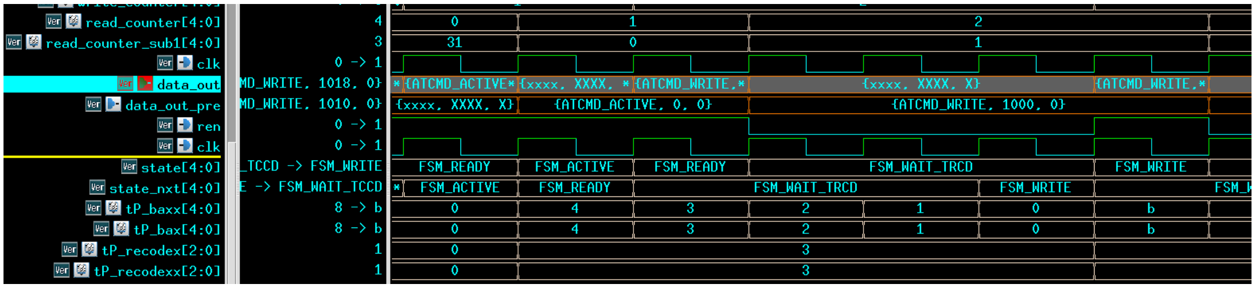


Figure 4. DRAM bank operation: Steps involved in serving a memory request [17] ($V_{PP} > V_{DD}$)

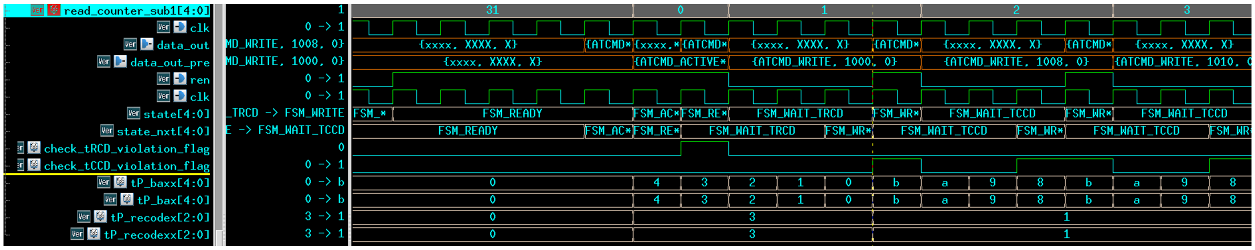
Category	RowCmd↔RowCmd			RowCmd↔ColCmd			ColCmd↔ColCmd			ColCmd→DATA	
Name	*tRC*	*tRAS*	*tRP*	*tRCD*	*tRTP*	*tWR**	*tCCD*	*tRTW*[†]	*tWTR**	*CL*	*CWL*
Commands	A→A	A→P	P→A	A→R/W	R→P	W*→P	R(W)→R(W)	R→W	W*→R	R→DATA	W→DATA
Scope	Bank	Bank	Bank	Bank	Bank	Bank	Channel	Rank	Rank	Bank	Bank
Value (ns)	**~50**	~35	13-15	13-15	~7.5	**15**	5-7.5	11-15	~7.5	13-15	10-15

A: ACTIVATE– P: PRECHARGE– R: READ– W: WRITE

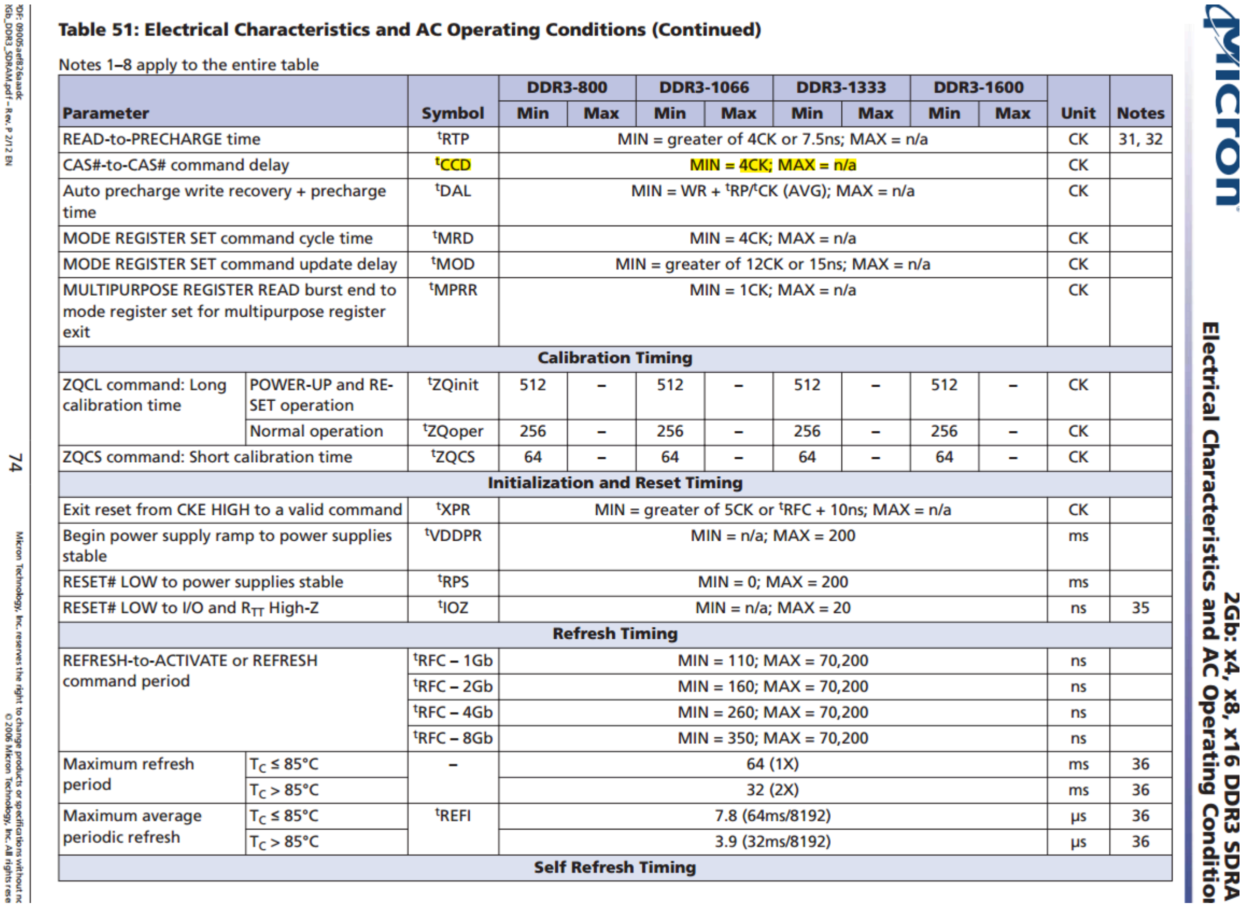
∗ Goes into effect after the last write *data*, not from the WRITE command


```
read_counter[4:0]        4                   0 | 1 | 2
read_counter_sub1[4:0]   3                   31 | 0 | 1
clk                      0 -> 1
data_out                 MD_WRITE, 1018, 0}  * {ATCMD_ACTIVE* {xxxx, XXXX, * {ATCMD_WRITE,* {xxxx, XXXX, X} {ATCMD_WRITE,*
data_out_pre             MD_WRITE, 1010, 0}  {xxxx, XXXX, X} | {ATCMD_ACTIVE, 0, 0} | {ATCMD_WRITE, 1000, 0}
ren                      0 -> 1
clk                      0 -> 1
state[4:0]               _TCCD -> FSM_WRITE  FSM_READY | FSM_ACTIVE | FSM_READY | FSM_WAIT_TRCD | FSM_WRITE
state_nxt[4:0]           E -> FSM_WAIT_TCCD  * FSM_ACTIVE | FSM_READY | FSM_WAIT_TRCD | FSM_WRITE | FSM_W
tP_baxx[4:0]             8 -> b              0 | 4 | 3 | 2 | 1 | 0 | b
tP_bax[4:0]              8 -> b              0 | 4 | 3 | 2 | 1 | 0 | b
tP_recodex[2:0]          1                   0 | 3
tP_recodexx[2:0]         1                   0 | 3
```

- For WRITE, first open the row, TRCD delays, later the row is opened
- After TRCD, row is opened, we can issue write, later consecutive writes requires TCCD
- Timing constraints related to write are tRCD,tCCD,tRTW, find their corresponding specification



- From the specification, check for tRCD and tCCD violation



PDF: 09005aef826aaadc
2Gb_DDR3_SDRAM.pdf – Rev. P 2/12 EN

2Gb: x4, x8, x16 DDR3 SDRA
Electrical Characteristics and AC Operating Conditio

Table 51: Electrical Characteristics and AC Operating Conditions (Continued)

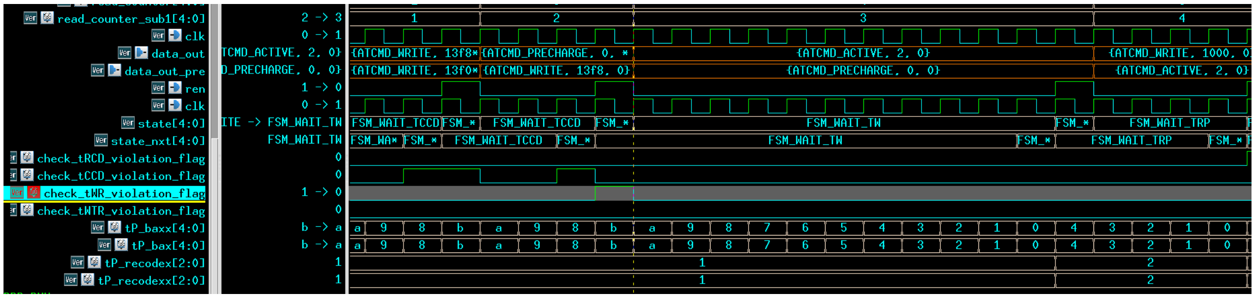
Notes 1–8 apply to the entire table

Parameter		Symbol	DDR3-800		DDR3-1066		DDR3-1333		DDR3-1600		Unit	Notes
			Min	Max	Min	Max	Min	Max	Min	Max		
READ-to-PRECHARGE time		ᵗRTP	MIN = greater of 4CK or 7.5ns; MAX = n/a								CK	31, 32
CAS#-to-CAS# command delay		ᵗCCD	MIN = 4CK; MAX = n/a								CK	
Auto precharge write recovery + precharge time		ᵗDAL	MIN = WR + ᵗRP/ᵗCK (AVG); MAX = n/a								CK	
MODE REGISTER SET command cycle time		ᵗMRD	MIN = 4CK; MAX = n/a								CK	
MODE REGISTER SET command update delay		ᵗMOD	MIN = greater of 12CK or 15ns; MAX = n/a								CK	
MULTIPURPOSE REGISTER READ burst end to mode register set for multipurpose register exit		ᵗMPRR	MIN = 1CK; MAX = n/a								CK	
Calibration Timing												
ZQCL command: Long calibration time	POWER-UP and RESET operation	ᵗZQinit	512	–	512	–	512	–	512	–	CK	
	Normal operation	ᵗZQoper	256	–	256	–	256	–	256	–	CK	
ZQCS command: Short calibration time		ᵗZQCS	64	–	64	–	64	–	64	–	CK	
Initialization and Reset Timing												
Exit reset from CKE HIGH to a valid command		ᵗXPR	MIN = greater of 5CK or ᵗRFC + 10ns; MAX = n/a								CK	
Begin power supply ramp to power supplies stable		ᵗVDDPR	MIN = n/a; MAX = 200								ms	
RESET# LOW to power supplies stable		ᵗRPS	MIN = 0; MAX = 200								ms	
RESET# LOW to I/O and R_{TT} High-Z		ᵗIOZ	MIN = n/a; MAX = 20								ns	35
Refresh Timing												
REFRESH-to-ACTIVATE or REFRESH command period		ᵗRFC – 1Gb	MIN = 110; MAX = 70,200								ns	
		ᵗRFC – 2Gb	MIN = 160; MAX = 70,200								ns	
		ᵗRFC – 4Gb	MIN = 260; MAX = 70,200								ns	
		ᵗRFC – 8Gb	MIN = 350; MAX = 70,200								ns	
Maximum refresh period	T_C ≤ 85°C	–	64 (1X)								ms	36
	T_C > 85°C		32 (2X)								ms	36
Maximum average periodic refresh	T_C ≤ 85°C	ᵗREFI	7.8 (64ms/8192)								µs	36
	T_C > 85°C		3.9 (32ms/8192)								µs	36
Self Refresh Timing												

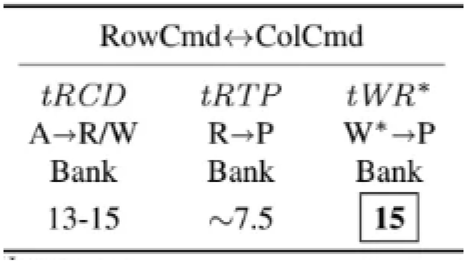
74

Micron Technology, Inc. reserves the right to change products or specifications without no
© 2006 Micron Technology, Inc. All rights rese

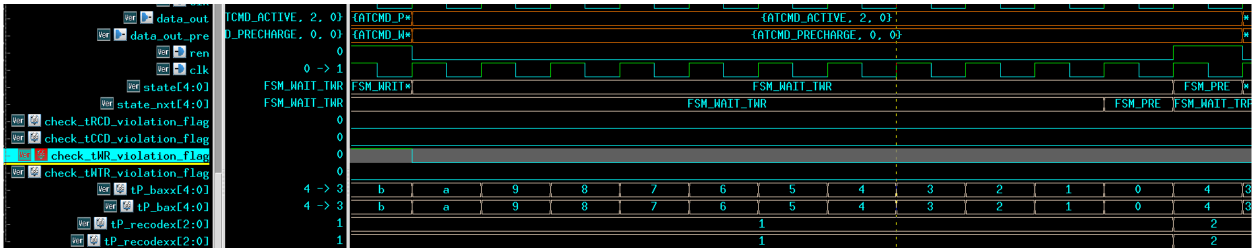
Row buffer conflicts



- Thus needs to wait tWR

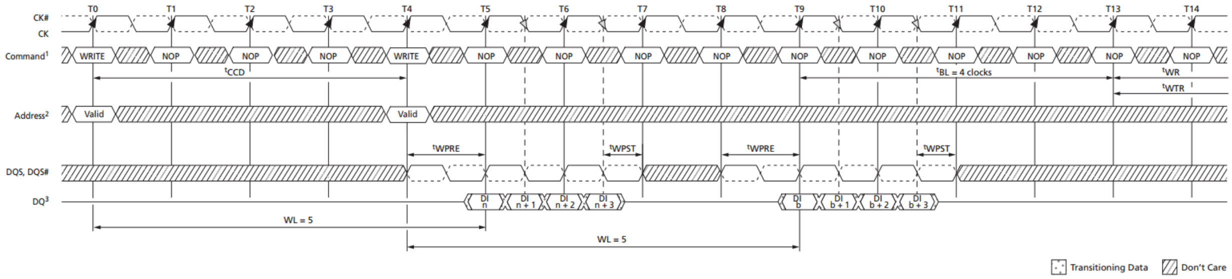


RowCmd↔ColCmd		
$tRCD$	$tRTP$	tWR^*
A→R/W	R→P	W*→P
Bank	Bank	Bank
13-15	~7.5	$\boxed{15}$



- tWR has to be waited before the precharge command

Figure 80: Consecutive WRITE (BC4) to WRITE (BC4) via OTF



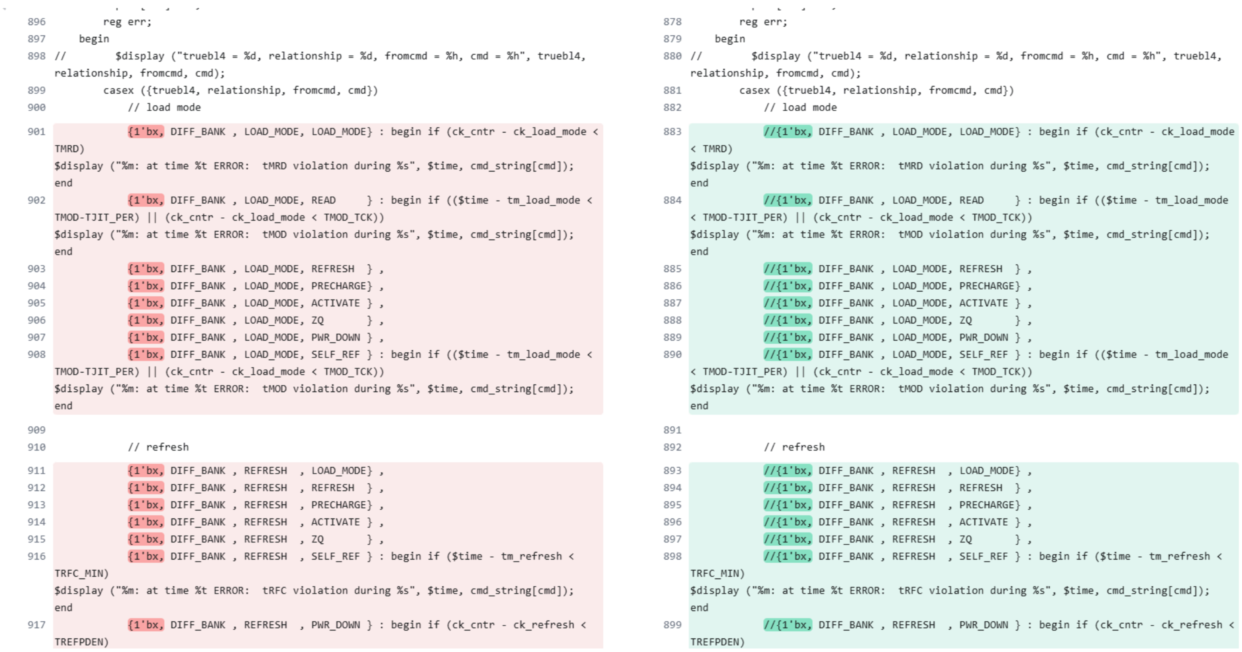
Notes:
1. NOP commands are shown for ease of illustration; other commands may be valid at these times.
2. BC4, WL = 5 (AL = 0, CWL = 5).
3. DI *n* (or *b*) = data-in for column *n* (or column *b*).
4. The BC4 setting is activated by MR0[1:0] = 01 and A12 = 0 during the WRITE command at T0 and T4.
5. If set via MRS (fixed) tWR and tWTR would start T11 (2 cycles earlier).

- d_state is used to control the WL delay, and since at the same time, due to the independence of Command line and DQ line, whenever a write signal is acquired, delay it multiple cycles then it can write onto the DQ bus

READ Burst tracing

Problems:

1. I cannot find the ADDR/RD WR FIFO, is there a newer version that I dont know?
2. How to obtain the temperature information of the DRAM chip and on-die temperature?DDR4 accomplish this through the usage of mode register.
3. According to DDR3 specification, there should be a 200us power up mode and 500us reset mode during the initializationand reset phase of the system, but I cannot find such configuration within the code, perhaps there is a newer version?
4. I have some problem with the Read/Write re-scheduling portion, since there is only one issue fifo , how can it be done?
5. I cannot find Refresh control or refresh state within Senior’s design, perhaps I have some misunderstanding about the specification of the DRAM?
6. Is the refresh controller implemented within the bank logic itself?
7. Within the 3D-DRAM, the timing violation checkers are all commented out~ So is there another model for timing constraint checking?



```
        reg err;
    begin
//        $display ("truebl4 = %d, relationship = %d, fromcmd = %h, cmd = %h", truebl4,
relationship, fromcmd, cmd);
        casex ({truebl4, relationship, fromcmd, cmd})
            // load mode
            {1'bx, DIFF_BANK , LOAD_MODE, LOAD_MODE} : begin if (ck_cntr - ck_load_mode <
TMRD)
$display ("%m: at time %t ERROR:  tMRD violation during %s", $time, cmd_string[cmd]);
end
            {1'bx, DIFF_BANK , LOAD_MODE, READ     } : begin if (($time - tm_load_mode <
TMOD-TJIT_PER) || (ck_cntr - ck_load_mode < TMOD_TCK))
$display ("%m: at time %t ERROR:  tMOD violation during %s", $time, cmd_string[cmd]);
end
            {1'bx, DIFF_BANK , LOAD_MODE, REFRESH  } ,
            {1'bx, DIFF_BANK , LOAD_MODE, PRECHARGE} ,
            {1'bx, DIFF_BANK , LOAD_MODE, ACTIVATE } ,
            {1'bx, DIFF_BANK , LOAD_MODE, ZQ       } ,
            {1'bx, DIFF_BANK , LOAD_MODE, PWR_DOWN } ,
            {1'bx, DIFF_BANK , LOAD_MODE, SELF_REF } : begin if (($time - tm_load_mode <
TMOD-TJIT_PER) || (ck_cntr - ck_load_mode < TMOD_TCK))
$display ("%m: at time %t ERROR:  tMOD violation during %s", $time, cmd_string[cmd]);
end

            // refresh
            {1'bx, DIFF_BANK , REFRESH  , LOAD_MODE} ,
            {1'bx, DIFF_BANK , REFRESH  , REFRESH  } ,
            {1'bx, DIFF_BANK , REFRESH  , PRECHARGE} ,
            {1'bx, DIFF_BANK , REFRESH  , ACTIVATE } ,
            {1'bx, DIFF_BANK , REFRESH  , ZQ       } ,
            {1'bx, DIFF_BANK , REFRESH  , SELF_REF } : begin if ($time - tm_refresh <
TRFC_MIN)
$display ("%m: at time %t ERROR:  tRFC violation during %s", $time, cmd_string[cmd]);
end
            {1'bx, DIFF_BANK , REFRESH  , PWR_DOWN } : begin if (ck_cntr - ck_refresh <
TREFPDEN)
```

```
        reg err;
    begin
//        $display ("truebl4 = %d, relationship = %d, fromcmd = %h, cmd = %h", truebl4,
relationship, fromcmd, cmd);
        casex ({truebl4, relationship, fromcmd, cmd})
            // load mode
            //{1'bx, DIFF_BANK , LOAD_MODE, LOAD_MODE} : begin if (ck_cntr - ck_load_mode
< TMRD)
$display ("%m: at time %t ERROR:  tMRD violation during %s", $time, cmd_string[cmd]);
end
            //{1'bx, DIFF_BANK , LOAD_MODE, READ     } : begin if (($time - tm_load_mode
< TMOD-TJIT_PER) || (ck_cntr - ck_load_mode < TMOD_TCK))
$display ("%m: at time %t ERROR:  tMOD violation during %s", $time, cmd_string[cmd]);
end
            //{1'bx, DIFF_BANK , LOAD_MODE, REFRESH  } ,
            //{1'bx, DIFF_BANK , LOAD_MODE, PRECHARGE} ,
            //{1'bx, DIFF_BANK , LOAD_MODE, ACTIVATE } ,
            //{1'bx, DIFF_BANK , LOAD_MODE, ZQ       } ,
            //{1'bx, DIFF_BANK , LOAD_MODE, PWR_DOWN } ,
            //{1'bx, DIFF_BANK , LOAD_MODE, SELF_REF } : begin if (($time - tm_load_mode
< TMOD-TJIT_PER) || (ck_cntr - ck_load_mode < TMOD_TCK))
$display ("%m: at time %t ERROR:  tMOD violation during %s", $time, cmd_string[cmd]);
end

            // refresh
            //{1'bx, DIFF_BANK , REFRESH  , LOAD_MODE} ,
            //{1'bx, DIFF_BANK , REFRESH  , REFRESH  } ,
            //{1'bx, DIFF_BANK , REFRESH  , PRECHARGE} ,
            //{1'bx, DIFF_BANK , REFRESH  , ACTIVATE } ,
            //{1'bx, DIFF_BANK , REFRESH  , ZQ       } ,
            //{1'bx, DIFF_BANK , REFRESH  , SELF_REF } : begin if ($time - tm_refresh <
TRFC_MIN)
$display ("%m: at time %t ERROR:  tRFC violation during %s", $time, cmd_string[cmd]);
end
            //{1'bx, DIFF_BANK , REFRESH  , PWR_DOWN } : begin if (ck_cntr - ck_refresh <
TREFPDEN)
```

- Or there are probably some assumptions about how this work is done