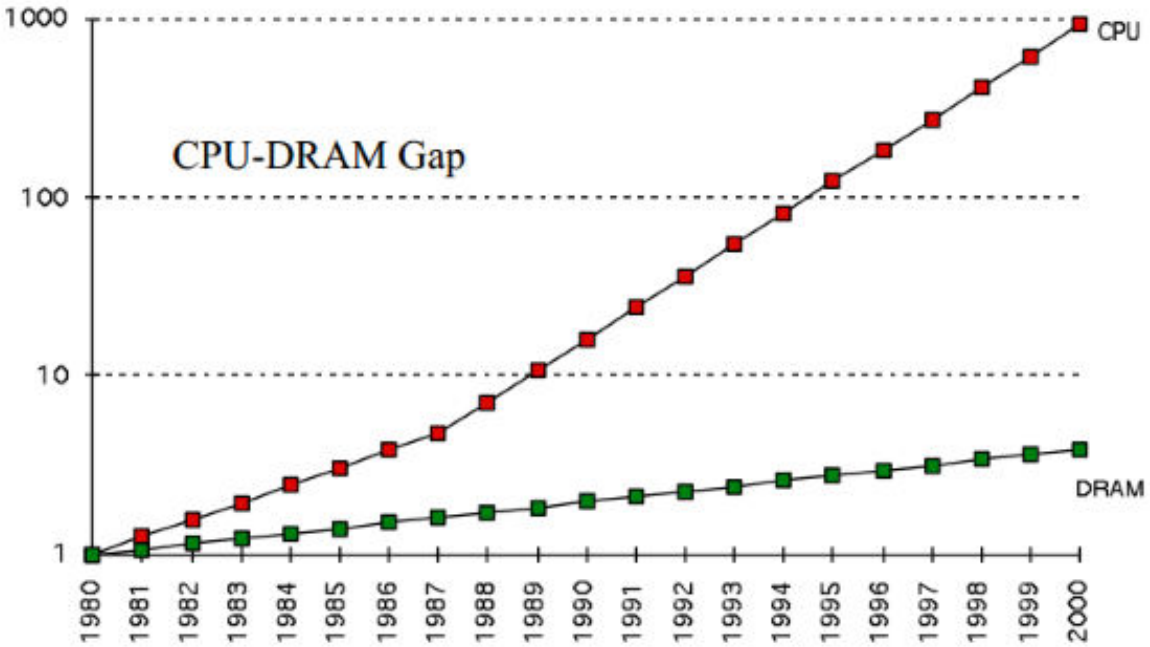
Synchronous Dram (SDRAM)

- As we saw in the last lecture, a huge gulf had opened up between the Multi-GHz speeds of the fastest CPUs and the slow speed of conventional Async Drams. The graph below shows that CPU performance has doubled every **18 months**, but dram performance has only doubled every **10 years.**
- Fast Page Mode Drams and on-chip caches helped, but dram latency has proved to be a real bottleneck to processor performance. Something had to change.

- Processor vs Memory Performance



Synchronous Dram (SDRAM)

- To increase performance, a new generation of Synchronous Drams were developed that could 'burst' multiple items of data from successive addresses (i.e. the contents of a dram row buffer) onto the data bus using a high speed clock synchronized to the CPUs memory controller. Bursting would help to fill the CPU cache must faster.

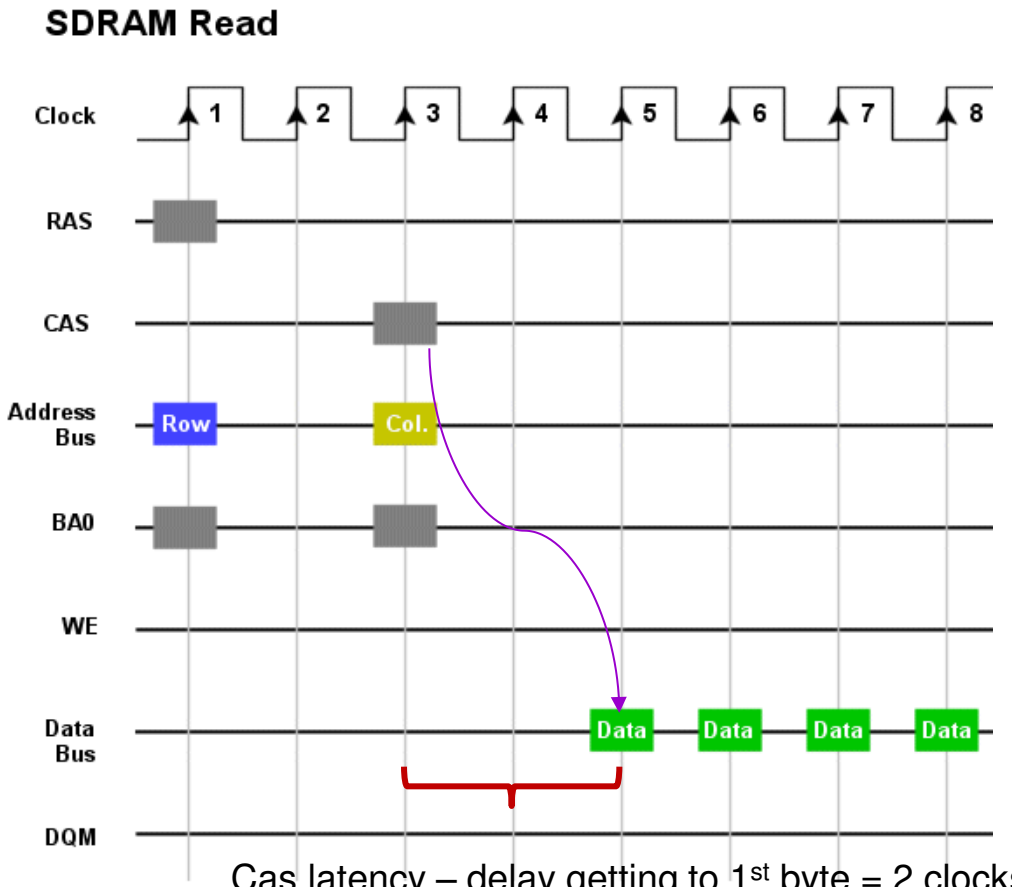
- For example, a dram with a 1024 byte row buffer could have it's row contents clocked out at say 1Ghz like a shift register.

- Pipelining improves this even more by allowing a sophisticated dram controller to issue new commands to a Synchronous Dram memory chip before the memory chip has finished dealing with the last command, i.e. it allowed for read/write requests to overlap in an attempt to keep the data transfer flowing

- To take advantage of this new technology required a new generation of more sophisticated Dram Controllers.

Synchronous Dram (SDRAM)

- Simplified Synchronous Dram timing diagram.
- After presentation of a row and column address, data is pumped or "**burst**" out of the Dram (*via a shift register*) under the control of a high speed clock.
- The CPU then grabs the data with each clock cycle and transfers it to the



Cas latency – delay getting to 1st byte = 2 clocks

Note that no new addresses are needed after the initial row and column address have been presented

A burst of 4 bytes of data. At 1GHz this equates to a new byte every 1nS vs 40ns with Fast Page mode dram. This is equal to 1GByte per second

Synchronous Dram (SDRAM)

Synchronous Dram Commands

- Unlike Asynchronous Dram, the new generation of SDrams require that a command is issued with each POSITIVE edge of the clock indicating the type of operation to performed, such as activate, read, write, stop burst, refresh etc.

  These commands are entered via the CS*, WE*, RAS* and CAS* signals on the Sdram in conjunction with an address/bank value.

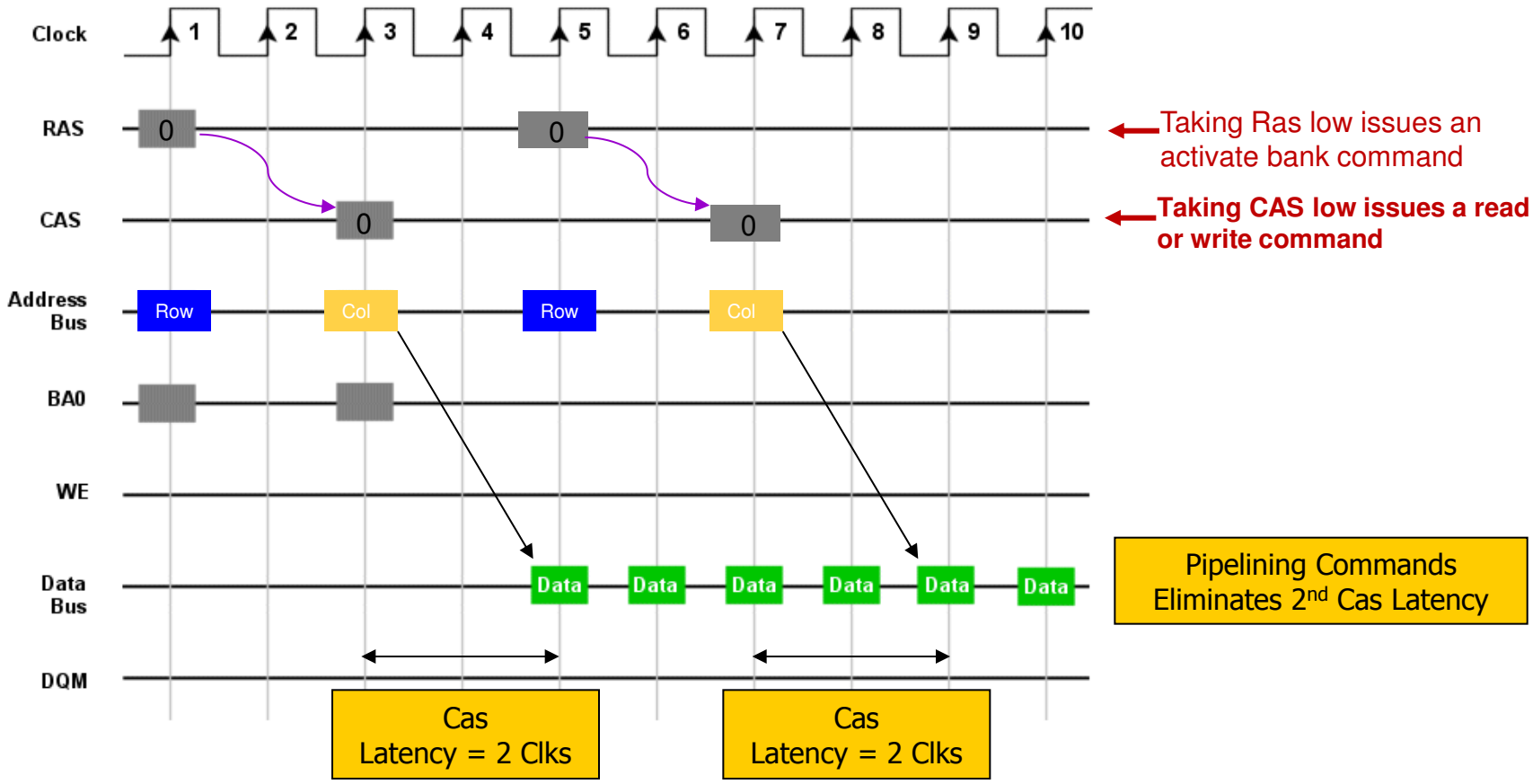
- Unless you intend to issue a new command, you must present a NOP command with each clock

NAME (FUNCTION)	CS#	RAS#	CAS#	WE#	ADDR
DESELECT (NOP)	H	X	X	X	X
NO OPERATION (NOP)	L	H	H	H	X
ACTIVE (Select bank and activate row)	L	L	H	H	Bank/Row
READ (Select bank and column, and start READ burst)	L	H	L	H	Bank/Col
WRITE (Select bank and column, and start WRITE burst)	L	H	L	L	Bank/Col
BURST TERMINATE	L	H	H	L	X
PRECHARGE (Deactivate row in bank or banks)	L	L	H	L	Code
AUTO REFRESH or SELF REFRESH (Enter self refresh mode)	L	L	L	H	X
LOAD MODE REGISTER	L	L	L	L	Op-Code

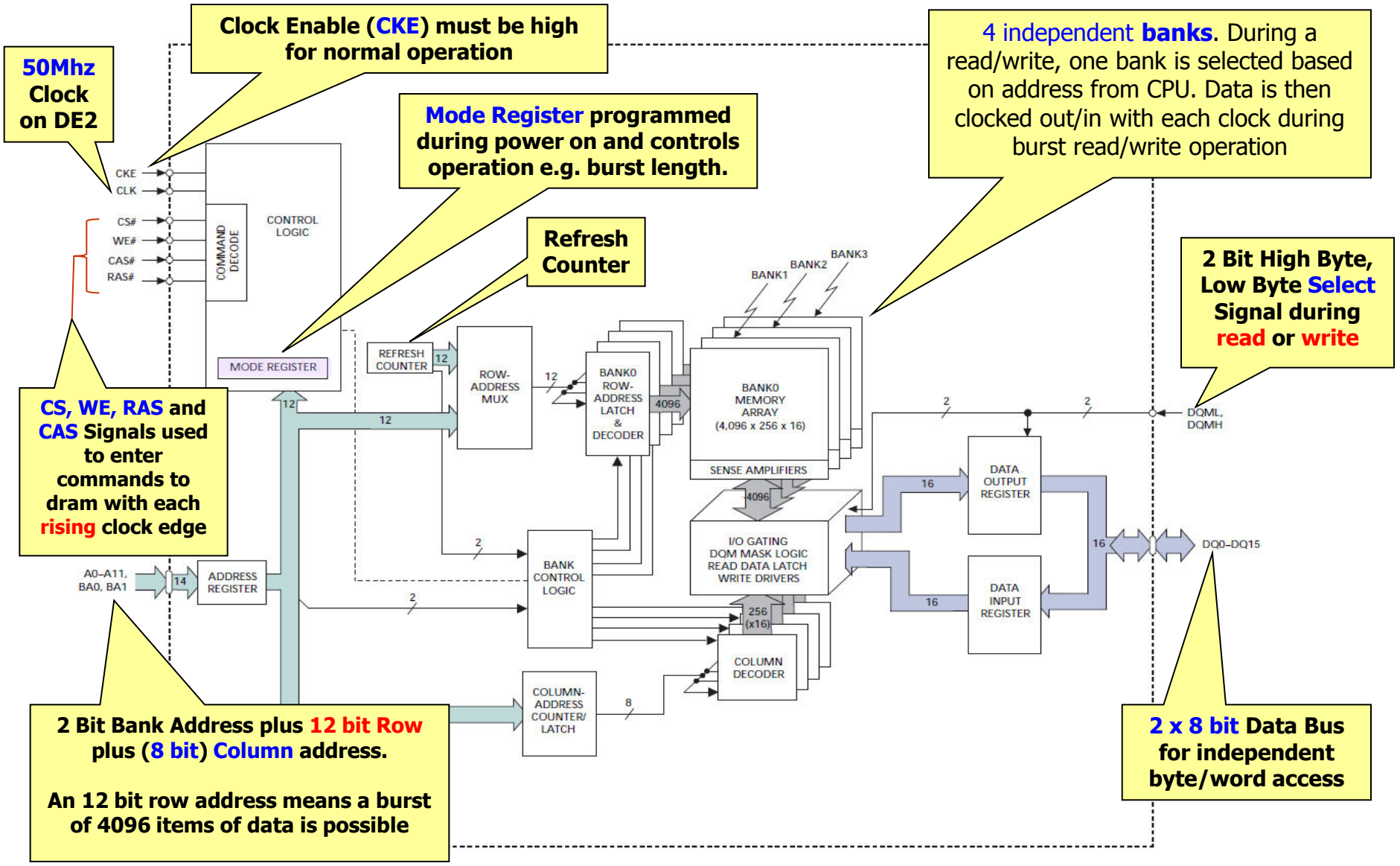
Synchronous Dram (SDRAM)

Pipelining Commands

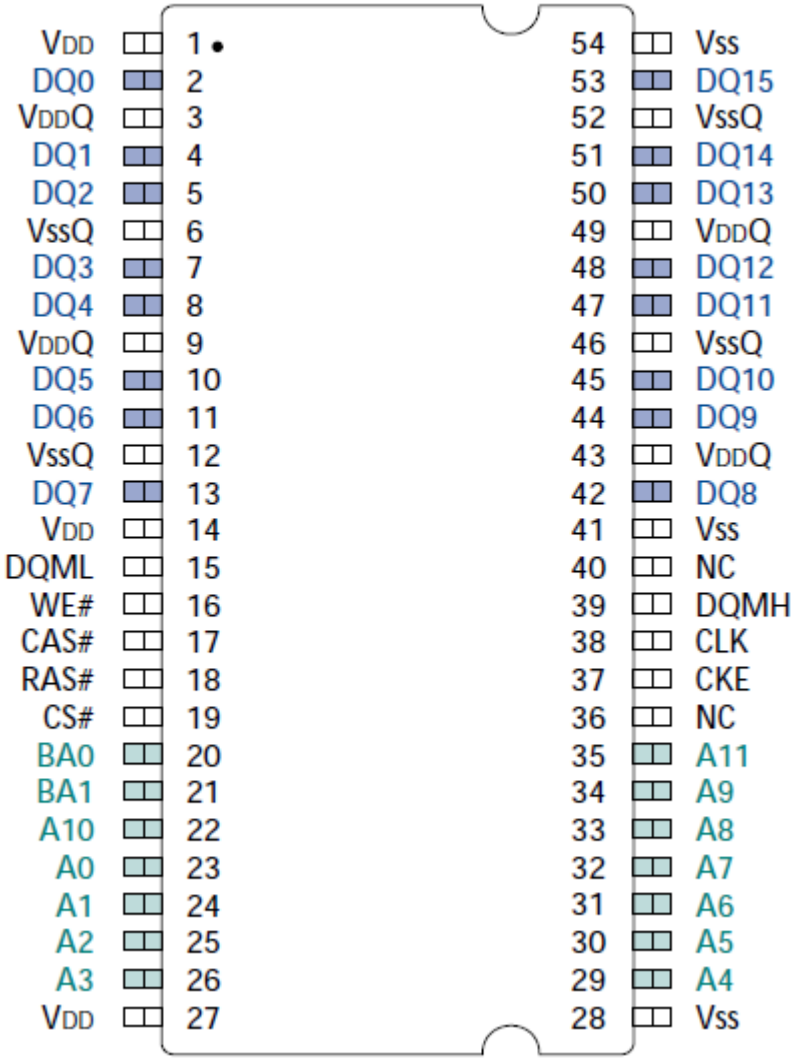
- Pipelining allows a a dram controller to issue new dram commands before the previous one has completed, this allowing continuous data transfer without incurring subsequent CAS latencies. This is shown below where two consecutive bursts of 4 bytes are requested and occur back to back.



Typical 4M * 16 bit SDram Architecture (as per DE2)



Typical 4M x 16bit SDram Pin Out



Access time from CAS is a fixed delay e.g. 5.4ns as shown below. With higher clock speeds (e.g. 143Mhz, we need to wait more clock periods after CAS (the **CAS Latency**) to build up the 5.4ns delay before we can get at the 1st item of data, but access to subsequent data is faster overall .

KEY TIMING PARAMETERS

SPEED GRADE	CLOCK FREQUENCY	ACCESS TIME		SETUP TIME	HOLD TIME
		CL = 2*	CL = 3*		
-7E	143 MHz	–	5.4ns	1.5ns	0.8ns
-75	133 MHz	–	5.4ns	1.5ns	0.8ns
-7E	133 MHz	5.4ns	–	1.5ns	0.8ns
-75	100 MHz	6ns	–	1.5ns	0.8ns

12 Bit Address Bus: A0-A11
16 Bit Data Bus: DQ0-DQ15
2 Bit Bank Address: BA0-BA1
2 Byte Select signals: DQMH/DQML

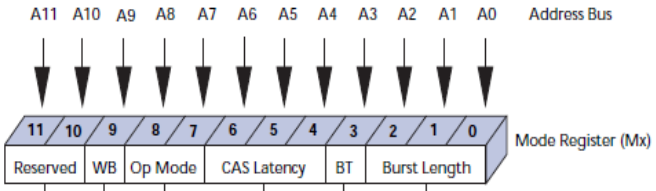
(The Byte select signals are used to activate one or other or both halves of the data bus. This mirrors the operation of the 68k's UDS and LDS* signals)*

SDram Chip Commands

- An SDram chip first has to be programmed before use with a Load Mode Register command.
- Commands are presented to the SDram chip by the SDram controller and sampled on each rising edge of the clock.
- Commands are interpreted by the signal levels on CS#, RAS#, CAS# and WE# (*see below*).
- Activate commands are issued to activate a Bank within the chip while at the same time presenting a ROW address to that bank (*this is analogous to the old asynchronous dram act of issuing RAS*).
- Read/Write commands require a bank and a column address which, when combined with the previous activate command uniquely identifies bank/row and column location within the chip (*this is analogous to the old asynchronous dram act of issuing CAS*).
- Pre-charging closes a bank/row in preparation for opening another with an activate command.
- No Operation (NOP) corresponds to a do noting or “carry on as before”.
- Auto Refresh operations can be initiated by the SDram Controller.

NAME (FUNCTION)	CS#	RAS#	CAS#	WE#	ADDR
DESELECT (NOP)	H	X	X	X	X
NO OPERATION (NOP)	L	H	H	H	X
ACTIVE (Select bank and activate row)	L	L	H	H	Bank/Row
READ (Select bank and column, and start READ burst)	L	H	L	H	Bank/Col
WRITE (Select bank and column, and start WRITE burst)	L	H	L	L	Bank/Col
BURST TERMINATE	L	H	H	L	X
PRECHARGE (Deactivate row in bank or banks)	L	L	H	L	Code
AUTO REFRESH or SELF REFRESH (Enter self refresh mode)	L	L	L	H	X
LOAD MODE REGISTER	L	L	L	L	Op-Code

Mode register programmed with data on the lowest 12 bits of Address Bus: A11-A0



Program BA0, BA1, M11, M10 = "0, 0" to ensure compatibility with future devices.

M9	Write Burst Mode
0	Programmed Burst Length
1	Single Location Access

M8	M7	M6–M0	Operating Mode
0	0	Defined	Standard Operation
–	–	–	All other states reserved

	Burst Length	
M2 M1 M0	3 = 0	M3 = 1
0 0 0	1	1
0 0 1	2	2
0 1 0	4	4
0 1 1	8	8
1 0 0	Reserved	Reserved
1 0 1	Reserved	Reserved
1 1 0	Reserved	Reserved
1 1 1	Full Page	Reserved

M3	Burst Type
0	Sequential
1	Interleaved

M6 M5 M4	CAS Latency
0 0 0	Reserved
0 0 1	Reserved
0 1 0	2
0 1 1	3
1 0 0	Reserved
1 0 1	Reserved
1 1 0	Reserved
1 1 1	Reserved

Burst Length and type depends upon the chip and how it is programmed.

Burst Length	Starting Column Address	Order of Accesses Within a Burst	
		Type = Sequential	Type = Interleaved
2	A0		
	0	0-1	0-1
	1	1-0	1-0
4	A1 A0		
	0 0	0-1-2-3	0-1-2-3
	0 1	1-2-3-0	1-0-3-2
	1 0	2-3-0-1	2-3-0-1
	1 1	3-0-1-2	3-2-1-0
8	A2 A1 A0		
	0 0 0	0-1-2-3-4-5-6-7	0-1-2-3-4-5-6-7
	0 0 1	1-2-3-4-5-6-7-0	1-0-3-2-5-4-7-6
	0 1 0	2-3-4-5-6-7-0-1	2-3-0-1-6-7-4-5
	0 1 1	3-4-5-6-7-0-1-2	3-2-1-0-7-6-5-4
	1 0 0	4-5-6-7-0-1-2-3	4-5-6-7-0-1-2-3
	1 0 1	5-6-7-0-1-2-3-4	5-4-7-6-1-0-3-2
	1 1 0	6-7-0-1-2-3-4-5	6-7-4-5-2-3-0-1
	1 1 1	7-0-1-2-3-4-5-6	7-6-5-4-3-2-1-0
Full Page (y)	n = A0-A12/11/9 (location 0-y)	Cn, Cn + 1, Cn + 2 Cn + 3, Cn + 4… …Cn - 1, Cn…	Not Supported

Interleaved vs. Sequential Access

Burst Length	Starting Column Address	Order of Accesses Within a Burst	
		Type = Sequential	Type = Interleaved
2	A0		
	0	0-1	0-1
	1	1-0	1-0
4	A1 A0		
	0 0	0-1-2-3	0-1-2-3
	0 1	1-2-3-0	[illegible]
	1 0	2-3-0-1	2-3-0-1
	1 1	3-0-1-2	3-2-1-0
8	A2 A1 A0		
	0 0 0	0-1-2-3-4-5-6-7	0-1-2-3-4-5-6-7
	0 0 1	1-2-3-4-5-6-7-0	1-0-3-2-5-4-7-6
	0 1 0	2-3-4-5-6-7-0-1	2-3-0-1-6-7-4-5
	0 1 1	3-4-5-6-7-0-1-2	3-2-1-0-7-6-5-4
	1 0 0	4-5-6-7-0-1-2-3	4-5-6-7-0-1-2-3
	1 0 1	5-6-7-0-1-2-3-4	5-4-7-6-1-0-3-2
	1 1 0	6-7-0-1-2-3-4-5	6-7-4-5-2-3-0-1
	1 1 1	7-0-1-2-3-4-5-6	7-6-5-4-3-2-1-0
Full Page (y)	n = A0-A12/11/9 (location 0-y)	Cn, Cn + 1, Cn + 2 Cn + 3, Cn + 4... ...Cn - 1, Cn...	Not Supported

The choice of Interleaved or sequential access depends on the type of cache controller on your CPU.

Sequential access is easy to understand. The CPU outputs a (column) address for say a read, and the sdram responds with the data at that address followed by the data from successive address locations up to the programmed burst length in the Mode register, cycling back to the first item where necessary.

For example, a read from location 3 with a burst length of 8 gives the contents of location 3, 4, 5, 6, 7 before wrapping back to 0, 1 and finally 2.

Interleaved access is very difficult to comprehend at first sight, i.e. what order do we get the data out of the sdram for a particular start/column address and a programmed burst length? It starts off being the same as Sequential, i.e. the address the CPU issues is always the location returned first. However, the data from subsequent locations is determined by EX-ORing the Column address bits from the CPU with the internal Sdram burst counter.

For example take a column address (A2,A1,A0) of 101 and a burst length of 8. If we EX-OR 101 with 000, then 001, 010, 011,100,101,110, and finally 111 we get, 5-4-7-6-1-0-3-2. Intel CPU cache controllers prefer this mode that's why it exists.

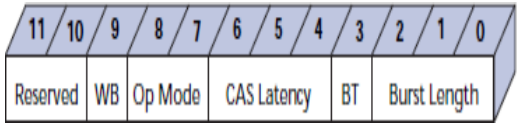
Programming the Mode Register

- Programming the Mode Register is performed a.s.a.p. after power on by the SDram controller, before the CPU accesses the memory for the first time.

- The Zentel SDram chip on the DE2 board can have a clock latency of 2 clock periods for a clock running at a relatively slow 50Mhz.

- Because the 68k does not have a cache and therefore does not issue or deal with burst requests, read and write burst lengths should be set to 1 (the choice of *sequential or interleaved access therefore becomes irrelevant*).

- The Zentel / DE2 chip has 12 address lines so the mode register is set with the following data on address lines A11..A0 = 001000100000. i.e. hex 220



- This configures the chip as follows:-

  - Read/Write burst length = 1 location,
  - Sequential access,
  - Cas latency = 2 clock periods

- To Load the Mode register we present the above value on the sdram address lines, set BA0 and BA1 to be "00" and take CS=0, WE=0, RAS=0 and CAS=0 (*see page 7*).

Power On Initialisation

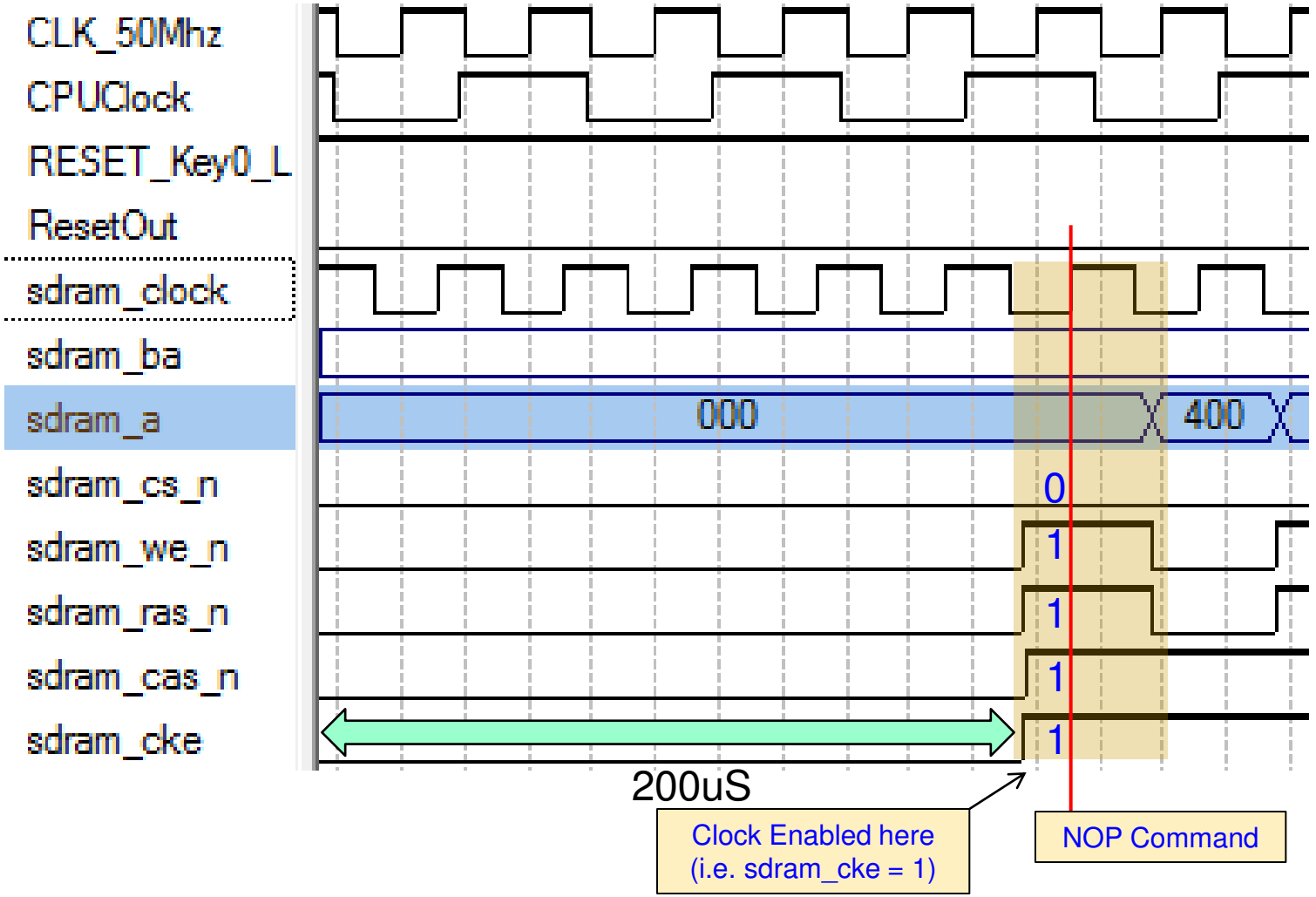
- The power on and initialisation sequence for the ZENTEL SDram is crucial.
- The data sheets say, or imply, from timing diagrams the following sequence.

1. Apply power and a stable clock but maintain the Sdram Clock Enable (CKE) signal in the INACTIVE (low) state for 200us. *In practice the 200us is not essential due to the long power up phase of FPGAs.*
2. After the 200us period, take CKE high (active) and commence with at least 1 NOP command to the SDRam.
3. Next issue a Pre-charge command for all banks of the devices. This means issuing a pre-charge command while at the same time setting address line A10 = 1 (as per Page 7).
4. Issue 1 NOP command after above Pre-charge command
5. Next issue 8 or more Auto-Refresh commands (Page 7) **with** 3 NOP commands between each Auto-refresh.
6. Next program device operation with a Load Mode Register Command.
7. Finally issue 3 NOP commands.
8. The device is then ready for normal use, i.e. read/write operation.

- A simulation of the power on initialisation for an SDram is shown over the page, for the working version of the DE2

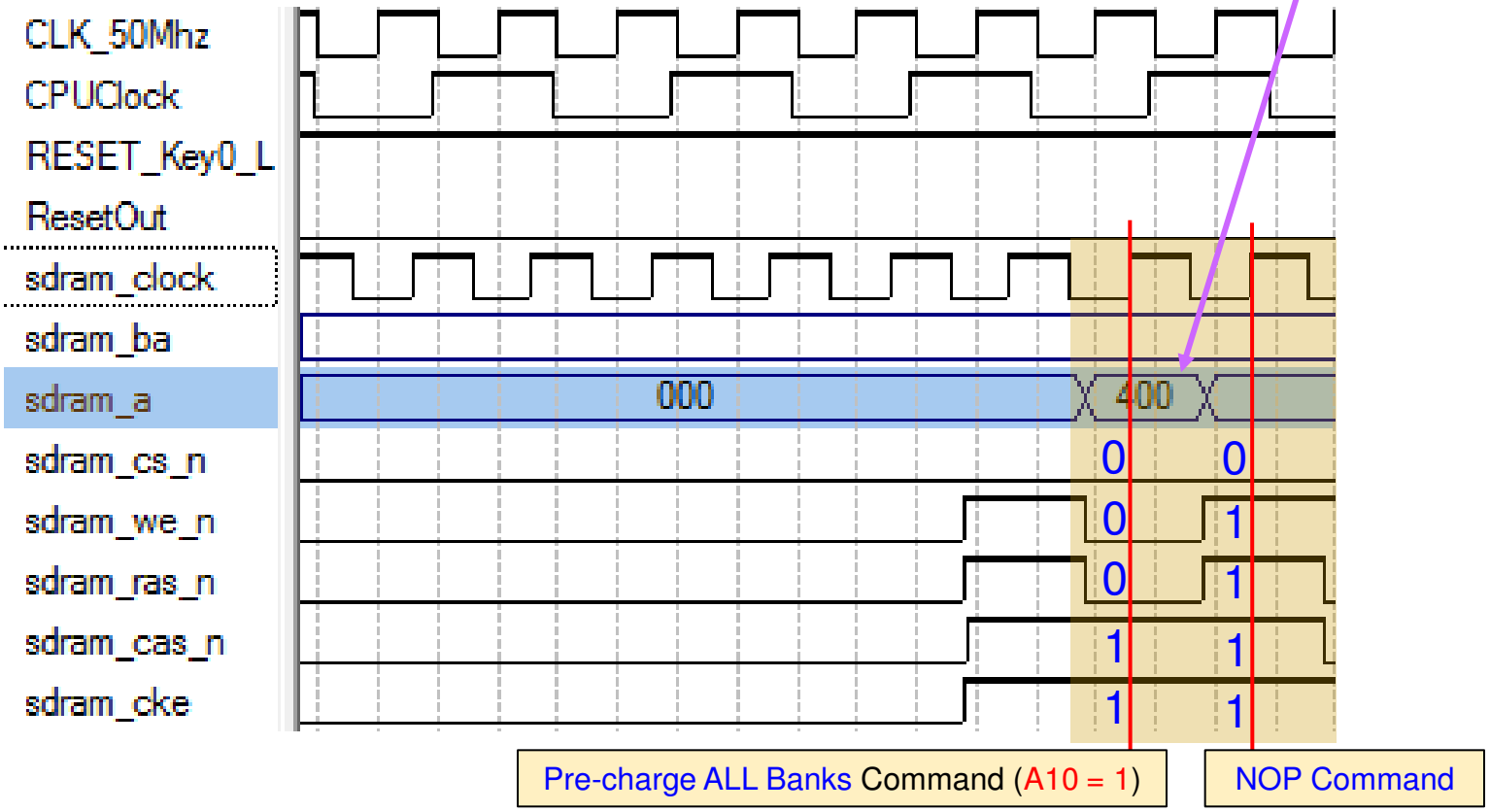
Power On Initialisation

- Here is the 1st and 2nd step where SDram CKE is taken to logic 1 after 200uS and we start issuing a NOP command, i.e. CS, WE, RAS and CAS are all logic 1.



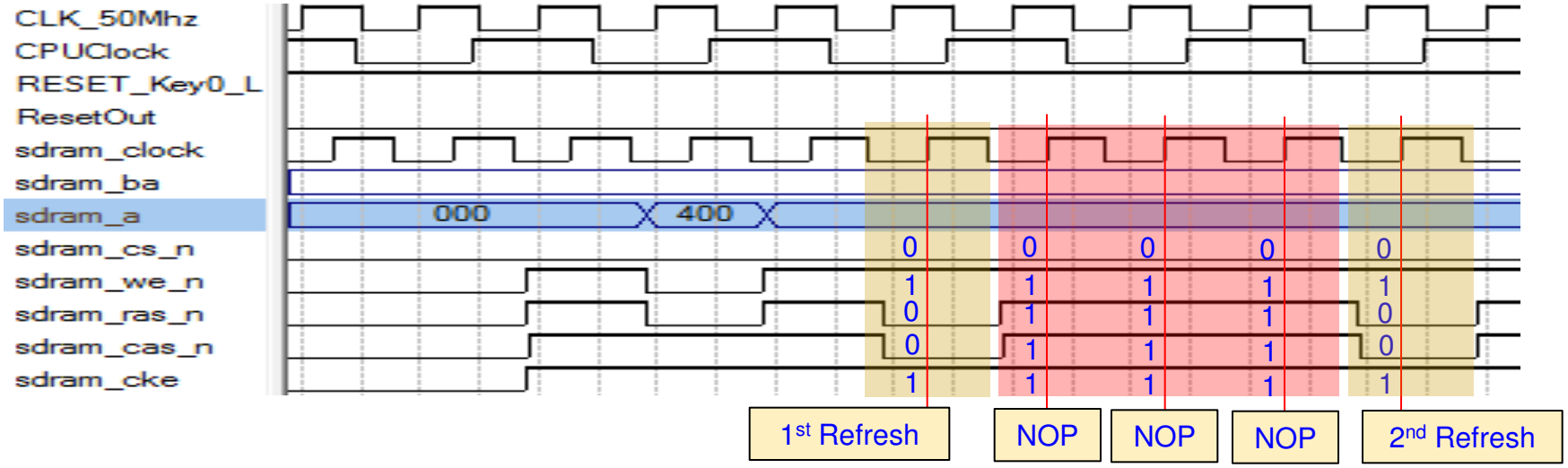
Power On Initialisation

- Here is the 3rd and 4th steps where we issue a Pre-charge all banks command (*i.e. a pre-charge command – Page 7 where CS=0, WE=0, RAS=0 and CAS=1*). This deactivates all banks and rows within the chip. There must be a delay of at least **18ns** after the Pre-charge command (the *timing parameter t_{RP} –page 38 in the Zentel data sheet*) before a new command so for safety follow with 1 NOP command (at 50Mhz a new command is issued every 20ns).
- Note **A10 = 1 for pre-charge ALL banks command.** This explains the hex value 400 we can see on the 12 bit address bus A11-A0 below, i.e. 0100 0000 0000.



Power On Initialisation

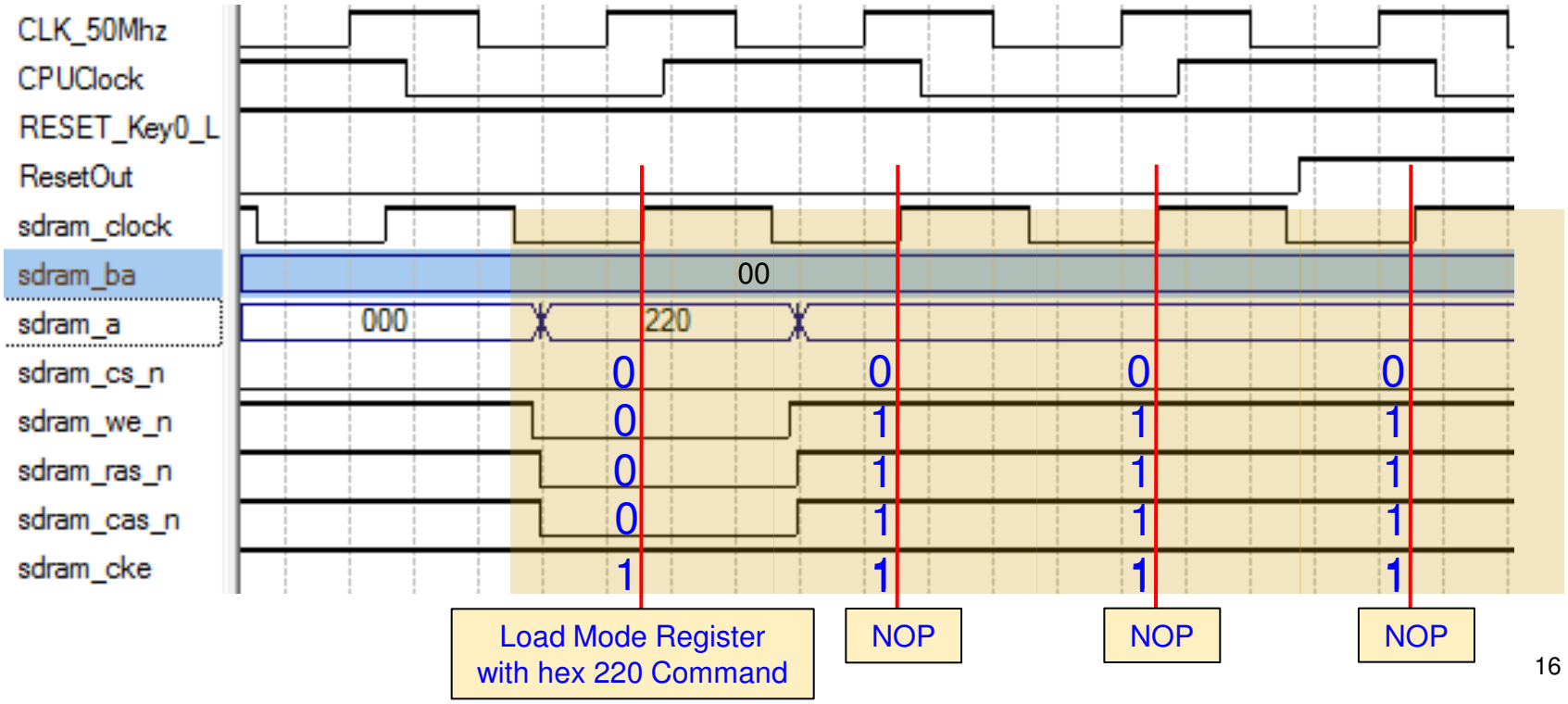
- Here is the 5th step where we issue 8 or more refresh commands (*see Page 7*) i.e. CS=0, WE=1, RAS=0 and CAS=0. The Zentel chip data sheet suggests a min of 8 refresh commands, whereas most other manufacturers chips seem to require only 2.
- I issued 8 in total (*you can only see 2 refresh commands in the screen shot below*).
- Refresh commands on the Zentel chip of the DE2 board require a minimum of 58ns of delay (Time t_{RC} on Zentel data sheet) *which for safety margin means at least 4 clock periods @ 50Mhz*) between them and the next command (*whatever that command may be*), so **always** issue 3 NOP commands after **each** refresh command before issuing any further commands to the Dram.



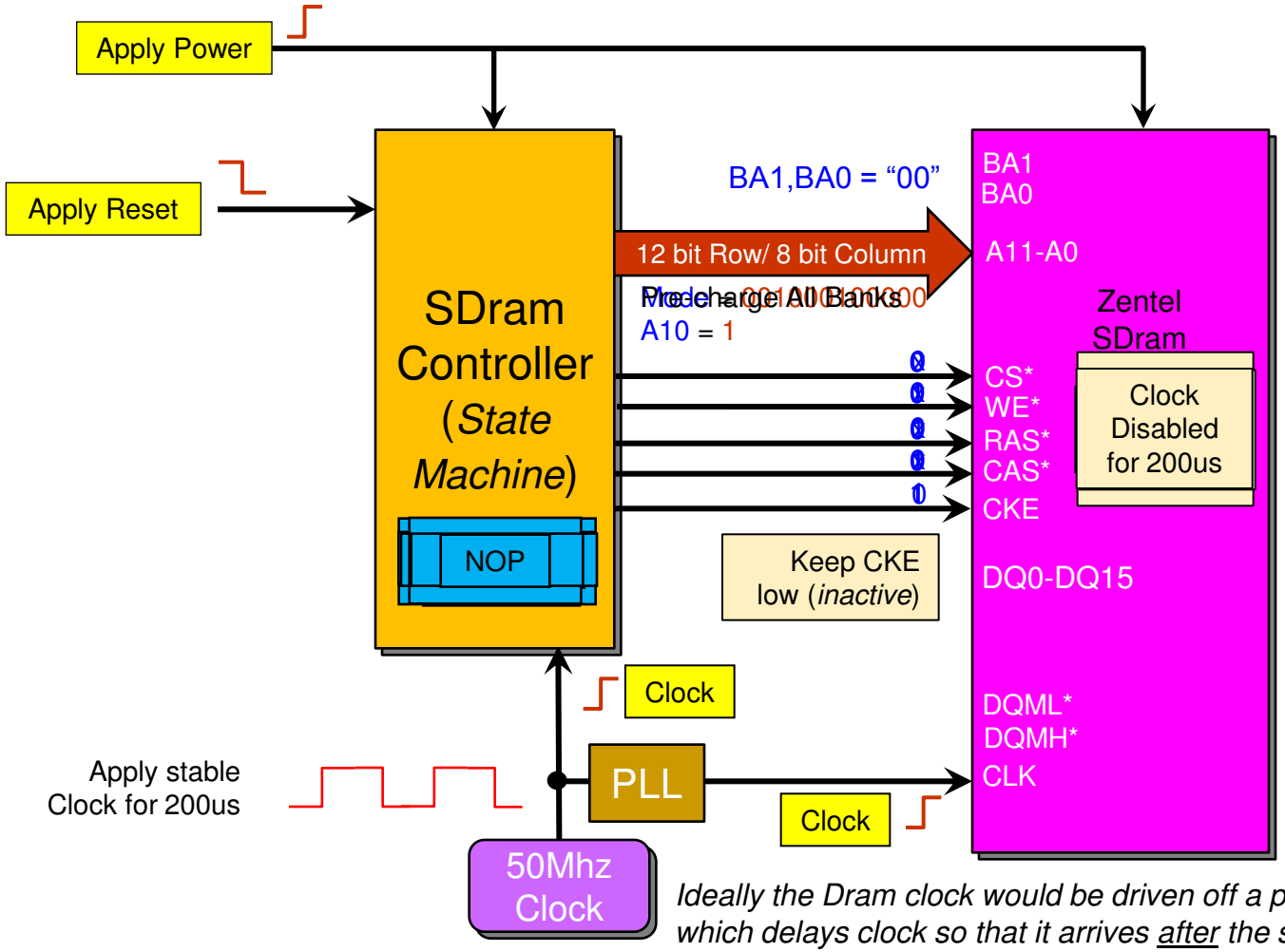
- The last step is to program the mode register.

Programming the Mode Register

- You can see the timing diagram associated with loading the mode register here. The programming occurs on the rising edge of the SDram clock as shown below, where hex 220 (*see Page 10*) is presented on the address lines and CS=0, WE=0, RAS=0 and CAS=0 and CKE=1.
- NOTE: Bank Address should be "00" when loading the mode register. This is something the Zentel Chip mentions but not all SDram chips seem to need this.
- At least 3 NOP commands must be issued after loading the mode register before the device is ready to accept read or write commands.



Animation showing Reset and Power On Initialisation



Ideally the Dram clock would be driven off a phase locked loop (PLL) which delays clock so that it arrives <u>after</u> the state machine has issued the Dram command (i.e. meets command setup time at Dram). For low speed clocks we can use an Inverter/Not gate

Note this animation shows the sdram controller issuing just 2 refresh commands (*followed by 3 NOPs*). The Zentel chip says 8 refresh commands are required.

Power On Initialisation

- How do I issue these commands?
- You design a state machine driven off the same clock as the SDram to drive the SDram signals CS, WE, RAS, CAS and CKE signals.
- After reset this state machine runs through a set of initialisation states to program the SDram chips and then enters an idle state (where it issues NOP or refresh commands to the dram) ready to accept read and write requests from the processor.
- By driving the sdram clock off a time delayed state machine clock (i.e. the output of the PLL - which can be fine tuned), we can get the state machine to output new commands (i.e. values for CS, WE, RAS, CAS, CKE signals) on one edge of the SDRAM clock, thus allowing plenty of time to meet the set up times at the SDram pins prior to its clock arriving at the SDram
- This has been done on the template project file you have. However because the clocks speeds on the DE2 are slow, we can get away with a simple inverter/NOT gate to delay the dram clock by half a clock relative to the sdram controller state machine (i.e. plenty of time 50Mhz = 20ns, so half a clock delay is 10ns, that is the sdram clock is delayed by 10ns relative to Sdram controller clock). We'll look at the timing later.

Issuing a CPU Address to the SDram Chip

- Reading and Writing to the SDram chip both require the presentation of a CPU address to the chip.

- The SDram latches the address internally as separate bank, row and column addresses upon application of an Activate, Read or Write command.

- We thus have to split the address lines coming from the 68k into groups of Bank, Row and Column address bits. So how many address lines do we allocate to the Bank, Row and Column addresses?

  We look at the data sheet for the Dram chip we are using, i.e. the Zentel chip on the DE2 and study it's architecture.

- The Zentel Chip is an 4M Word chip organised internally as 4 banks of 1M x 16 bits.

- This means that it requires a 22 bit address (2 bits for the Bank and 20 bits to specify a unique Row/Column location within the 1M space of the chosen bank).

Issuing a CPU Address to the SDRam Chip

- Although it's not important in this application (*because we do not use burst mode for read or write operations*), it makes sense to allocate A22-A21 of the 68k's address bus as the Bank address inputs to the Sdram, i.e. map A22-A21 to BA1-BA0.

- We can then use A20-A1 as the remaining address lines to present as Row and Column addresses (*but any mix of address lines should work*).

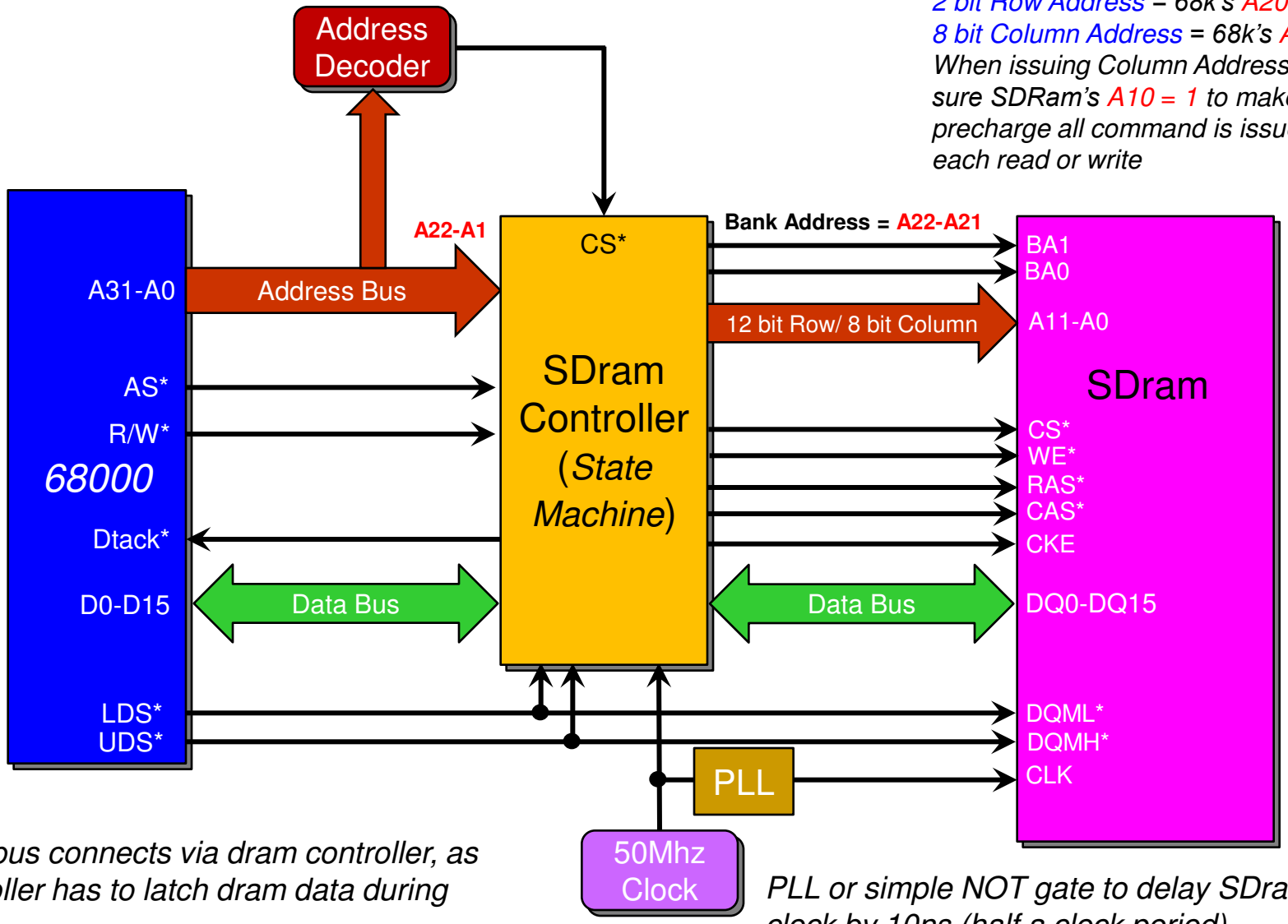
- Internally, each bank in the Zentel chip is organised into 4096 rows of 256 columns (*see page 5*), thus we need to break down A20-A1 into a 12 bit Row address and an 8 bit Column address and present each to the SDram via its 12 bit address bus.

- It's sensible then to present the 68k's address lines A20-A9 as the row address (i.e. map them to A11-A0 on the SDram) and A8-A1 as the column address (*i.e. map them to A7-A0 on the SDram*). This implies an address multiplexer of some kind inside the dram controller (*this is quite easy in VHDL*).

- NOTE: Early SDrams required that you issue a pre-charge command between a read or write command *(when for example switching to a new bank or a new row)*, but more recent devices like the Zentel chip can perform an auto pre-charge after any read or write provided you set A10 on the SDram to logic 1 when presenting the Column address for a read or write operation.

Interfacing the Soft Core 68k to 16 bit Wide SDram

2 Bit Bank Address = 68k's *A22-A21*
2 bit Row Address = 68k's *A20-A9*
8 bit Column Address = 68k's *A8-A1*
When issuing Column Address make sure SDRam's A10 = 1 to make sure a precharge all command is issued after each read or write



Data bus connects via dram controller, as controller has to latch dram data during read

PLL or simple NOT gate to delay SDram clock by 10ns (half a clock period)

Writing to the SDram

- In descriptive terms, the protocol for writing to a location in the SDram goes like this.

- 68k begins a write when R/W and *AS* goes to 0*, which can trigger SDram controller to present a Bank/Row address to the SDram and issue an Activate command (CS=0, WE=1, RAS=0, CAS=1) to activate a Bank/Row within the chip (*Page 7*).

- In the case of the 68k, we cannot give the SDram the write command immediately after an Activate command because the UDS*/LDS* signals from the 68k will not be valid at that point (*even though R/W indicates that a write is in progress*) so we have to insert SDram NOP commands until either UDS* or LDS* (*or both*) go low. This will ensure that the width of the 68k's data transfer is known to the SDram via the DQMH*/DQML* lines when a write command is actually issued.

- Once UDS*/LDS* become valid, controller can present a Bank/8 bit Column address (*remembering to set A10 =1 at this point for an auto pre-charge after the write*) and issue a Write command (*Page 7*) i.e. CS=0, WE=0, RAS=1, CAS=0

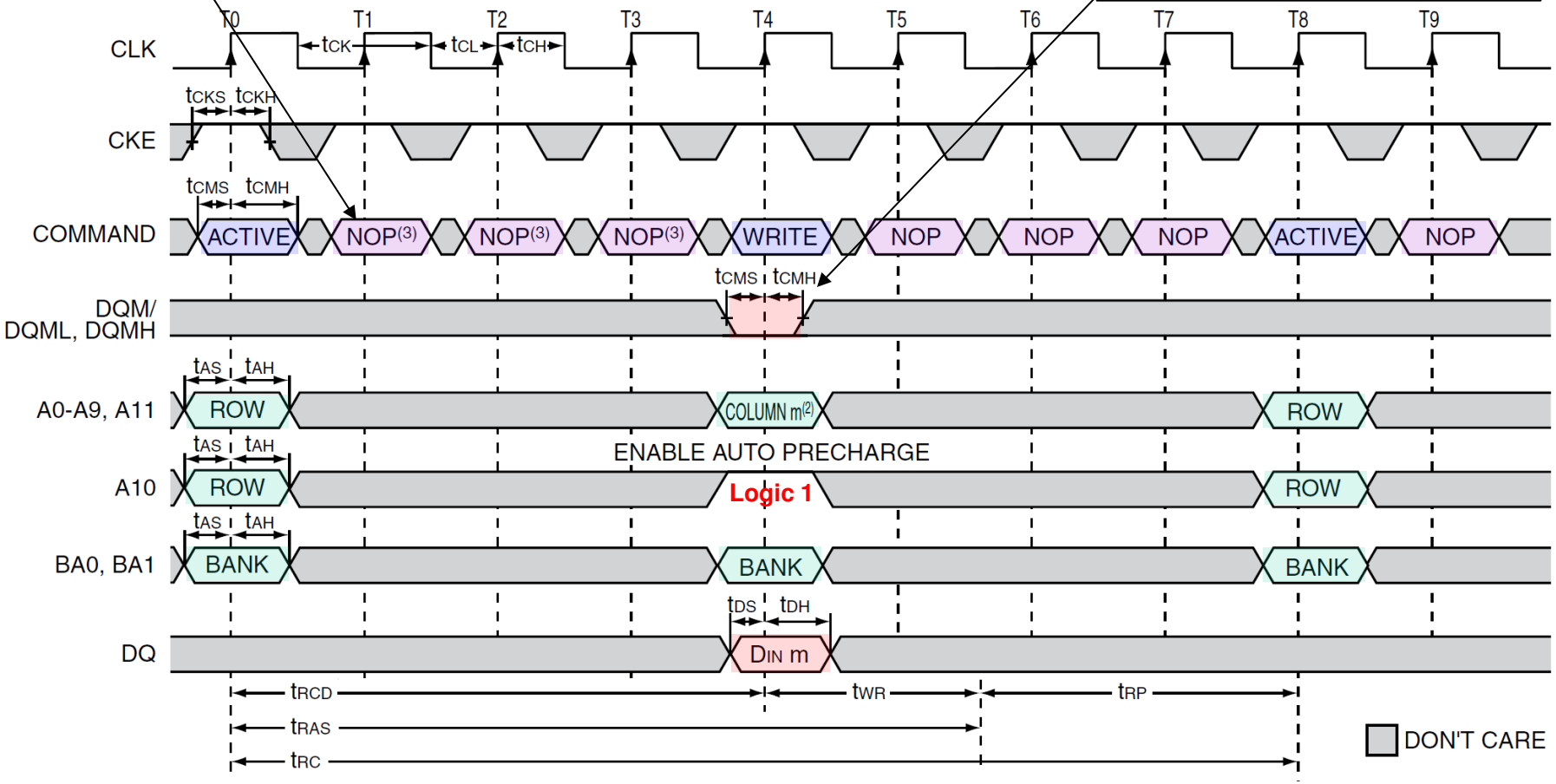
- The data we present on the SDram during the write comes from the SDram controller – which in turn is taken from the 68k data bus (*see previous slide*) and will then be stored into the bank/row/column address specified. The timing is shown overleaf. This means we **must** turn **on** (i.e. drive – not tri-state) the SDram controller **data out** signals during a write to drive data into the SDRam data lines.

Writing to the SDram

NOP commands only necessary to meet min delay between ACTIVATE and WRITE command. Can be 0 for a slow clock or more than that for a fast clock

Setup and hold times of all signals relative to clock = **1.5ns**

Byte select signals **DQMH/DQML** must be valid at time of write



Min row activate to read or write command
t_{RCD} >= 18ns or 1 clk @ 50Mhz

Min time between reads/writes
t_{RC} >= 58ns or 3 clk @ 50Mhz

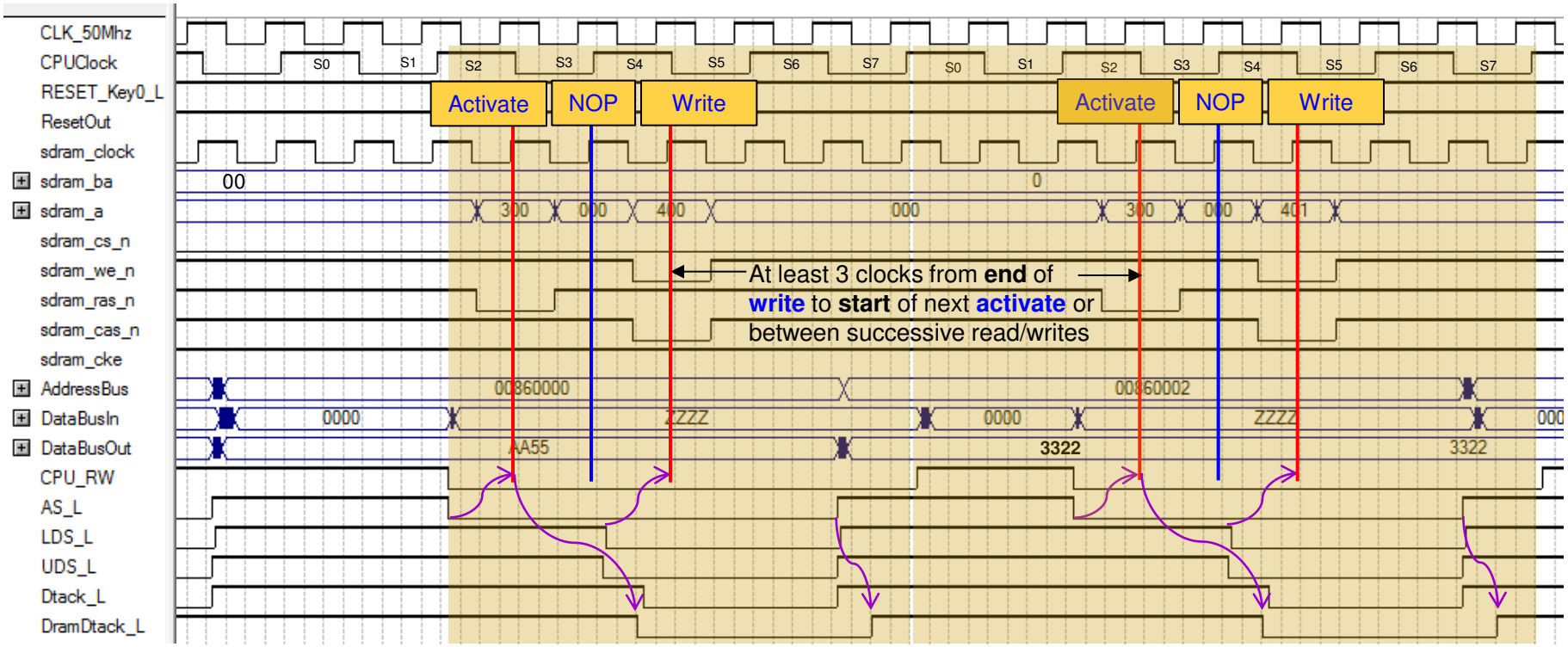
Time from end of write to activating a new row
t_{RP} + t_{WR} = 3 clks @ 50Mhz

Min row active Time
t_{RAS} >= 40ns or 2 clks @ 50Mhz

SINGLE WRITE - WITH AUTO PRECHARGE

Writing to the SDram

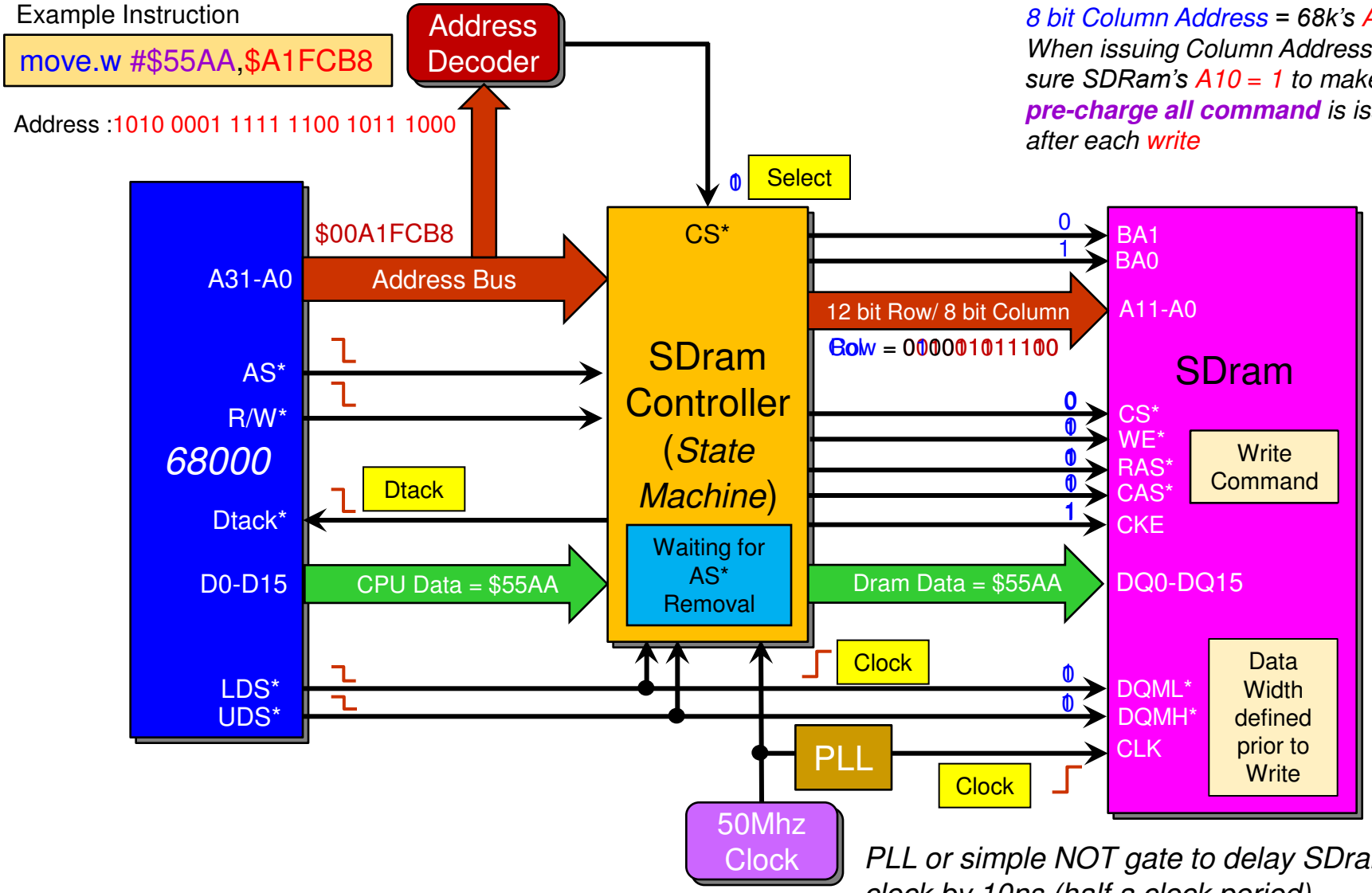
- You can see the simulated timing here where the 68k writes $AA553322 to location 00860000/1/2/3 (in this system it's dram not static ram). AS_L going low means we can issue **Activate** command. The address maps to a bank address of 0 a row address of hex 300 and a column address of 400 (*it's a column address of 0, but A10 = 1 (for the auto pre-charge command –see page 19) makes it 400*).
- Your dram controller can generate DramDtack_L pretty much as soon as it has presented the Activate command and must keep generating it until the 68k finishes its bus cycle (AS_L goes high).



Notice NOP command(s) inserted between ACTIVATE and WRITE command. We issue these while waiting for the 68k's UDS*/LDS* signals (and hence DQMH*/DQML* at the SDram) to become valid to indicate data width **before** issuing the WRITE command.

Animation of Write Operation

2 Bit Bank Address = 68k's *A22-A21*
2 bit Row Address = 68k's *A20-A9*
8 bit Column Address = 68k's *A8-A1*
When issuing Column Address make sure SDRam's A10 = 1 to make sure a ***pre-charge all command*** *is issued after each write*



PLL or simple NOT gate to delay SDram clock by 10ns (half a clock period)

Reading from the SDram

- Reading from the SDram involves presenting an address to the chip and issuing an Activate command, followed by a READ command. In descriptive terms, the operation to read goes like this.

- When CPU issues Address Strobe (AS* = 0, R/W = 1), present a Row and Bank Address to the Sdram and issue the Activate command to activate a bank/row within the chip (Page 7) i.e. CS=0, WE=1, RAS=0, CAS=1. You can also generate Dtack* here if your SDram design is fast enough to not require wait states on the 68000.

- Because UDS*/LDS* is driven **early** in the bus cycle for 68k read operation, we can present a Column and Bank address (*remembering to set A10 =1 at this point for an auto pre-charge*) and issue a READ command immediately after the Activate command (Page 7) which means setting CS=0, WE=1, RAS=1, CAS=0.

- **IMPORTANT**, your SDram controller must tri-state it's data out lines to the SDram during a read operation so as not to create conflict at the bi-directional data lines of the SDRam chip which will be driving data out at this time.

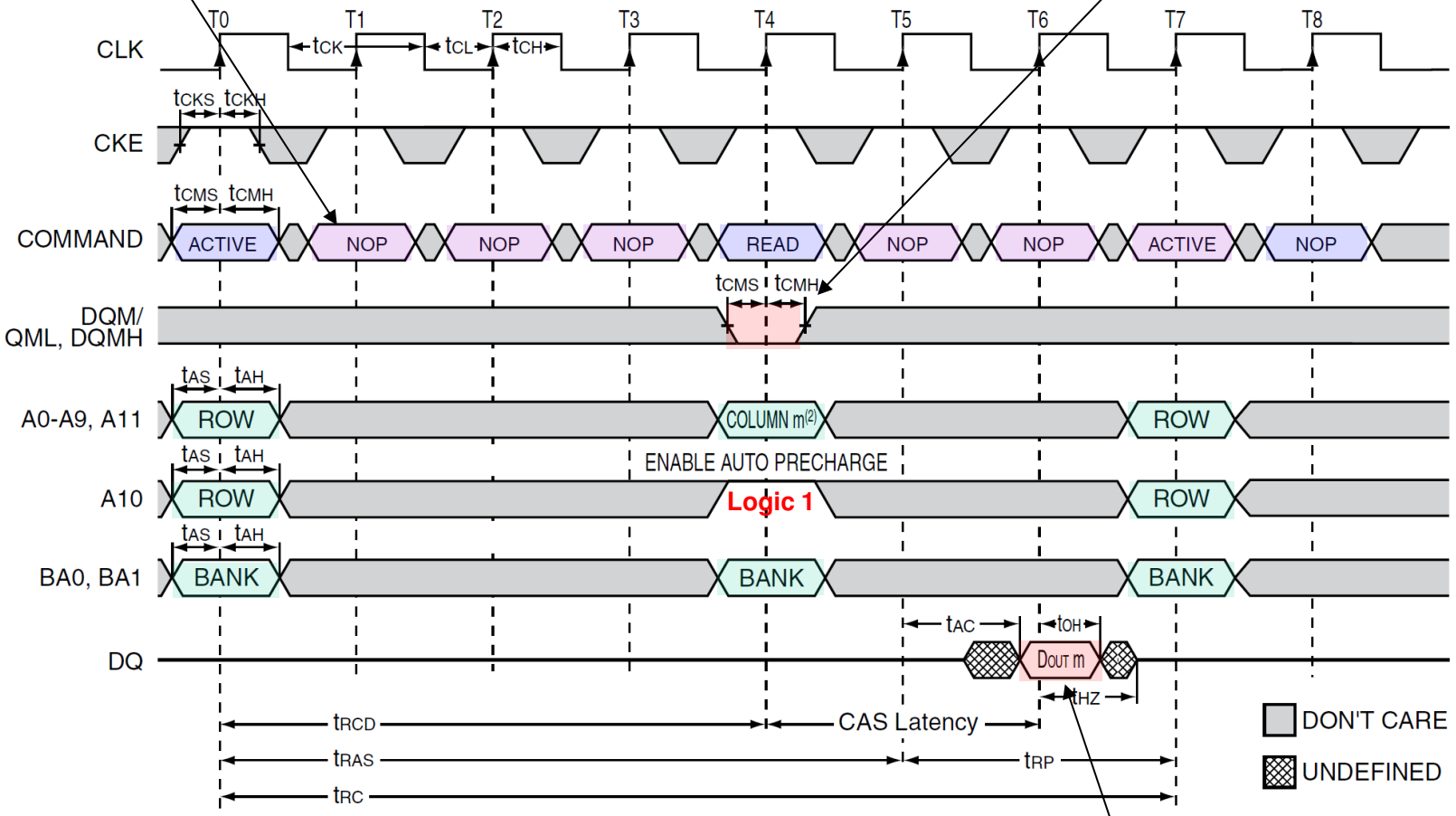
- The SDram will output it's data after the programmed CAS latency period of 2 clock cycles has expired. Your SDram controller **MUST** latch this data internally and keep presenting it to the 68k, since the SDram chip will remove the data on the next clock. This is IMPORTANT (*see next timing diagram*) !!

Reading from the SDram

NOP commands only necessary to meet min delay between ACTIVATE and READ command. Can be 0 for a slow clock or more than that for a fast clock

Setup and hold times of all signals relative to clock = **1.5ns**

Byte select signals **DQMH/DQML** must be valid at time of read



Min row activate to read or write command
t_{RCD} >= 18ns or 1 clk @ 50Mhz

Min time between reads/writes
t_{RC} >= 58ns or 3 clk @ 50Mhz

Time from end of read to activating a new row
t_{RP} = 18ns or 1 clk @ 50Mhz

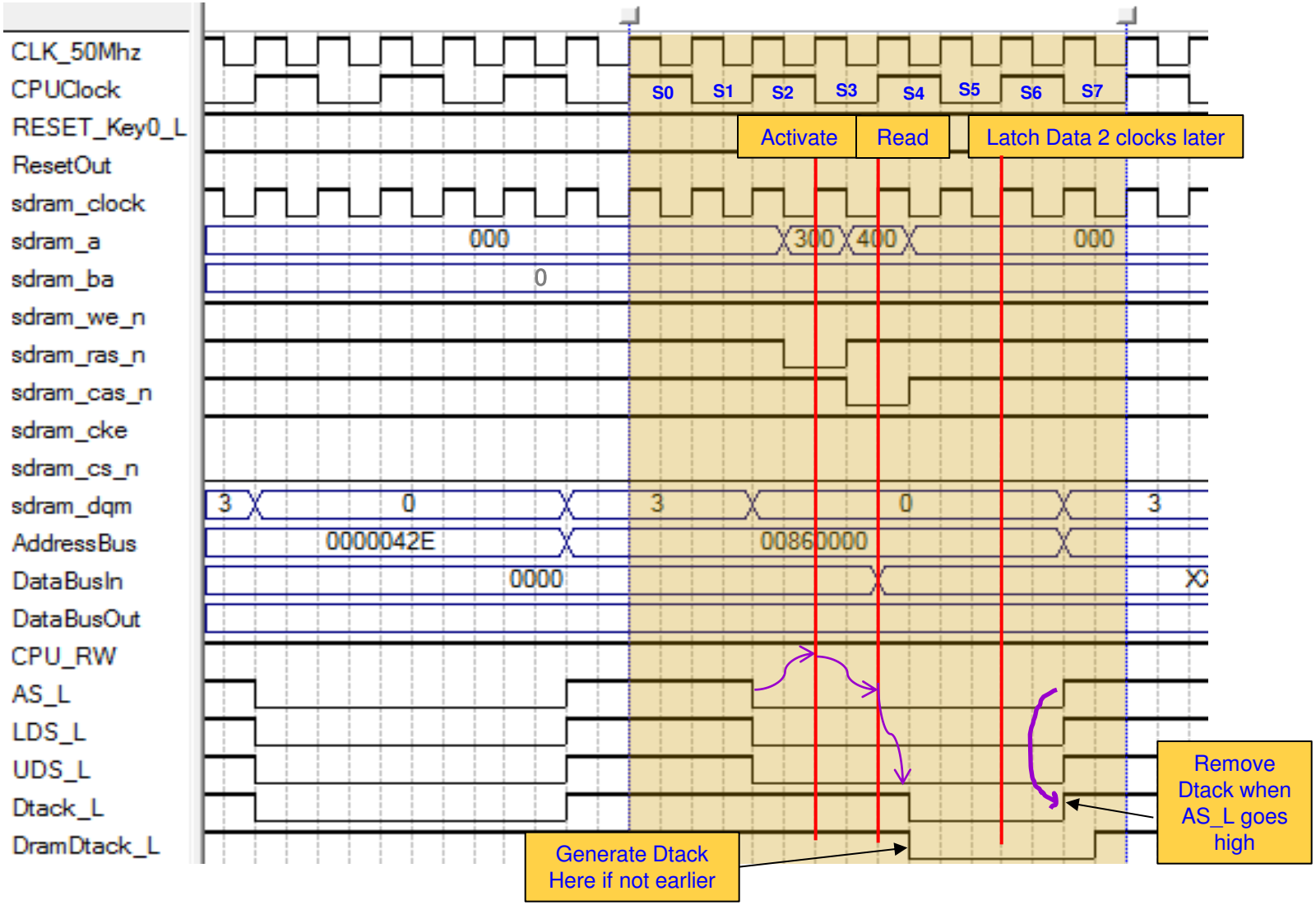
Min row active Time
t_{RAS} >= 40ns or 2 clks @ 50Mhz

SINGLE READ WITH AUTO PRECHARGE

Data must be latched by SDram controller 2 or 3 clocks after read (CAS latency)

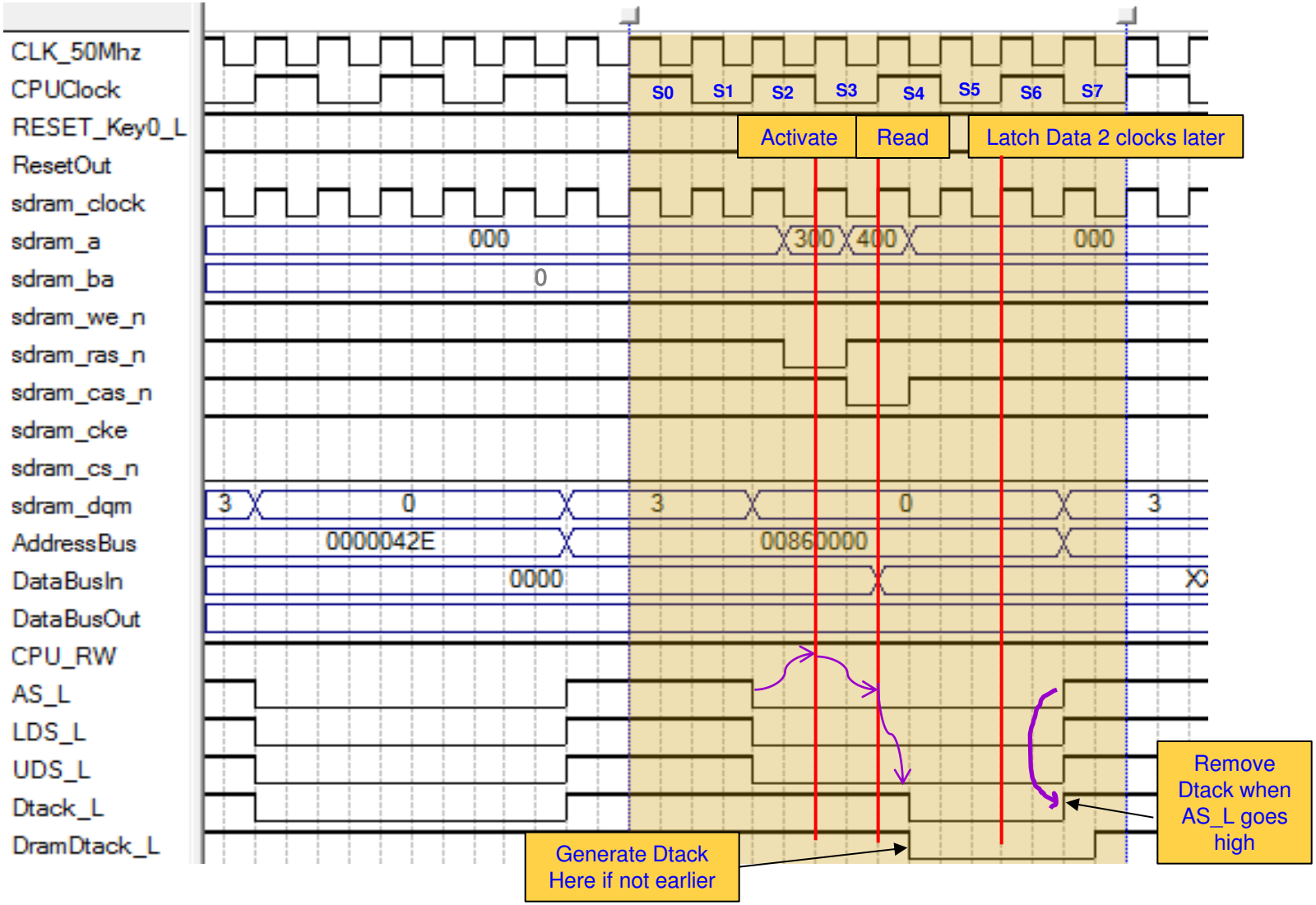
Reading from the SDram

- You can see the simulated READ timing here where the 68k reads a word at location 00860000. This maps to a bank address of 0 and a row address of hex 300 and a column address of 400 (*actually it's a column address of 0, but A10 = 1 makes it 400*).



Reading from the SDram

- You can see the simulated READ timing here where the 68k reads a word at location 00860000. This maps to a bank address of 0 and a row address of hex 300 and a column address of 400 (*actually it's a column address of 0, but A10 = 1 makes it 400*).



Reading from the SDram

- Dtack* should only be generated when you are sure your data will be ready to give back to the 68k at the start of S7. Too early and the 68k may finish the access before the dram controller has latched the data from the dram.

- After experimentation and debugging, it was found to work if you issue Dtack* in the state immediately after issuing the Read command (*see previous illustration*)

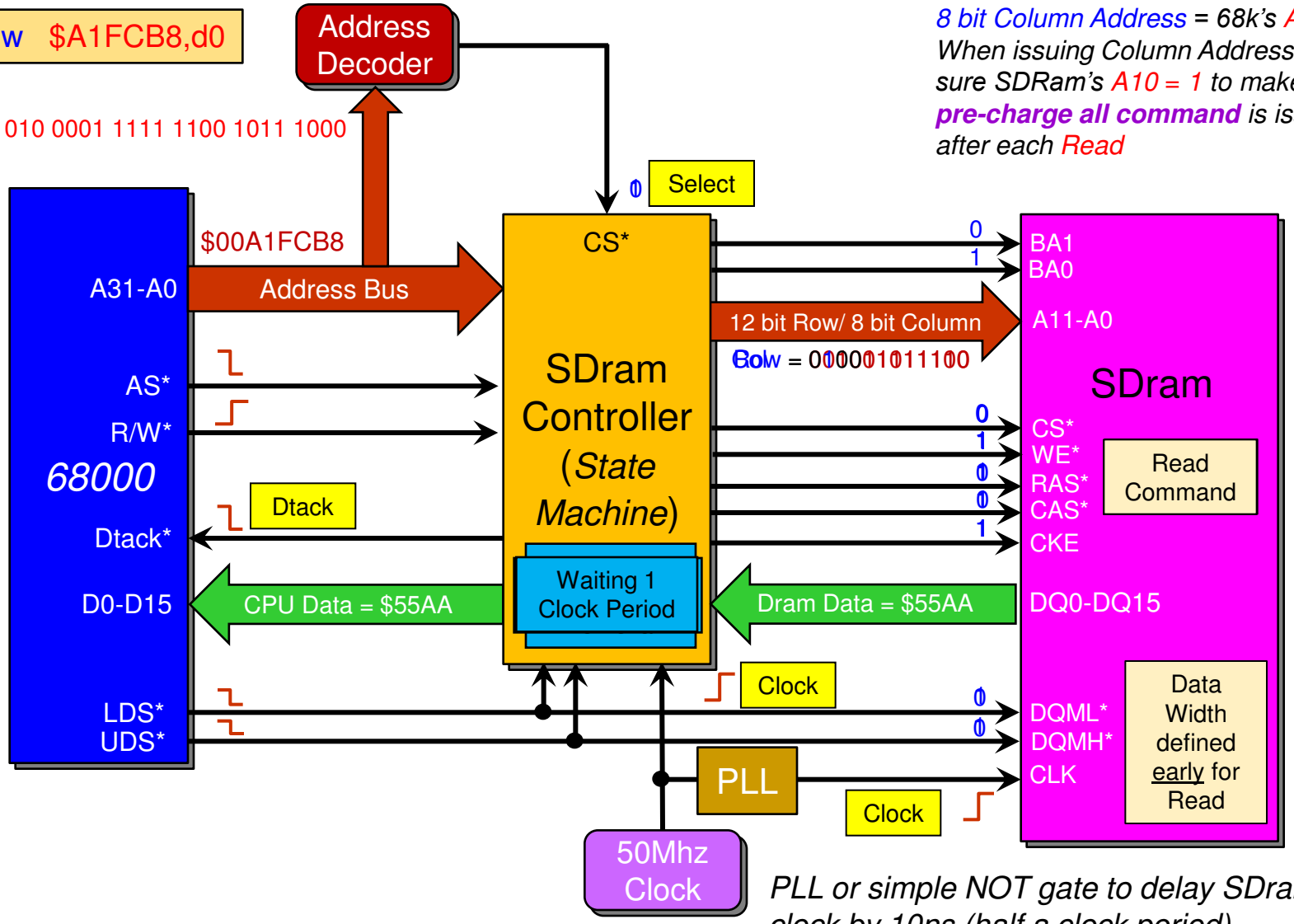
Animation of Read Operation

2 Bit Bank Address = 68k's *A22-A21*
2 bit Row Address = 68k's *A20-A9*
8 bit Column Address = 68k's *A8-A1*
When issuing Column Address make sure SDRam's A10 = 1 to make sure a ***pre-charge all command*** *is issued after each Read*

Example Instruction

move.w $A1FCB8,d0

Address :1010 0001 1111 1100 1011 1000



PLL or simple NOT gate to delay SDram clock by 10ns (half a clock period)

SDram Refreshing

What, Why and How

- Refreshing arises because the charge on the capacitor of the dram cell leaks away.

- Refreshing 'tops-up' the charge before the data is lost due to leakage.

- Few memory chips refresh completely by themselves. External hardware is usually required if only to tell the dram *when* to perform it.

- Refreshing involves reading every cell in the chip (*which implicitly means that the charge on the cell gets topped up*) within a period specified by the manufacturer, (*typically 64-128mS*)

Asynchronous Dram Refreshing

- A single refresh operation will refresh all cells sharing the same row address, thus we do not have to refresh all cells individually.

- For a 4 Meg location device such as the Zentel chip on the DE2 this means that every row has to be refreshed at least once during the specified 64mSec period and a row address is a 12 bit value, i.e. there are 4096 rows in that device.

- Most modern Dram chips simplify the task of refreshing by having a refresh counter fabricated onto the chip (*see page 5*) to keep track of which Row is to be refreshed next.

- **Important**: during a refresh, the CPU must not be permitted to access the memory chip. This can be achieved by delaying Dtack back to the 68k until the chip has refreshed and then perform the delayed 68k's read or write request (*driven off AS**).

Asynchronous Dram Refreshing

Hardware Refresh Modes

- Burst Mode:

  Here all Rows in the chip are refreshed in one continuous burst of refreshing, one row after the other as fast as possible.

  The downside is that the CPU will not be able to access the memory until all rows have been refreshed, leading to less predictable execution timing from a program, but most high performance CPUs have a cache these days, which means the CPU can probably carry on executing during a refresh.

- Interleaved Mode:

  This is the most common arrangement, here the refreshing is interleaved between normal CPU accesses to the memory as it leads to more predictable program execution. For example, a refresh request for one row in the dram can be initiated every 64/4096 ms, or every 15.6us for a device like the Zentel chip.

- You will need a refresh timer in your dram controller to count clock cycles at 50Mhz and issue a refresh request to your Dram controller state machine no later than every 15.6us.

Synchronous Dram Refreshing

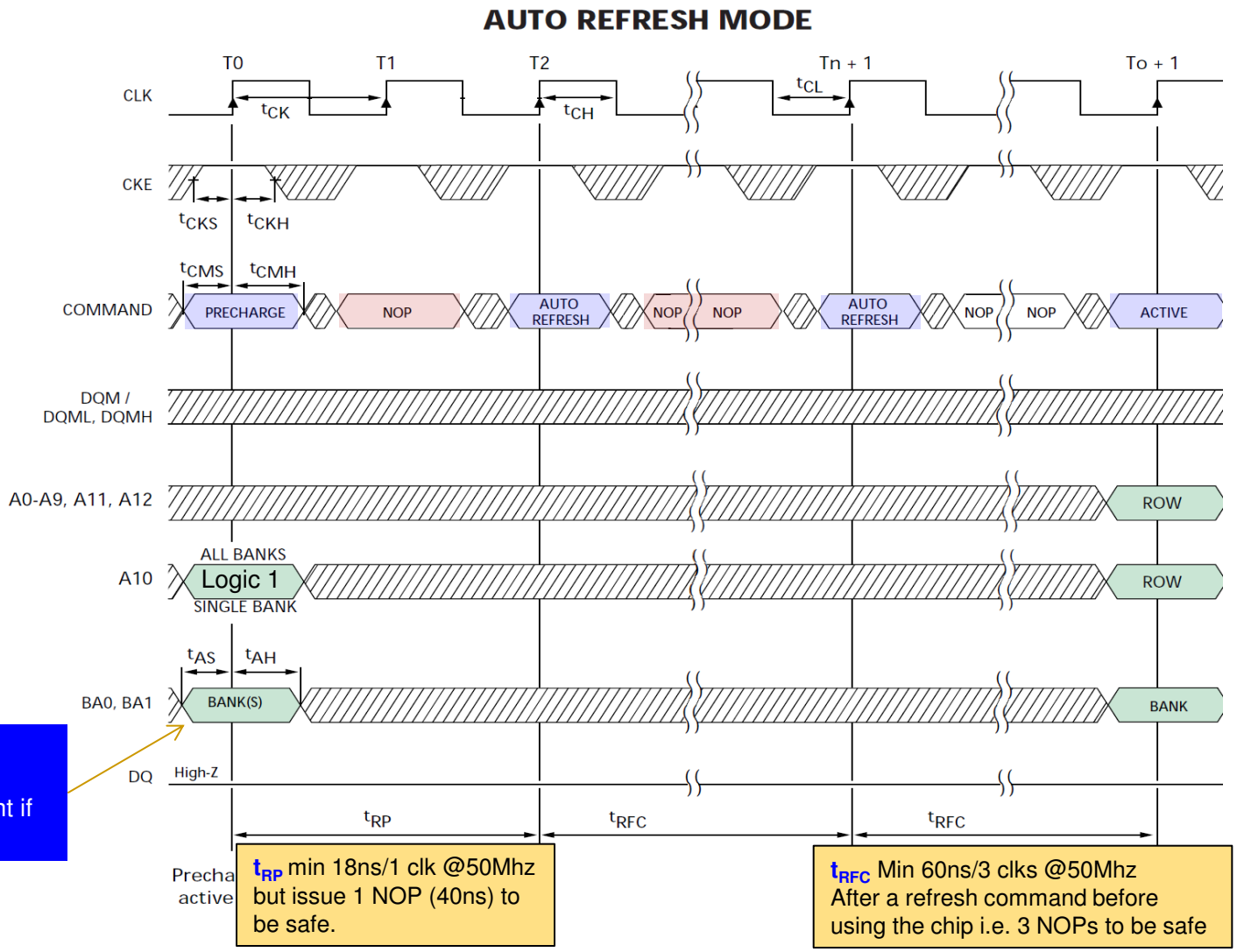
- The Sdram chip has a command to tell it to perform an internal refresh of the next row (*it keeps track of which row is next using an internal counter*). Page 7, Refresh command means CS=0, WE=1, RAS=0, CAS=0

- Refreshing works like this

1. Issue a pre-charge all banks command (A10 must = 1) which means that the bank address is *irrelevant*.
2. Issue 1 NOP command (i.e. at least 18ns – see next page)
3. Issue an Auto-Refresh command
4. Issue 3 NOP commands (i.e. at least 60 ns or 3 clocks after last command – see next page)
5. Dram is now ready for 68k to access.

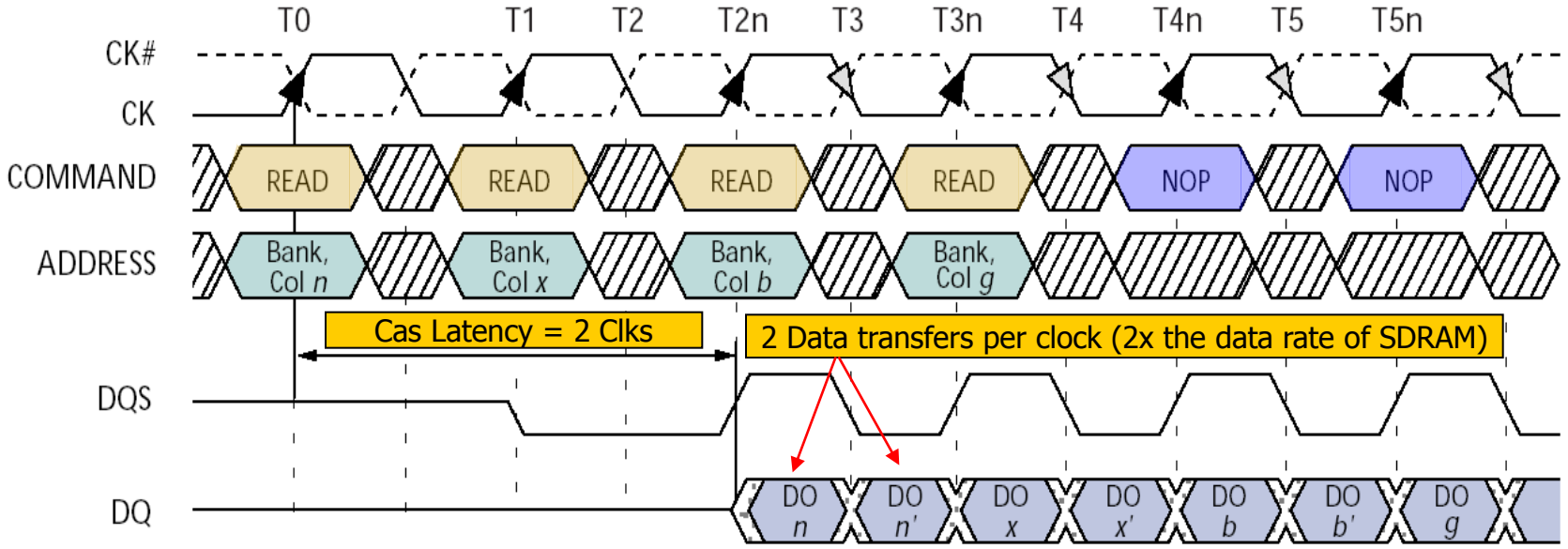
Synchronous Dram Refreshing

- The timing diagram is shown here



Double Data Rate Synchronous Dram (DDR SDRAM)

- This technique improves upon ordinary SDRAM by transferring data on both the *rising* and *falling* edges of the clock signal. DQS acts as a data strobe to help latch the output data during read operations and input data during writes.
- For a 133Mhz Clock and a byte wide chip, 266Mbytes of data per second can be transferred hence the name DDR-266 Ram.
- This makes it twice as fast as SDRAM after the initial CAS* latency, although in practice (*given the presence of good CPU caches*) the speed improvement as observed by the user over SDRAM is not large (maybe 5%).
- Today DDR3-33 Ram (167 MHz Clock) and DDR-400 (200Mhz) chips are available.
- The timing diagram below shows random column address access to the chip with data for column n and subsequent data n' available on each edge of the clock.

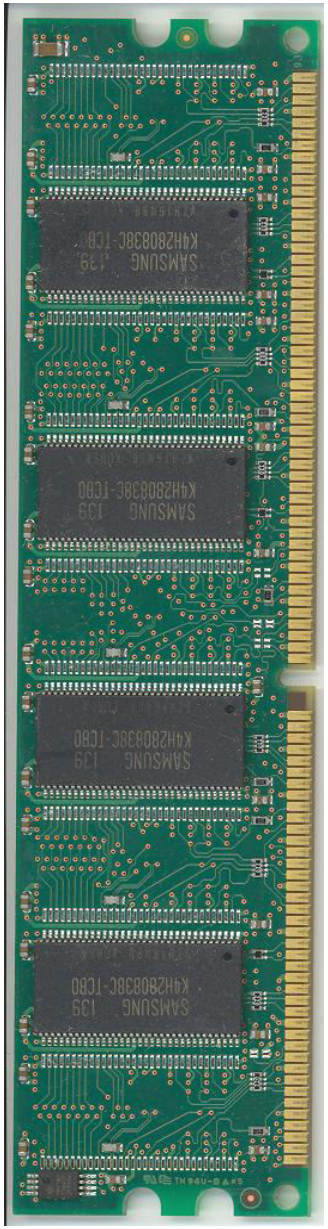


DDR SDRam DIMM Modules

- DDR SDram chips are most commonly used inside the PC where they are often packaged onto a small daughterboard module where typically 8 devices operate in parallel giving 64 bit wide data paths and helping to achieve high bandwidth.

- These modules are advertised with names such PC2100, 2700, 3200 etc. ram which derive their name from the bandwidth of the module, e.g. PC2100 = 2100 MB/sec.

- This is related to clock speed and the width of data. For example, using 133MHz DDRram with 64 bit wide data transfers on both rising and falling edge of the clock gives us 133 * 2 * 8 = 2100 MBytes per second.



DDR2 and Beyond

- DDR2 keeps the same relatively slow (*and cheap*) 'core' memory speed used in 133Mhz SDram and DDR 266 ram, with roughly a 15 ns latency or access time i.e. 2 clock cycles at 133 MHz to get at the first item of data in exactly the same time, but by doubling the clock rate of DDR ram, DDR2 ram could get at *subsequent* data in the same row even faster.
- However because of the slow core memory speed, initial CAS latency is now 4 clock cycles to get at the first item of data so they are labelled 4-1-1-1 etc.

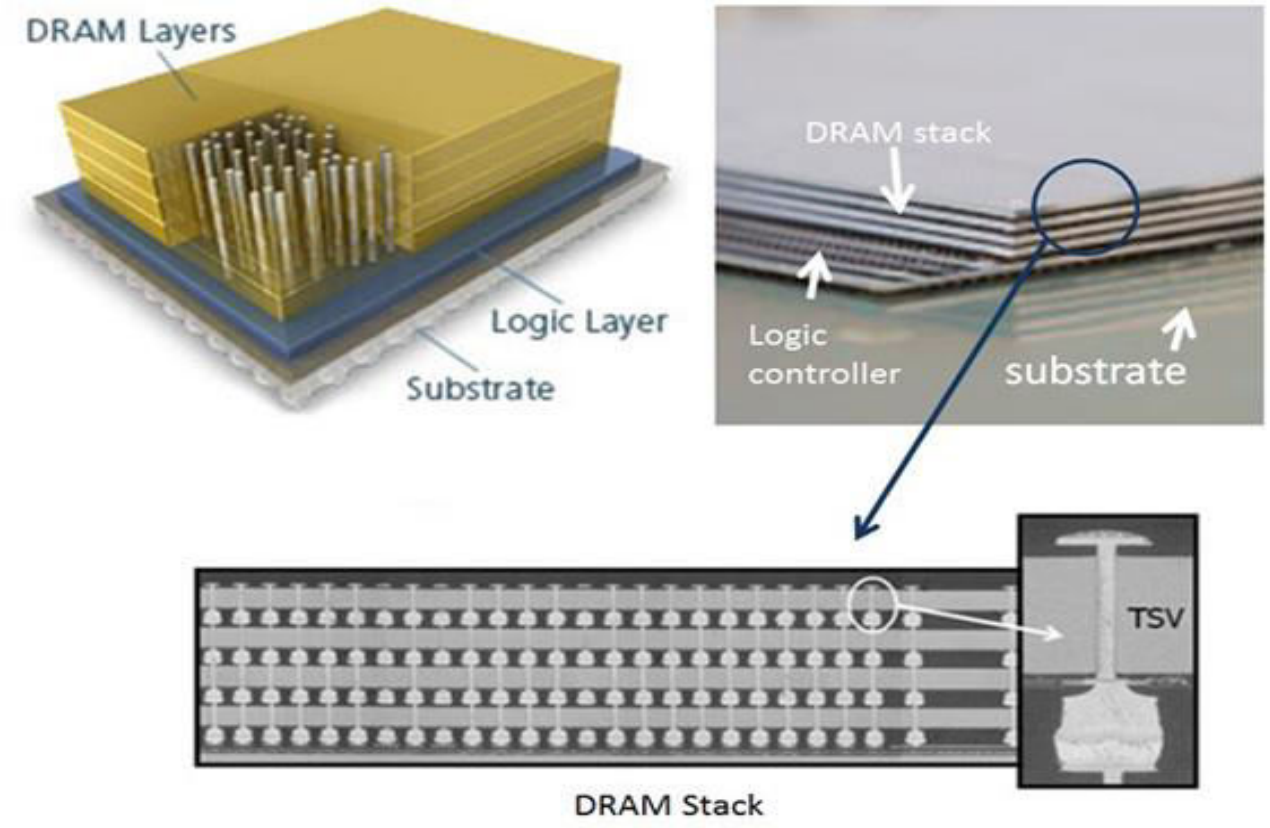
- So DDR2 ram clocked at 533Mhz transferring data on both edges giving us DDR2-1066 memory chips and PC2-8500 Memory modules.

Standard name	I/O Bus clock	Data transfers per second	Module name	Peak transfer rate
DDR2-400	200 MHz	400 Million	PC2-3200	3200 MB/s
DDR2-533	266 MHz	533 Million	PC2-4200 PC2-4300[1]	4266 MB/s
DDR2-667	333 MHz	667 Million	PC2-5300 PC2-5400[1]	5333 MB/s
DDR2-800	400 MHz	800 Million	PC2-6400	6400 MB/s
DDR2-1066	533 MHz	1066 Million	PC2-8500 PC2-8600[1]	8533 MB/s

- DDR3 uses lower voltages and even higher clock frequencies (1Ghz)
- DDR4 is expected to use 1.2v and clock frequencies >2Ghz

Intel and Micron's Hybrid Memory Cube Consortium

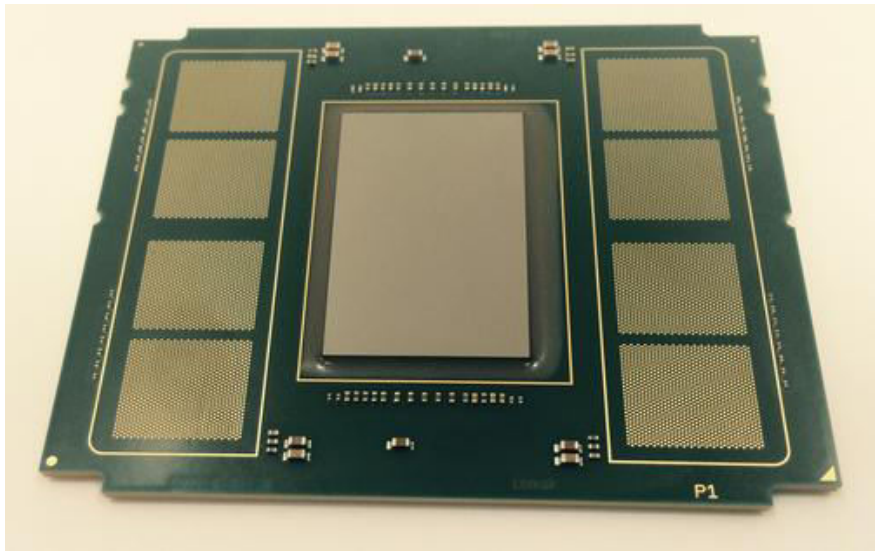
- Stack Dram chips vertically with connections made through TSV (*through silicon vias*). Needs very precise alignment during manufacture < 3uM.
- 160GB/sec bandwidth – 1 possible replacement is for DIMM memory modules.
- Vastly reduced power.



Intel and Micron's Hybrid Memory Cube Consortium

- Intel's Xeon Phi "*Knights Landing*" (3-TFLOPS) processor developed for massively parallel processing, is rumoured to feature up to 72 cores with connections ready to accept 8 hybrid memory cubes yielding 16-Gbytes of on-chip memory.
- It has enough in-package memory and off-chip DDR (max 384 Gbytes) to run massively parallel applications without adding fancy swapping and caching techniques.

Micron 2GB memory cube



Intel's Xeon Phi "*Knights Landing*" CPU

Improving Memory System Performance

- Essentially there are four ways to improve the performance of a memory system

1. The most obvious is to raise the IO clock speed of the device so that data can be accessed more quickly (as with DDR3 Ram which run at up to 1GHz).

2. Increase data width e.g. 64bit, 128bit, 256bit, 512bit wide memory etc. to get more data per clock.

3. Have multiple banks of memory e.g. multiple hybrid memory cubes.

4. In areas such as graphics cards, where massive performance is required, these techniques are combined. The latest AMD Radeon R9 290X (dual GPU) utilises 2 x 512 bit wide memory data paths (128 bytes transferred per clock) with GDDR5 Ram (*essentially DDR3 optimised for graphics applications*) clocked at 1.25Ghz to achieve transfer rates in excess of 320 GBytes/sec (i.e. 1.25 * 2 * 128). Which is enough for 600 frames per second at 1080p resolution