Refer to the datasheet for the Micron SDRAM MT48LC16M16A2TG-7E.

1) Basic info
   a. What do the first and second "16"s in the chip name mean?
      The portion of the part number "16M16" tells us that this chip uses a 16 Meg x 16 architecture. Note that all the chips described by this datasheet have the same density: 256Mb. The first "16" in the chip name means there are 16 million words (actually, 16,777,216 or 2^{24}). The second "16" means there are 16 bits per word, so the data bus is 16 bits.
   b. How many banks per chip? 4; Rows per bank? 8192; Columns per row? 512; Bits per column? 16; What's the product of all of these numbers (the units are bits per chip)? 268,435,456 = 256Mb.
   c. How many bits do I need to specify a single bank in the chip? 2; A single row in a bank? 13; A single column in a row? 9.
   d. In Figure 3, how many pins are dedicated to address information? 15; Data? 16.
   e. Are the banks square (same number of rows as bits/row)? Yes – 8192 x 8192 bits. Why are the row and column address sizes different? There are 8192 individual rows, so we need 13 bits to address the rows (2^{13}=8192). The columns each contain 16 bits, so there are only 512 columns. 512x16=8192, so there are the same number of rows as bits/row, but we only need 9 bits to address 512 columns (2^9=512).
   f. Does the chip use positive or negative edge-triggered logic? All SDRAM input signals are sampled on the *positive* edge of CLK.
2) Basic operation
   a. Approximately how long does it take to initialize the chip?
      (100us) + (t_{RP} precharge) + (2 x t_{RFC} autorefresh cycles) + (t_{MRD} mode register programming). Using values for the -7E given in Table 20 & 22: 100us + 15ns +2*66ns + 2*t_{CK}. With a 27MHz clock, t_{CK} = 3.8e-8 sec = 38ns. So the total time for initialization is 100us after the clock is stable before any commands, then 223ns for precharge, autorefresh, and mode register programming.
   b. What is the purpose of the mode register?
      Identifies the specific mode of operation of the SDRAM, including burst length, burst type, CAS latency, operating mode, and write burst mode.
   c. What does "burst mode" mean?
      In burst mode, the READ and WRITE accesses start at a selected location (column) and continue for a programmed number of locations in a programmed sequence. Bascially, one command effectively accesses multiple columns. The number of columns accessed is determined by the BL (burst length) setting.
   d. What is "CAS latency"?

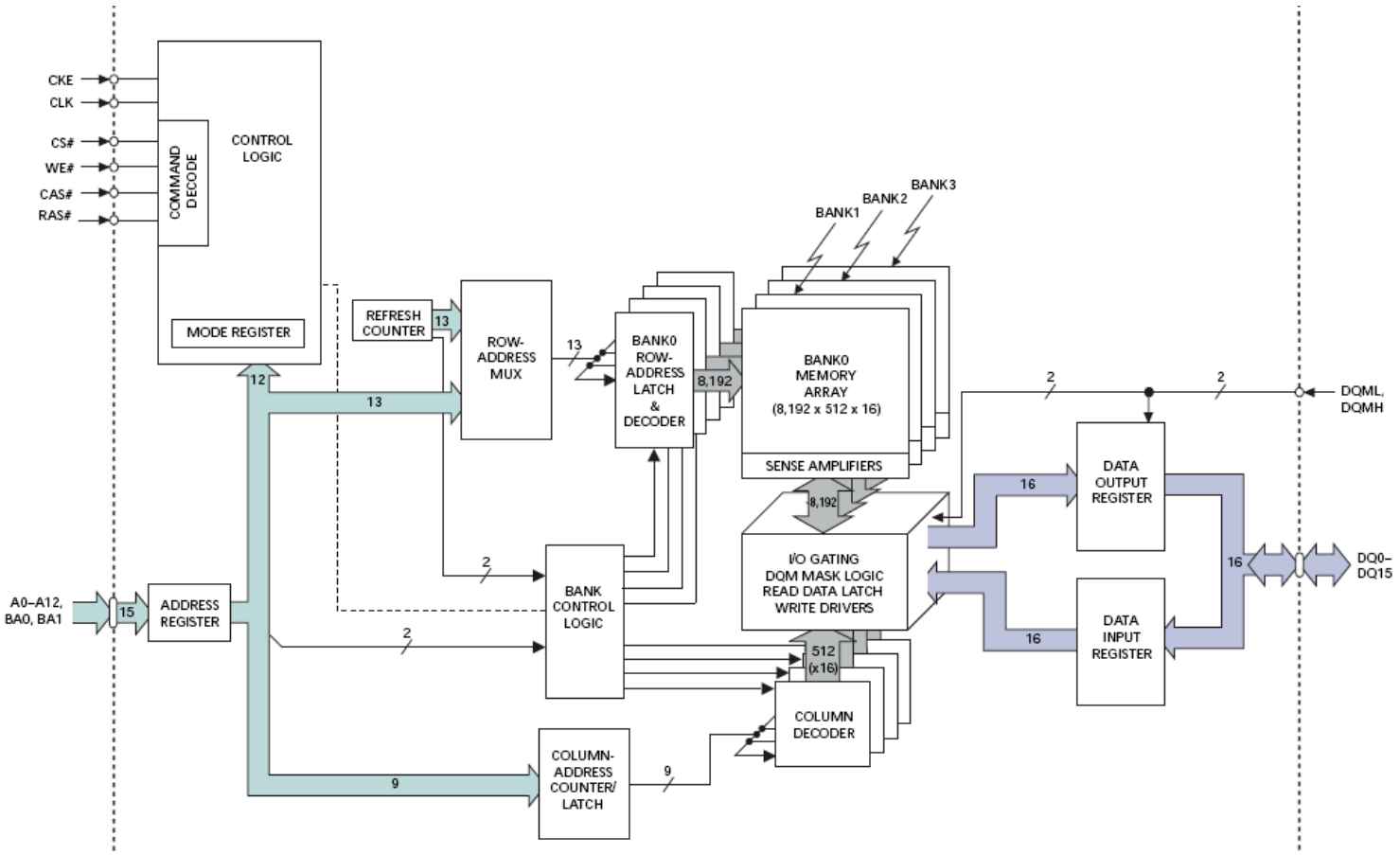
The delay, in clock cycles, between the registration of a READ command and the availability of the first piece of output data.

e. In Table 8, what do the address pins do during
   i. an "ACTIVE" command?
      Select ("activate") the *row* in a particular bank to be accessed.
   ii. a "READ" command?
      Selects the *column* location where the read will start.

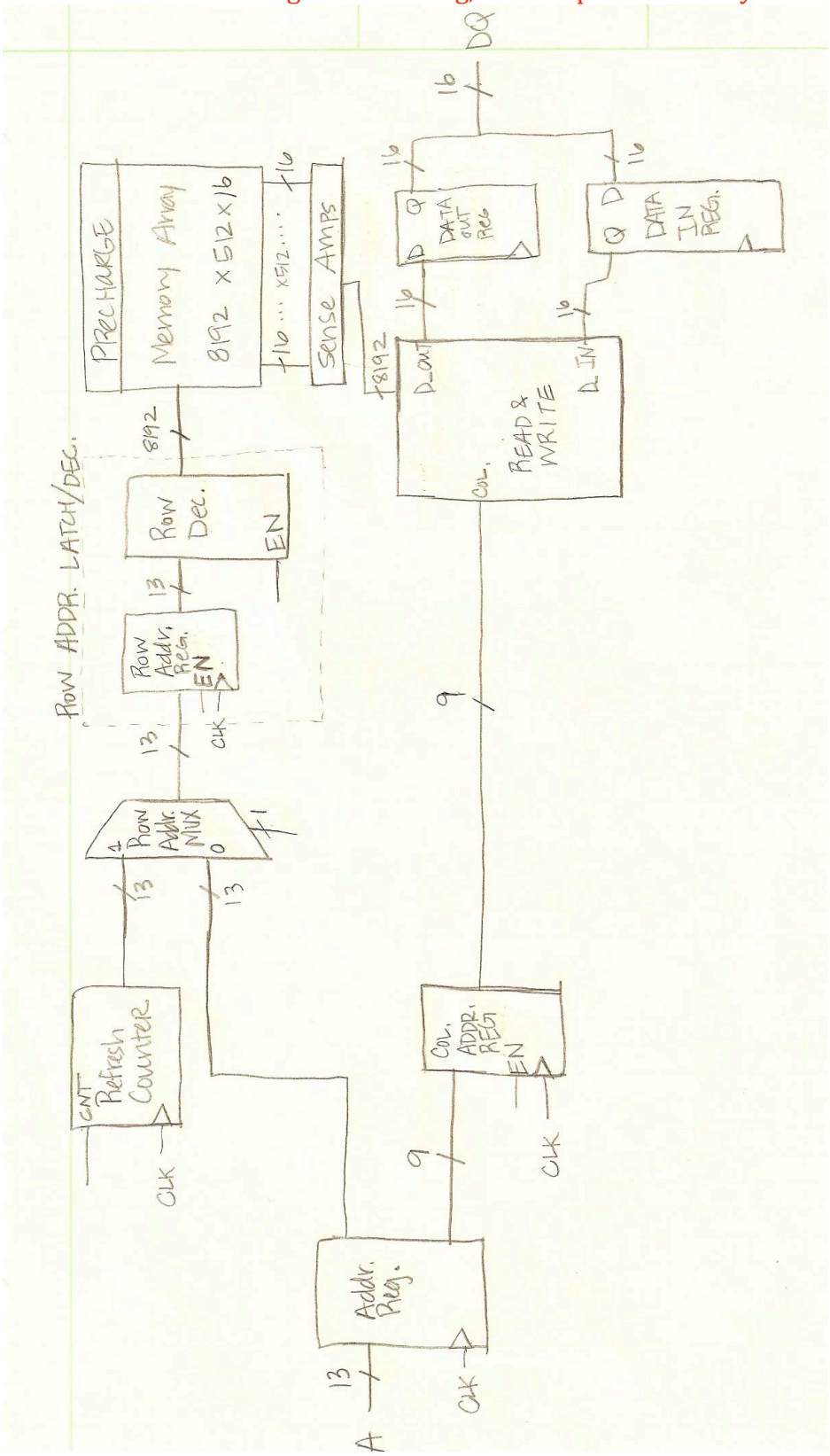
3) Design a simplified version of the chip, down to the level of registers, MUXs, encoders/decoders, etc. The bit array, with sense amps and pre-charge circuitry, may be treated as a black box. You may assume:
   a. Only one bank.
   b. You only have to implement commands ACTIVE, READ, WRITE, PRECHARGE, and AUTO REFRESH.
   c. Anything else you need to get it to work. State your assumptions clearly. CKE is tied high, DQM is tied low, Burst mode operation.

   For your design:
   a. Use Figure 3 as your top-level block diagram.
   b. Refine each of the blocks in that diagram
   c. Make a bubble diagram of the state machine (you don't have to show the implementation).

Figure 3 from the datasheet:



Here is a high-level diagram very similar to Figure 3, but showing a bit more detail on the Row and Address latching and decoding, and simplified for only one bank.

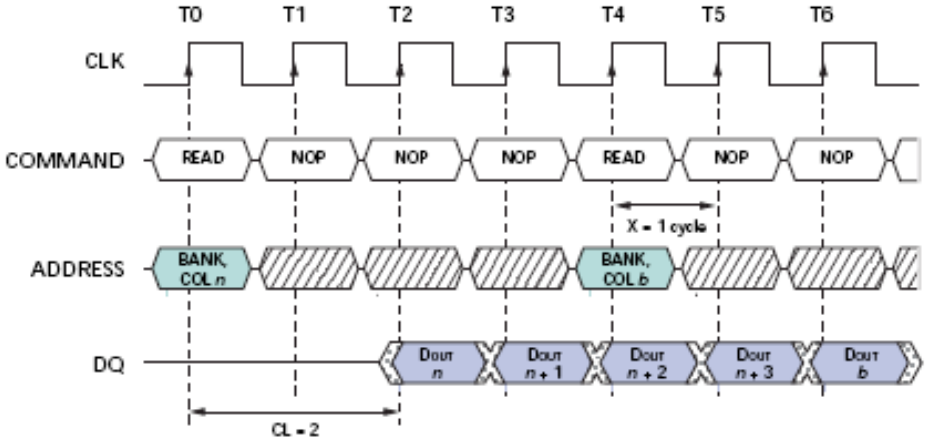


From here, we need to further break down the "Read & Write" block. We need to be able to WRITE (put data from the Data Input Register into the correct Row Register column and back into the Memory Array) and READ (put one column of one row of data from the Memory Array into the Data Output Register).

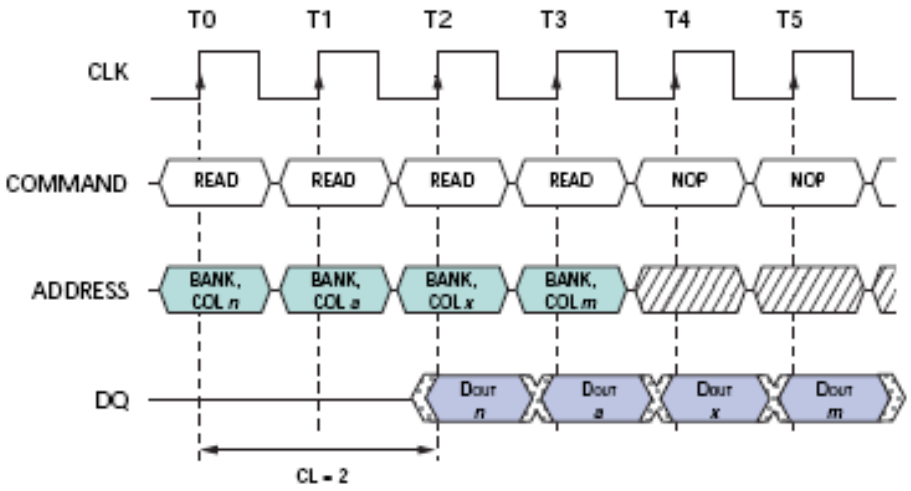
Implementing the WRITE is fairly straightforward. The Data Input Register contains 16 bits of data. We need to Activate a Row and bring its entire contents to the Row Register, replacing one of the 16-bit columns in the Row Register with the new 16 bits of data. This indicates that we'll need a MUX at the input to the Row Register. One input of the MUX will come from the sense amps (existing contents of the memory array) and the other input will be from the Data Input Register. We connect the 16 bit input to a MUX on each of the 512 columns in the Row Register, then we use the Column Decoder output as the MUX select signal to choose which of the 512 columns gets the new 16 bits of data. The complete Row Register is then connected back into the bit lines of the memory array and the new Row is stored.

A READ is a bit more complicated. Notice the Read timing specs on pages 28-29 of the Micron datasheet, shown below. The first block of data comes out after a 2 cycle delay, as you would expect. Subsequent blocks, however, are output on consecutive clock cycles, regardless of whether it was a sequential burst read or a random access read. This tells us that they have done some clever design work to speed up the Reads.

Consecutive READ Bursts



Random READ Accesses



The simplest way to get data from the Memory Array to the output register would be to have a single 512:1 MUX on the output of the Row Register, controlled by the Column Address. The output of the MUX would be the 16 bit column from the desired Row, and would be stored in the Data Output Register. The problem with this simple implementation is that a 512:1 MUX is VERY slow. To implement a 512:1 MUX, it takes 9 stages of 2-input MUXes! In order for the signals to propagate all the way through the 9 stages in one clock cycle, the clock has to be run very slowly. We can break up the size of the MUX to speed up the Read operation.

Instead of a single 512:1 MUX, we'll split the 9 bits of the column address into control signals: the upper 5 bits – Col[8:4] – will control a 512:16 MUX. This will take 16 of the original 512 columns from the Row Register and store it in a Partial Row Register. The lower 4 bits of the column address – Col[3:0] – are loaded into the Partial Column Address Counter at the same time the 16 columns get latched into the Partial Row Register. The lower 4 bits of the column address are then used as the control input to a 16:1 MUX to select a single column from the Partial Row Regsiter. This single column is then latched into the Data Output Register.

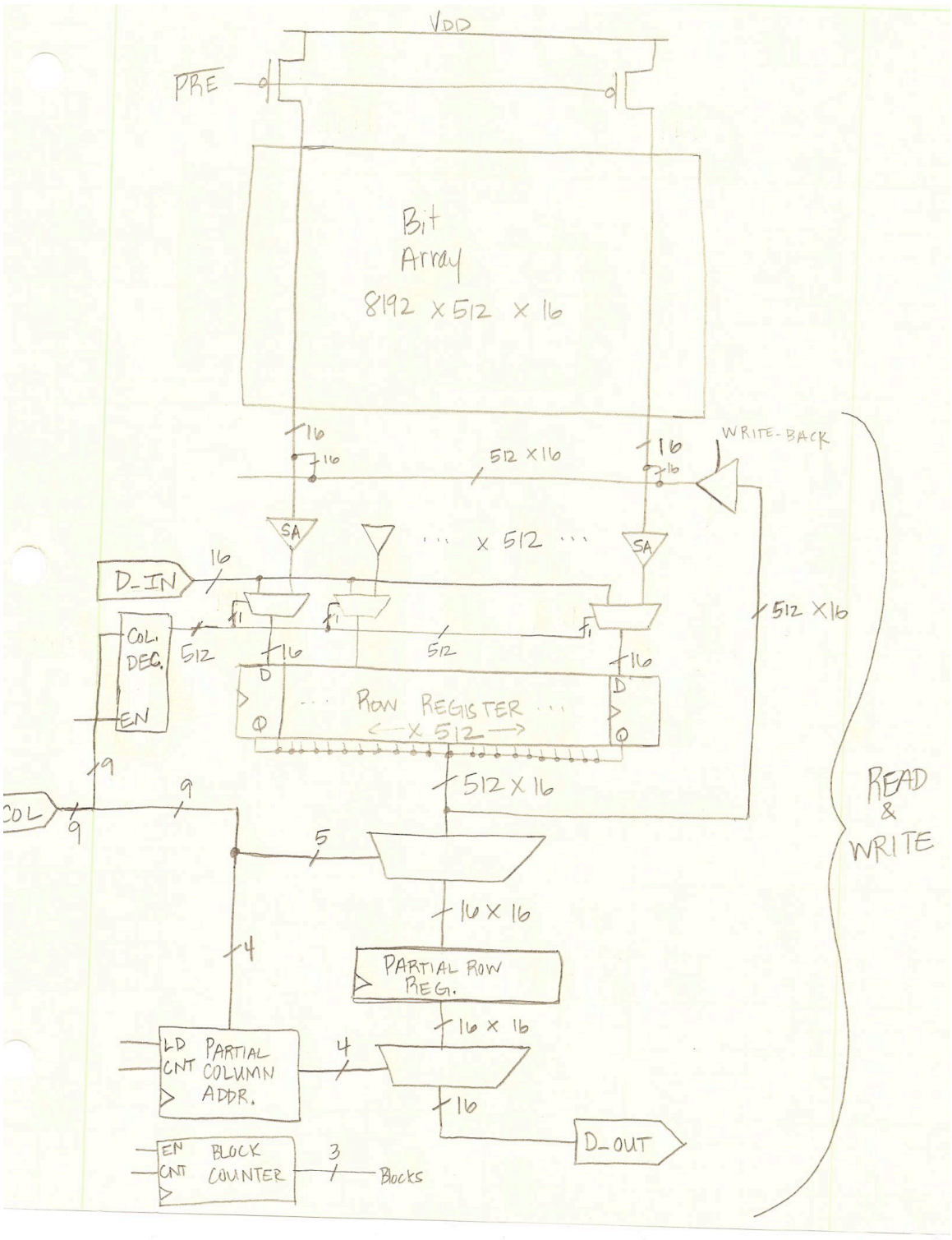
This design allows for much faster *sequential* reads because the Partial Column Address Register can simply be incremented, and a new column of data (already stored in the Partial Row Register) can be latched into the Data Output Register on the next clock cycle.

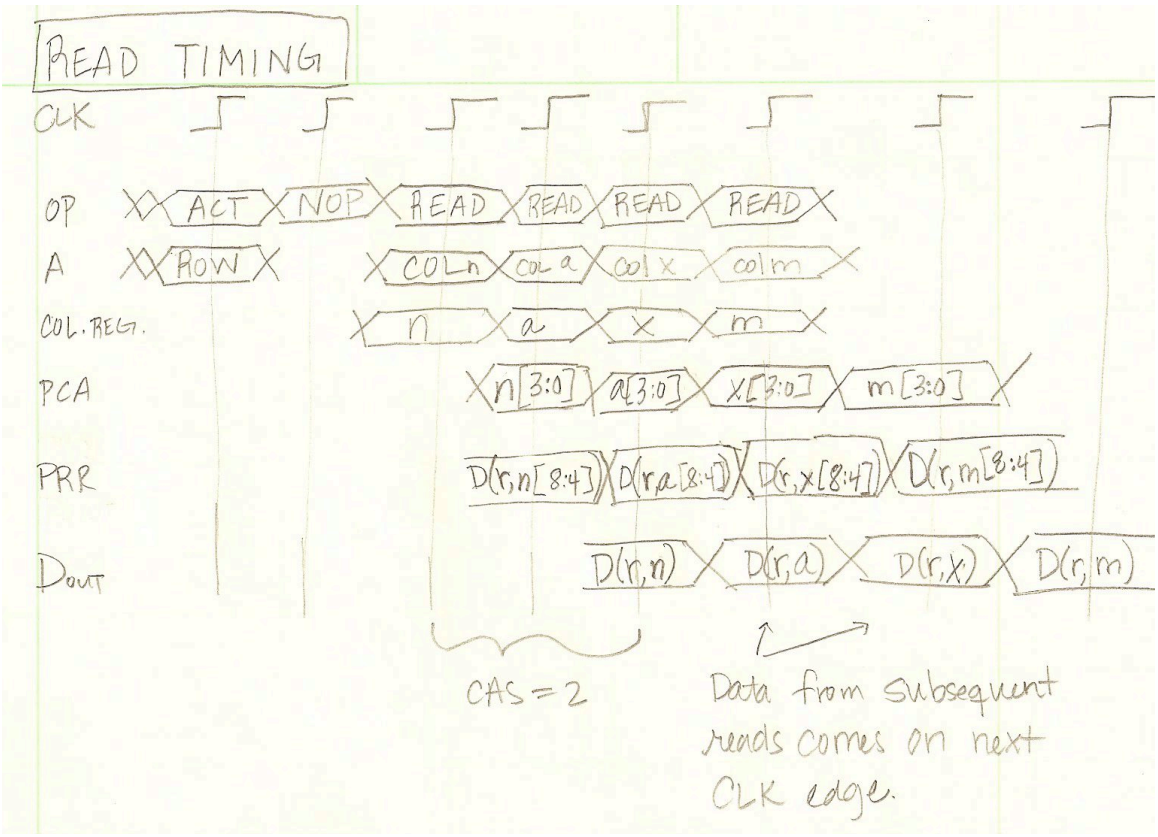
This design also speeds up *random access* reads, which is slightly more complicated. Refer to the timing diagram above for Random Read Accesses. Four Read commands occur on four consecutive clock cycles, and the columns to be read are not adjacent. We cannot simply increment the counter as we did for sequential reads. The entire row has been activated and stored in the Row Register.

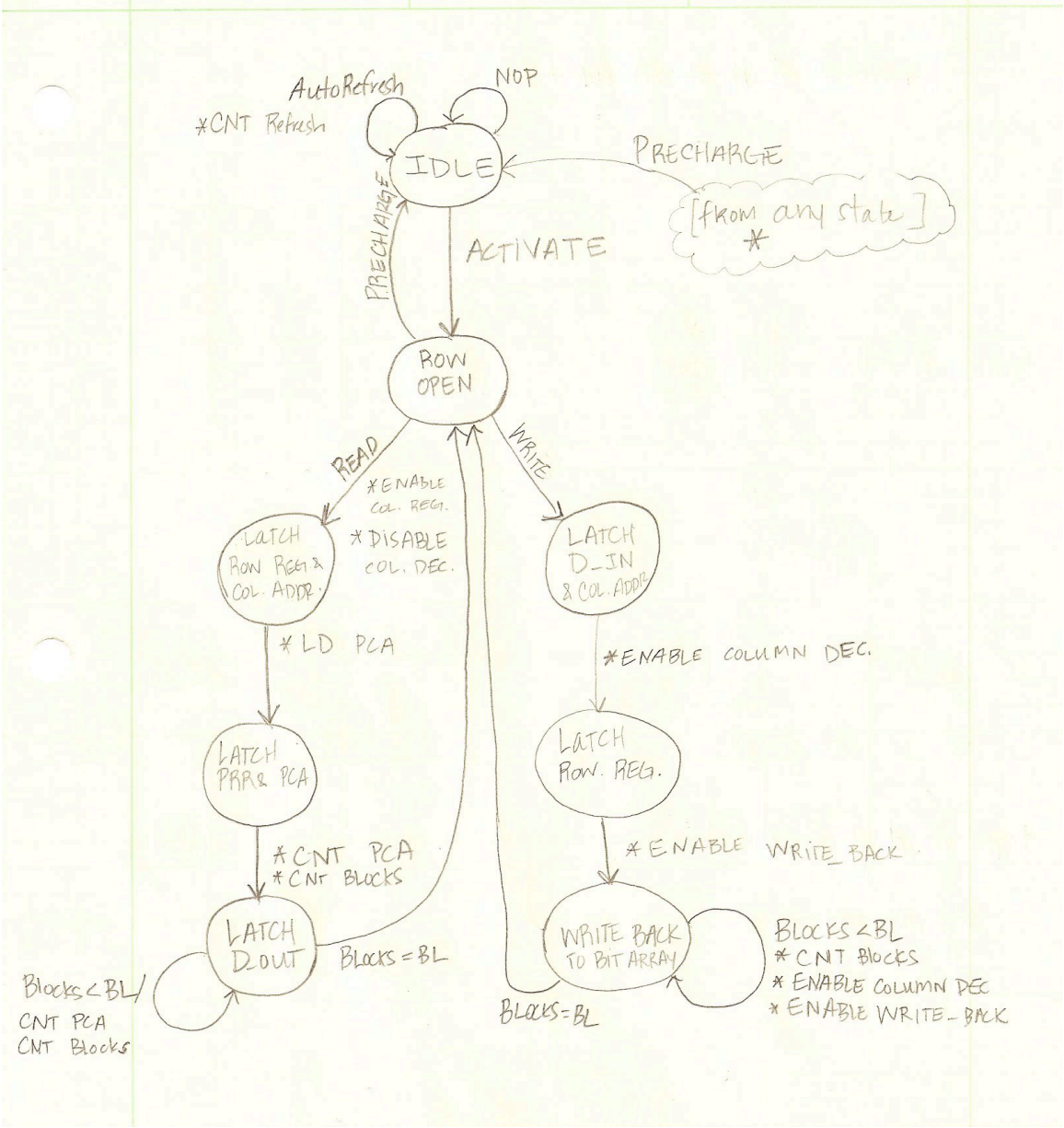
At T=0: -- the first Read command for column *n* is issued
-- 9 bit column address *n* is latched into the Column Address Register
At T=1: -- the lower 4 bits of *n* are latched into the Partial Column Address
-- 16 columns selected by *n*[8:4] are latched into the Partial Row Register
-- a new value, *a*, is latched into the Column Address Register
At T=2: -- Column *n* is latched into the Data Output Register
-- the lower bits of *a* are latched into the PCA
-- 16 columns selected by *a*[8:4] are latched into the PRR
-- a new column value, *x*, is latched into the Column Address Register
At T=3: -- Column *a* is latched into the Data Output Register
-- the lower bits of *x* are latched into the PCA
-- 16 columns selected by *x*[8:4] are latched into the PRR
-- a new column, *m*, is latched into the Column Address Register
etc….

Notice that the data is always available at the output 2 cycles after the column address was read.

The following figures show the details of the Read & Write block, as well as the Read Timing Diagram for this implementation. Finally, the FSM shows the control signals for the system.


VDD
PRE
Bit
Array
8192 x 512 x 16
16
16
512 x 16
16
16
WRITE-BACK
SA
... x 512 ...
SA
16
D_IN
512 x 16
COL.
DEC.
512
512
16
16
D
Q
ROW REGISTER
← x 512 →
EN
9
512 x 16
COL
9
9
5
READ
&
WRITE
16 x 16
4
PARTIAL ROW
REG.
16 x 16
LD
CNT
PARTIAL
COLUMN
ADDR.
4
16
D_OUT
EN
CNT
BLOCK
COUNTER
3
Blocks


READ TIMING
CLK
OP
ACT
NOP
READ
READ
READ
READ
A
ROW
COLn
col a
col x
col m
COL. REG.
n
a
x
m
PCA
n[3:0]
a[3:0]
x[3:0]
m[3:0]
PRR
D(r,n[8:4])
D(r,a[8:4])
D(r,x[8:4])
D(r,m[8:4])
DOUT
D(r,n)
D(r,a)
D(r,x)
D(r,m)
CAS = 2
Data from subsequent reads comes on next CLK edge.


AutoRefresh
*CNT Refresh
NOP
IDLE
PRECHARGE
[from any state]
*
PRECHARGE
ACTIVATE
ROW OPEN
READ
WRITE
*ENABLE COL. REG.
*DISABLE COL. DEC.
LATCH ROW REG & COL. ADDR.
LATCH D_IN & COL. ADDR
*LD PCA
*ENABLE COLUMN DEC.
LATCH PRR & PCA
LATCH ROW. REG.
*CNT PCA
*CNT BLOCKS
*ENABLE WRITE_BACK
LATCH D_OUT
WRITE BACK TO BIT ARRAY
Blocks<BL
CNT PCA
CNT Blocks
Blocks=BL
Blocks=BL
Blocks<BL
*CNT Blocks
*ENABLE Column DEC
*ENABLE WRITE_BACK