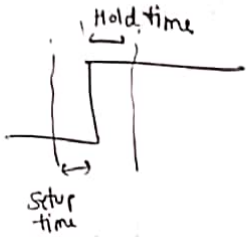
CS220 (16/01/2020)




setup time is the largest
hold time smallest

$t_{prop} = t_{clock\text{-}to\text{-}Q}$ (close to setup time)


clock skew : the same clock edge may not arrive at both x and y at the same time

- $t_{cycle} \geq L_f$ (Latency through f)

But now we say

$$t_{cycle} \geq t_{prop} + L_f + t_{setup} + t_{skew}$$

($t_{prop}$: propagation delay)

$L_f$ is the max delay in f to give output (f may have many routes for example full adder has two paths sum/carry)

Explanation: $t_{prop}$ was added because this is the propagation error through x.

$t_{setup}$ came because y wants f's output to be ready $t_{setup}$ time before the clock edge

- We are assuming that $t_{setup}$, $t_{prop}$ and $t_{hold}$ is same for every flip flop because we can control that by design
- $t_{skew}$ is tricky because ff may be at diff distances so we take max($t_{skew}$)

Case: Clock edge at x arrives $t_{skew}$ after clock edge at y.

The input arrive late by $t_{skew}$ thus for the computation to get done we should add $t_{skew}$ because we decreased the clock time by $t_{skew}$.

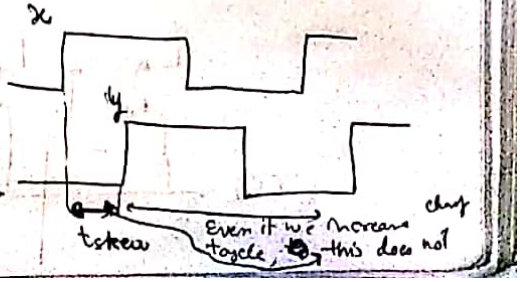
Case: clock edge at x arrives $t_{skew}$ time before clock at y:

Input comes early. f computes and by the time clock reaches y, new output is stored & previous value is lost

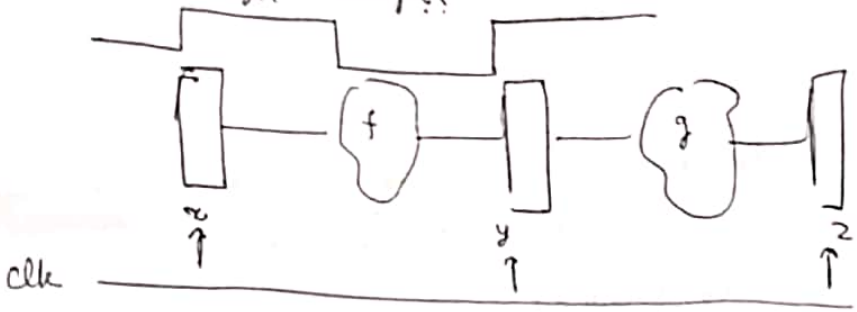
$$\rightarrow t_{skew} \geq t_{prop} + t_f^{min} + t_{setup}$$ (This is when real problem starts arising)

To use this we have to give a delay at x to our flip flop

It has to be done along with lengthening the cycle time. And where ever the delay problem arises you have to add delay at those particular places explicitly



- Latch - level-sensitive. changing input when clock is high affects the output
- Trick to design. Why ??

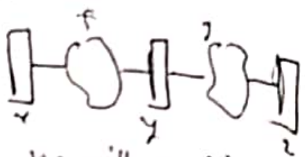



Problem occurs when f computes in less than half cycle and g takes full cycle.

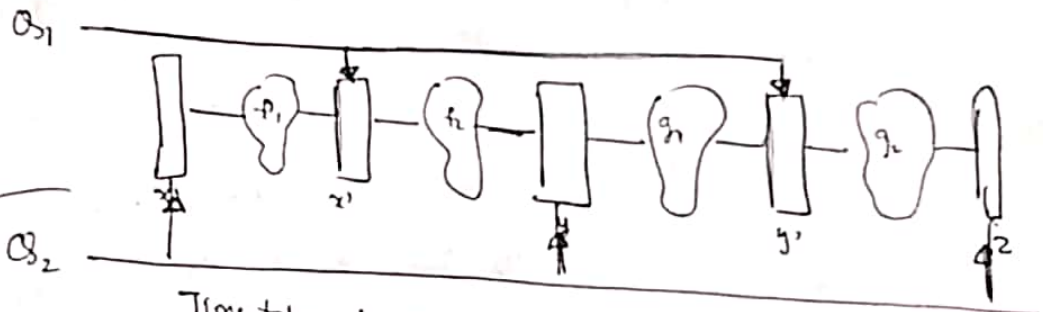
Also f and g take different times depending on whether x is 0 or 1

- In such case, it would not be possible to even infer what is latch
- Use of Two phase clock :

Earlier




we will modify this to.


Time taken to compute is same except that some prop. error is added

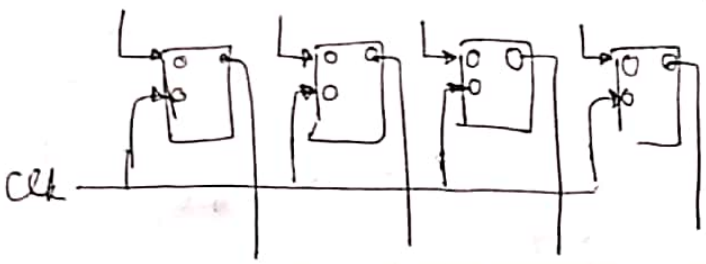
Also flip flop take 2x latches

- This design also take 2x latches design thus size is same

We'll not use latches

- If we use blocking assignments in always block for sequential logic, it create a latch. Avoid using that.

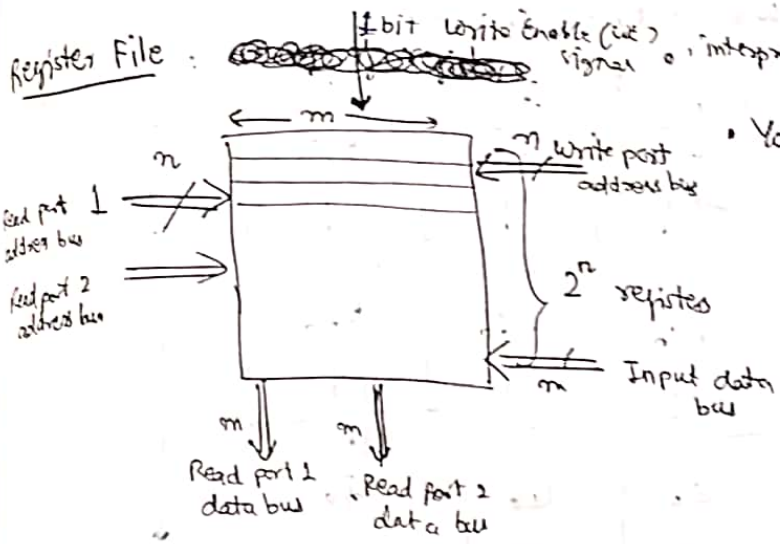
REGISTER: an array of flip flops (each storing a bit))



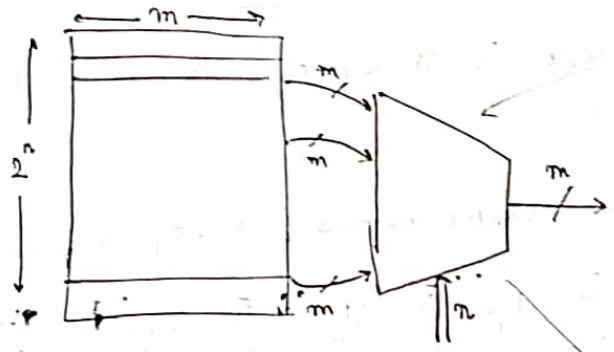

4-bit register

Reading on register doesn't require an clock but writing does

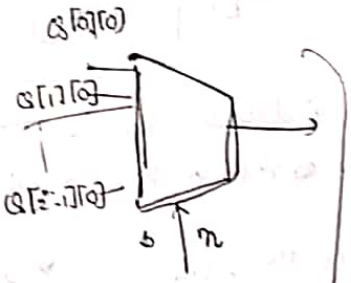
Register File :




- You need multiple ports for speeding computing.
  Ex. Suppose you want to add data of two registers




Mechanism of reading




Actual implementation

We use m 1 bit multiplexer where first choose from first bits of all register.
2nd choose from second bits of all register

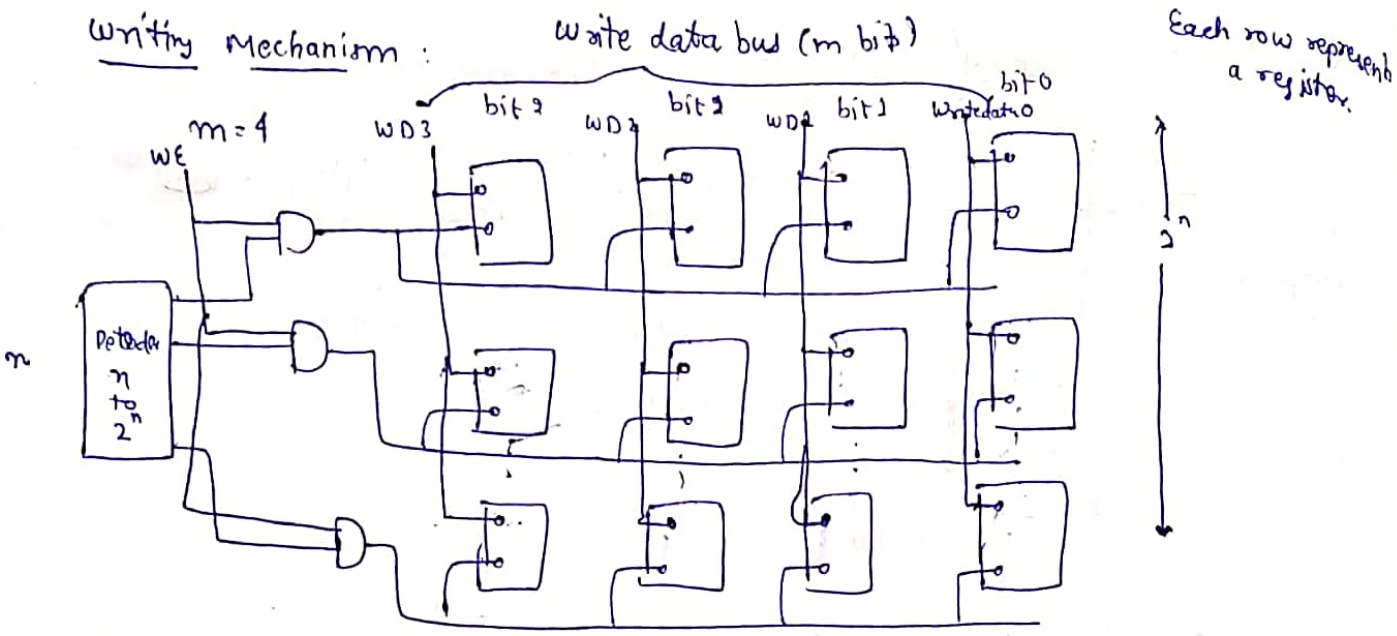
This is huge bulky and the combinational formula will be huge & deep

Also note that this is not a very nice way of reading

- Reading from a register file is combinational
- Writing is sequential

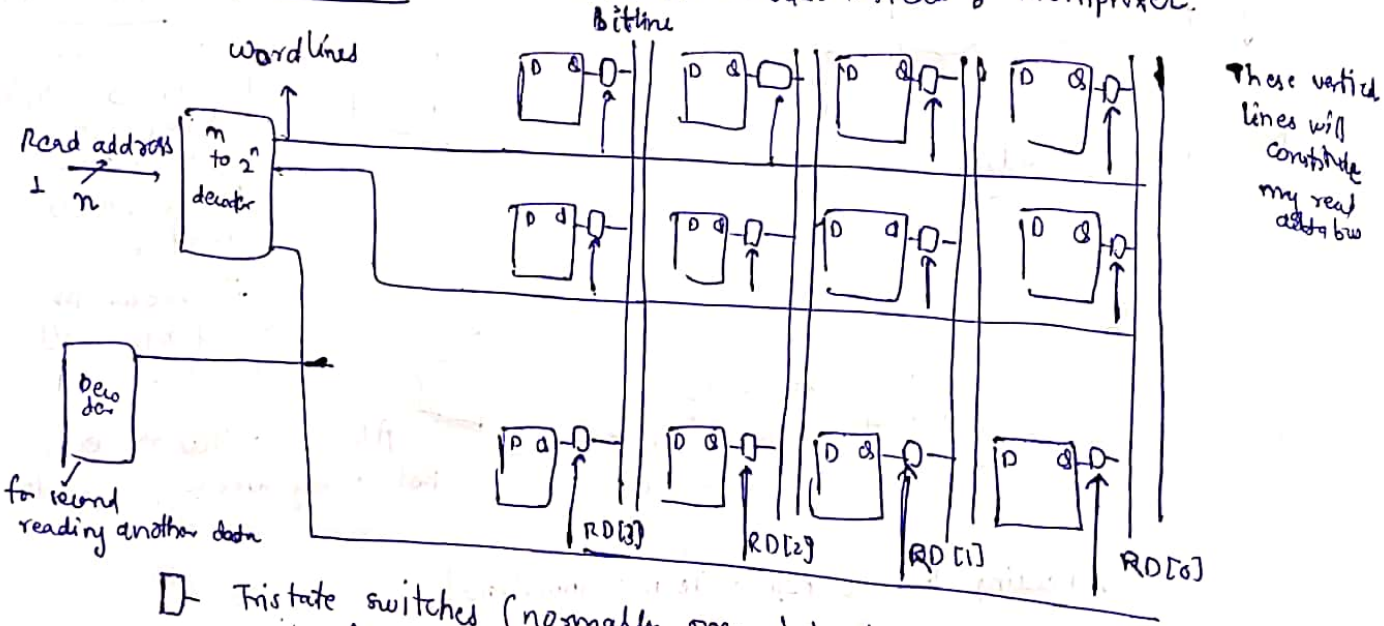
22/01/2019

Writing Mechanism :




You need a decoder for every write code. If there is a another write address then you should have another decoder.

Better Reading Mechanism : We will use decoders instead of multiplexer.




Tristate switches (normally open. but when a high voltage is applied it closes and output of Q is passed onto output

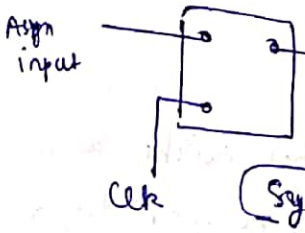
Why is this faster ? Decoder has $2^n$ lines which work in parallel and thus it is fast.

But multiplexer first decode the select lines and work in sequential manner. Thus it is slow., logic takes more time to be computed

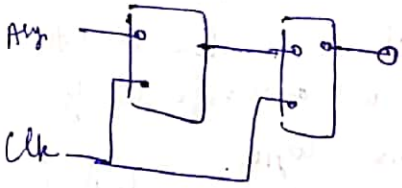
Adding new ports to read increase the area in quadratic fashion in record model. Also more wire mean more R and C and thus RC delay will be high. Thus you cannot increase the ports indefinitely because that will slow down.

Thus there must be an optimum no. of ports you can use

**Asynchronous Input** : input are not supp. to change with clock. Input might change during the hold time



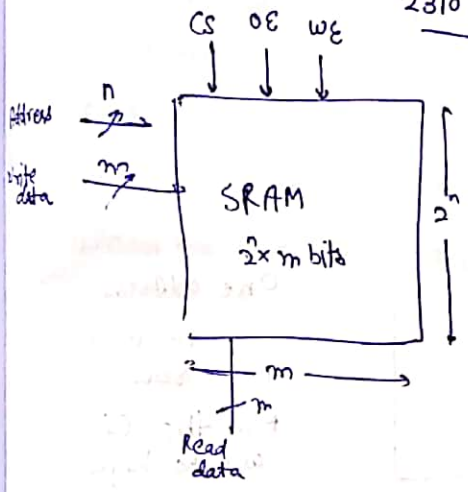

→ if and only if the asy. input does not change in the setup time + hold time.




- You use the output after some time the end of hold time
- If a metable stable state does not resolve in one cycle, then add more flip flops

---

23/01/2020




$2^n$ rows each m bits

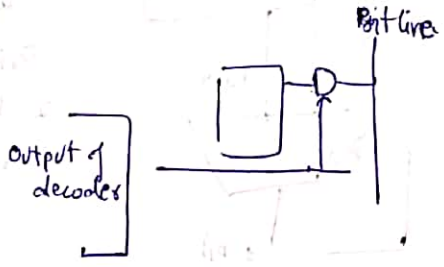
You send address that you want to read/write

WE high when you write

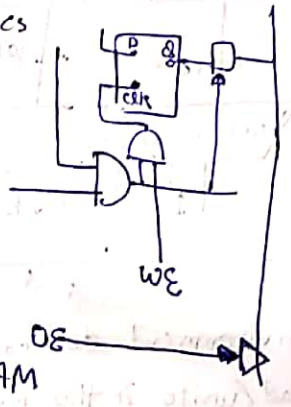
CS must be high while both write/read

OE is high while reading

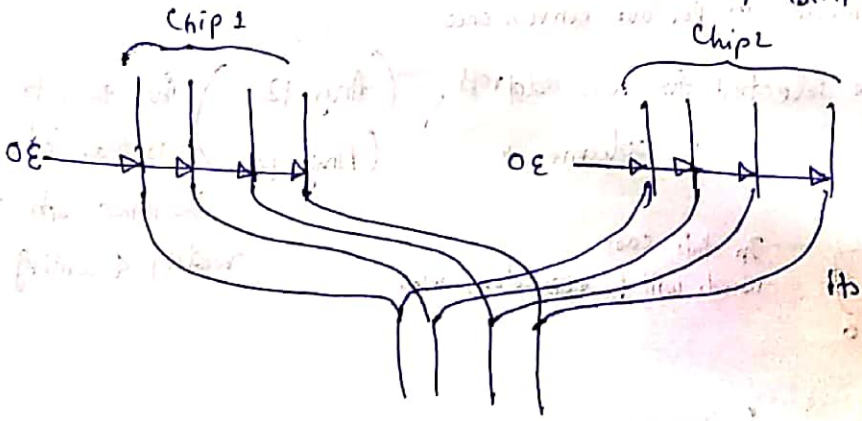
Structure is similar to the register file




What we can do is



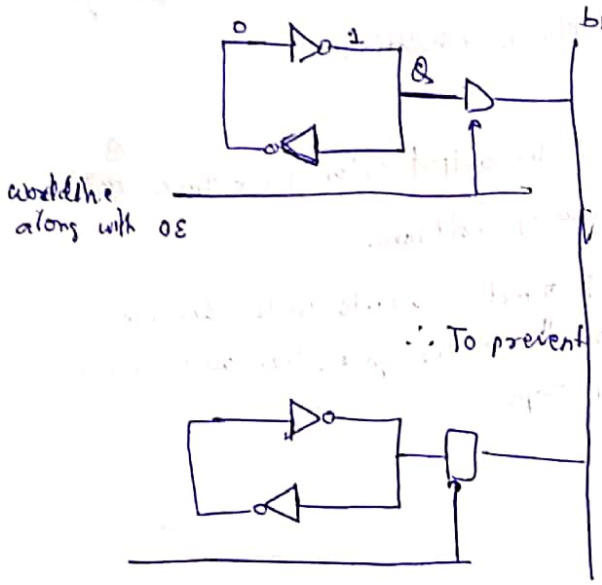

CS is used mainly when there are multiple chips SRAM




- Primary assumption is that at most one OE is high. Whichever is high its output will be displayed in output bus

Register file had separate address bus for reading & writing. Here you have common address bus. Thus you can only read/write at a time. If you want to do both add one more address line

- Each bit is stored as 2 NOT gates



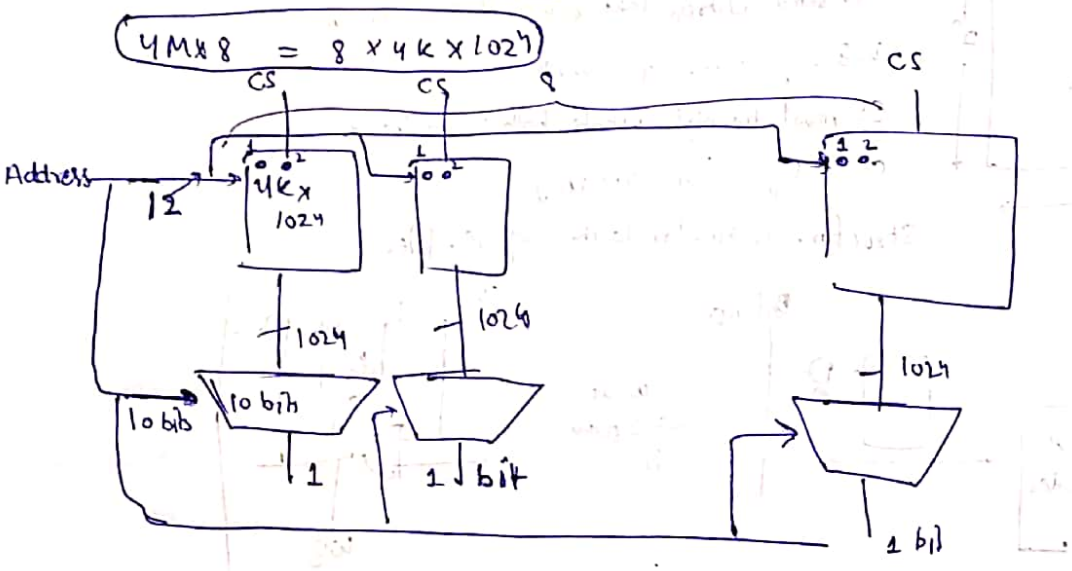

Problem here is that switches are not ideal. When they're open they are not ideally open, sneak path are there through which charge can flow to bitline slowly. It will discharge slowly. → LOSS OF MEMORY

∴ To prevent this we add additional logic.

Although the inverters will try to maintain the charges but still it can decay

## BUILDING AN SRAM

4M x 8 = 8 x 4K x 1024




One address for same row.

All the CS will be high

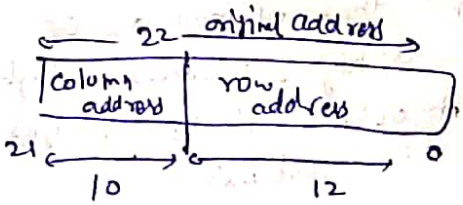
Originally you got 22 bits and you give 8 bits

Environment does not have to know what we have done.

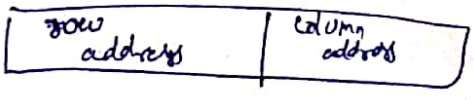
We read/write in this model as per our convenience

Which 12 bits should be selected for row address & which 10 bit " " " column. (Any 12) (Any 10)

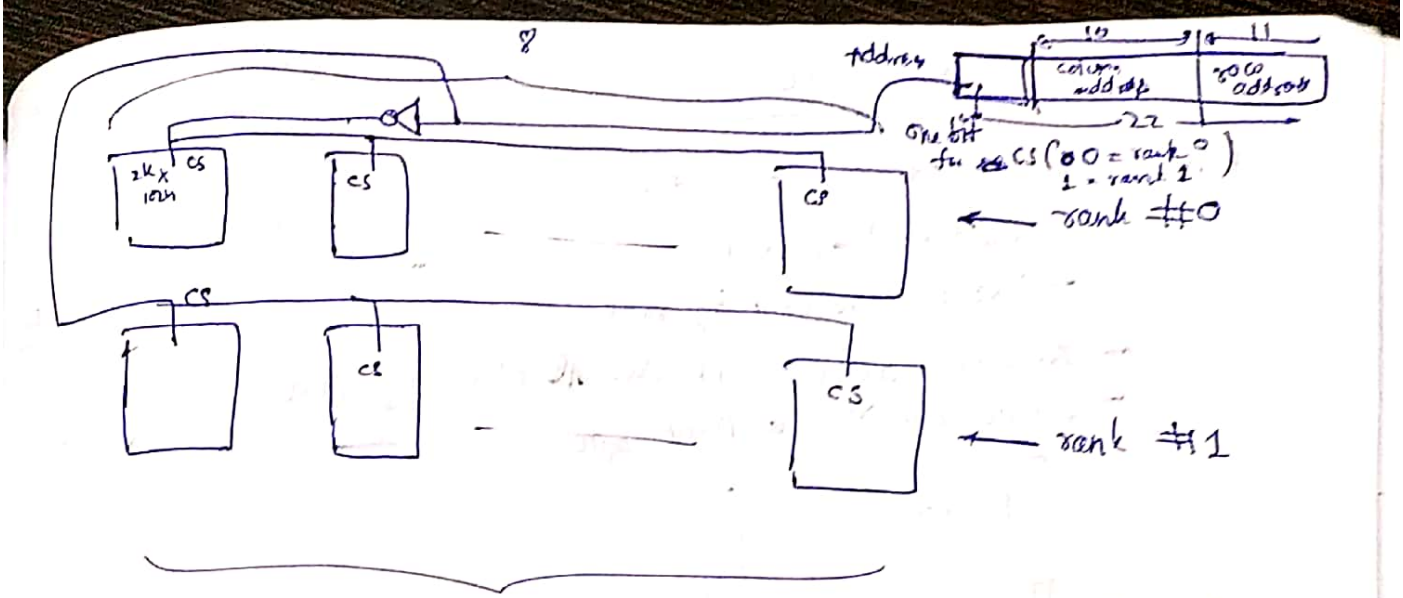
But thing to note is that you do the same both while reading & writing.




In this case words will be column wise




In this case words will be stored row wise




- Increasing rank also comes at the cost of area.
- When we'll operate we'll do it only on one rank.
- Probably we haven't saved much in area. But we have saved a lot in time We have brought a lot of parallelism in our circuit: These 8 chips run together.

But can you keep on diving into more chips? No, because for every column you add a MUX as well so this will restrict the speed.

Therefore after a point of time, both more area & more time will be consumed.

How many chips to use? You will get a formula.

When you are using subset of chips like on top the pages, we are using same time and area but power consumption is half.

Therefore we have to optimize: performance x area x power consumption.

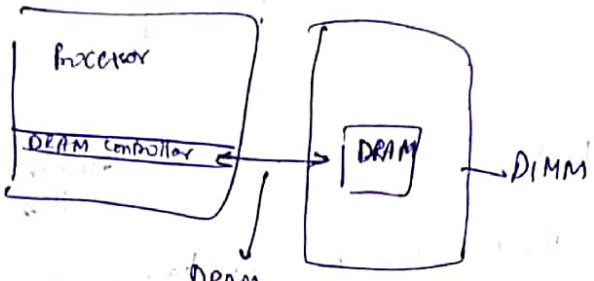
---

Dynamic Random Access Memory (DRAM)

24/01/2020

## DRAM. – a bit stored a capacitor

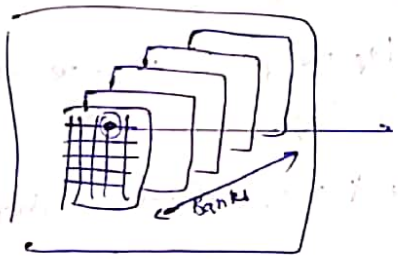
- must be "refreshed"
- "refresh" is rank by rank
- You cannot access a rank while it's being refreshed
- Each address refers a DRAM byte.




[ sometime ppl only refer to this as the DRAM Channel

- DRAM does consumes a lot more power than SRAM

## ORGANIZATN OF DRAM CHIPS :




This intersection is a bunch of bits, width of the output

X4 – gives 4 bits
X8 – 8 bits
X16 = 16 "

- Example : 512 Mbit 4 banks, 16k rows, 1k columns

$2^{29}$ bits each bank = $2^{27}$ bit

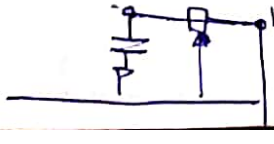
∴ $16k \times 1k = 2^{24}$ bit thus, each intersection = $\frac{2^{27}}{2^{24}}$ = 8 bits

(width of each column)

Thus X8.

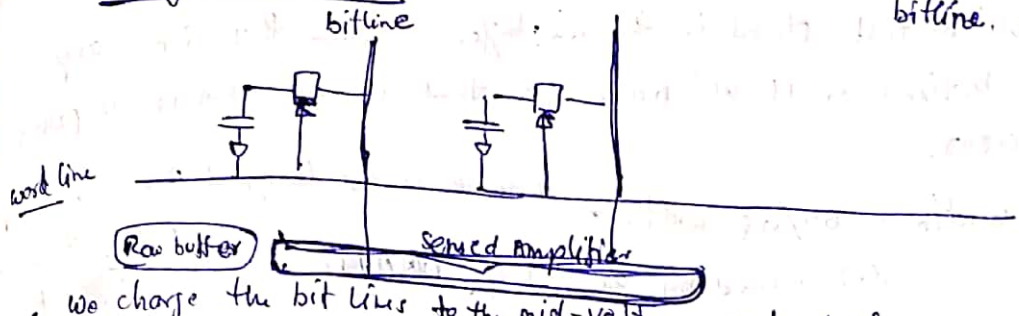
> Output width = Bank Capacity / (Rows X Columns)

- Every bank will have a decoder.
  - Switch is bidirectional (both reading & writing through a single bitline.




we donot have to worry about the read/write. charge will flow in appr. direction

- Reading from DRAM:




- Sense amp does not change the charge on bitline.

The whole row will be activated (of size 16k as in previous example)

- We charge the bit lines to the mid-voltage beforehandedly

So when 0 – 0V
1 – 2V
bitlines are charged to 1V

and when capacitor is uncharged charge moves from bitline to capacitor
" " " " charged " " " capacitor to bitline

Since at stable condition, bitline won't have 0 or 2V but some middle pt we have an amplifier which senses the direction of flow of charge and according make the bitline 0 or 2V.

- Charge flow takes some time. Precharging takes time, after that you activate the row line.

row's content is destroyed but sense amplifier stores the data of that row.
Subsequent reads will be done from the sense amp and not the bitline

- We can read a bunch of consecutive column. N – burst length fixed, not arbitrary. We can read these no. of columns with only one column address

BC = 1 BN = 2, 4, 8

Precharge Activate CAS (Take equal times)

- We can avoid precharge & activate again and again if we read from the same row because its contents are stored in row buffer.

Therefore go row by row, o/w we will have to charge the rows to their original values after every single step.

- Reads from the same row should be clustered together. To be done by DRAM controller

Increase performance by 3 times

DRAM Access Scheduling:

FR – FCFS

First ready — row buffer is ready.

First come First serve.