Digital Integrated Circuits
(83-313)

Lecture 8:
Memory Peripherals

Semester B, 2016-17

Lecturer: Dr. Adam Teman

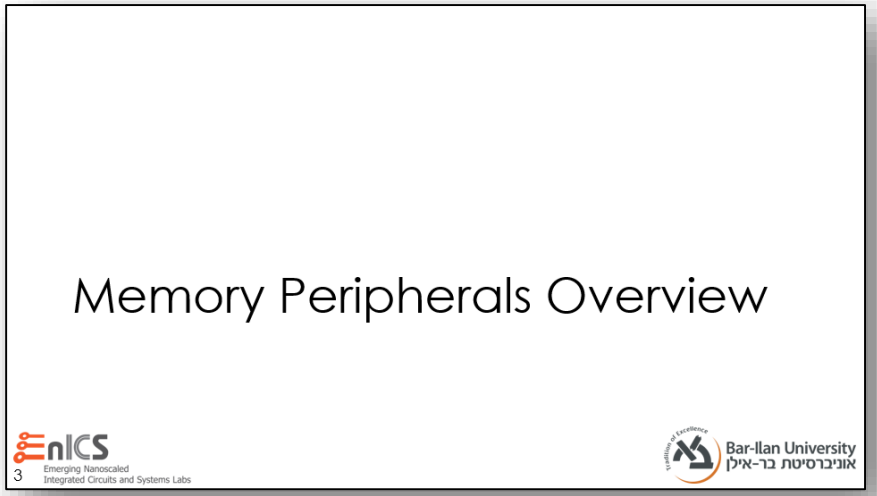
TAs: Itamar Levi,
Robert Giterman

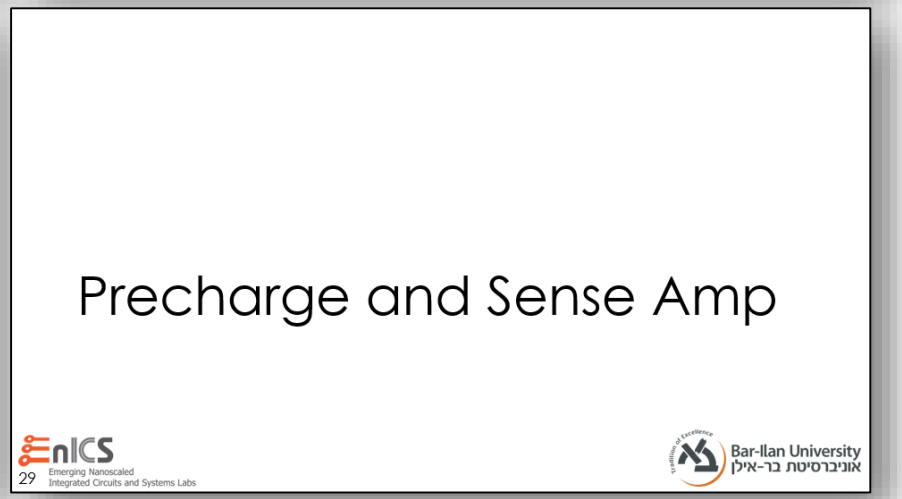
20 May 2017

Emerging Nanoscaled
Integrated Circuits and Systems Labs

Bar-Ilan University
אוניברסיטת בר-אילן

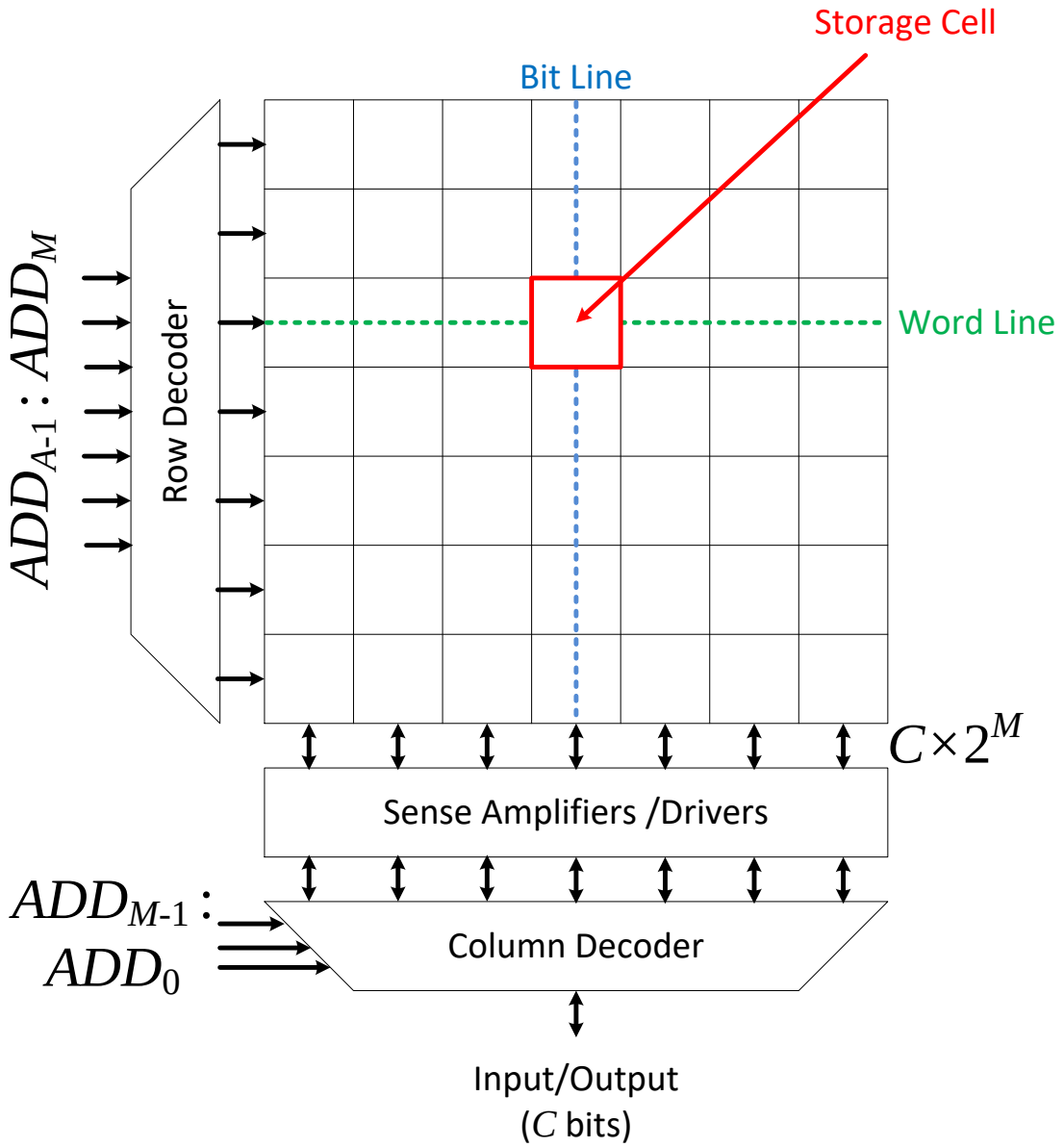
Lecture Content





Memory Peripherals Overview

Memory Architecture



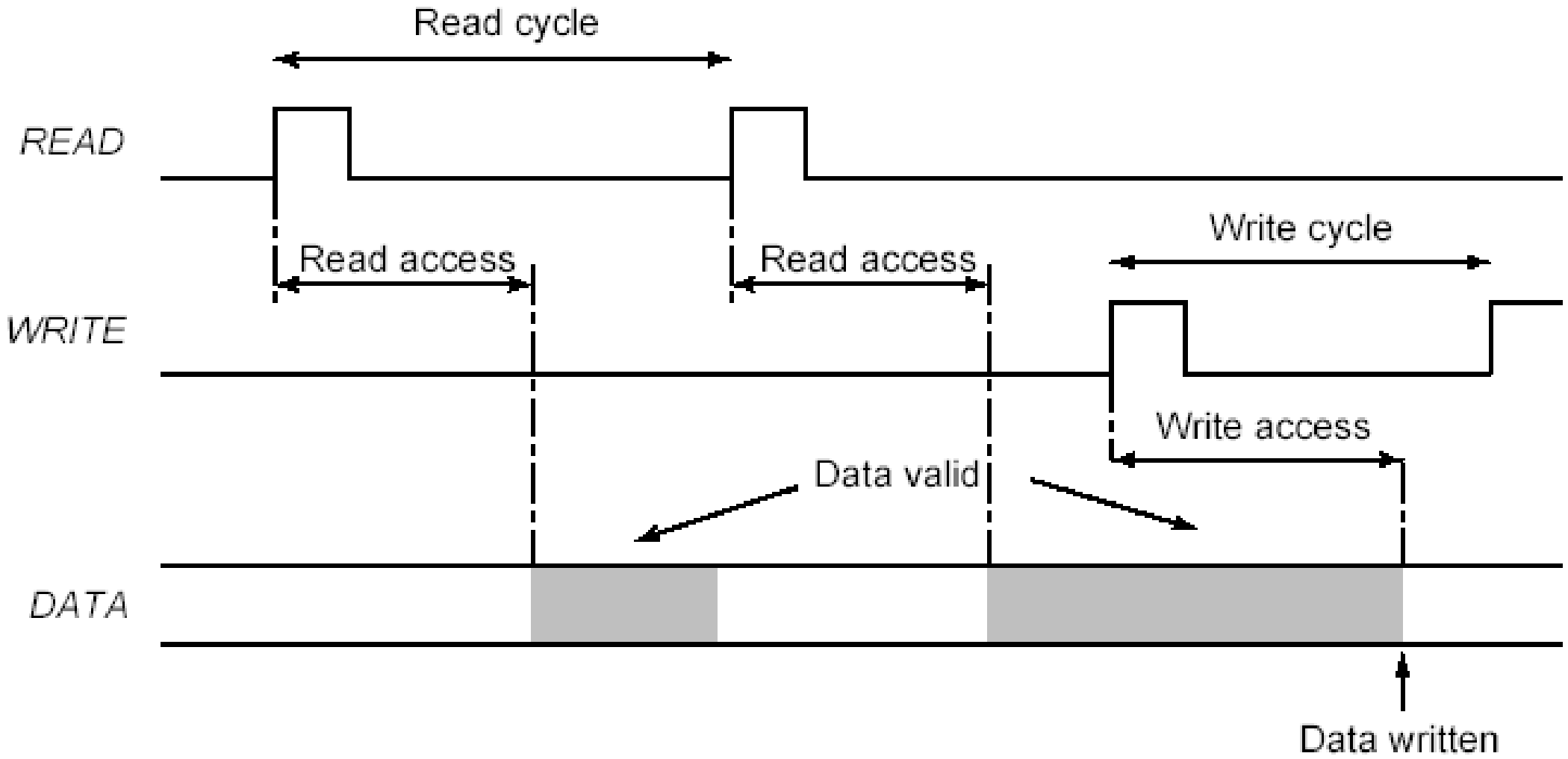
Memory Size: W Words of C bits
$= W \times C$ bits
Address bus: A bits
$\rightarrow W = 2^A$

Number of Words in a Row: 2^M
Multiplexing Factor: M

Number of Rows: 2^{A-M}
Number of Columns: $C \times 2^M$

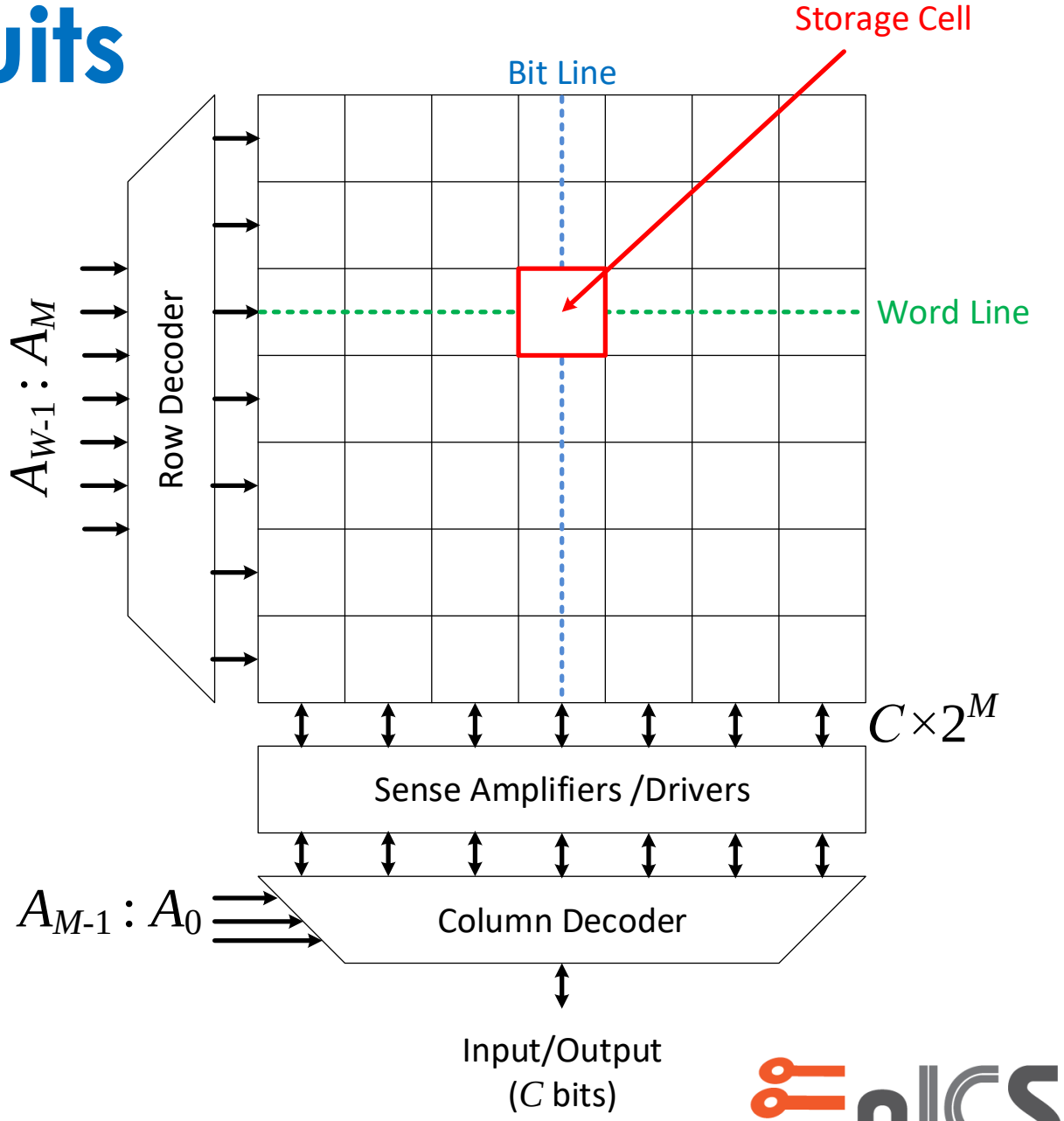
Row Decoder: $A\text{-}M \rightarrow 2^{A-M}$
Column Decoder: $M \rightarrow 2^M$

Memory Timing: Definitions



Major Peripheral Circuits

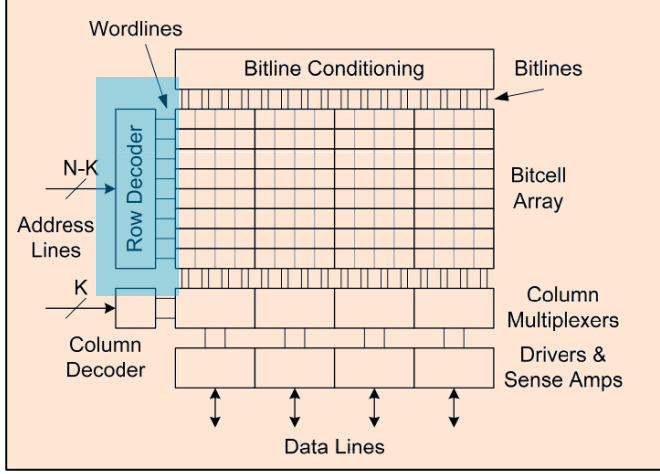
- Row Decoder
- Column Multiplexer
- Sense Amplifier
- Write Driver
- Precharge Circuit

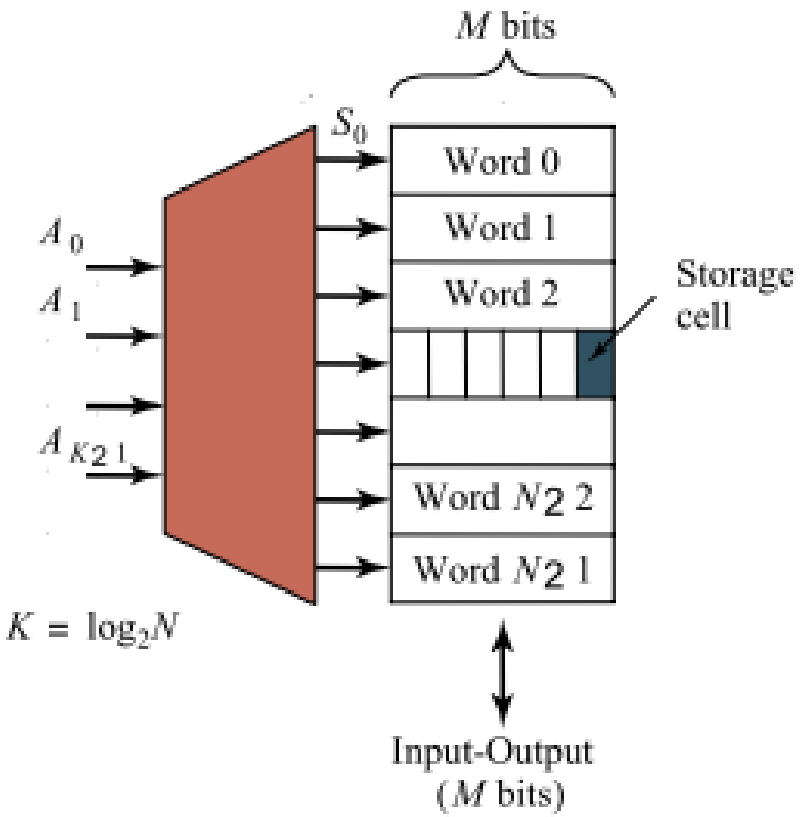


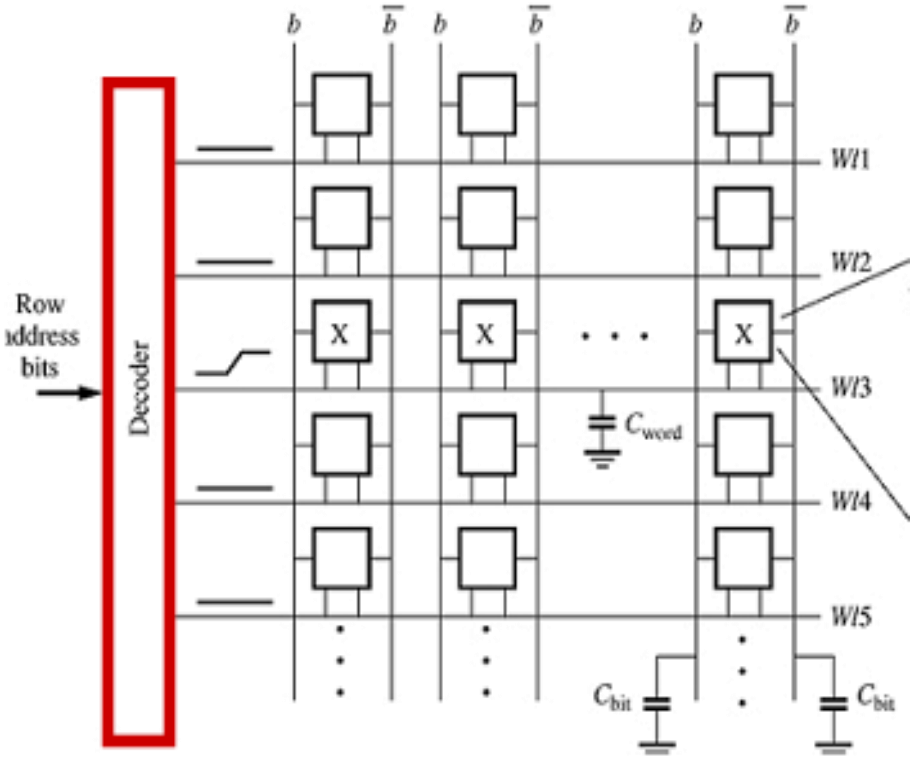
Row Decoder Design

Row Decoders

- **A Decoder reduces the number of select signals by $\log_2$.**
  - Number of Rows: N
  - Number of Row Address Bits: $\log_2 N$







Row Decoders

- **Standard Decoder Design:**
  - Each output row is driven by an AND gate with $k=\log_2 N$ inputs.
  - Each gate has a unique combination of address inputs (or their inverted values).
  - For example, an 8-bit row address has 256 8-input AND gates, such as:

$$WL_0 = \overline{A}_7 \overline{A}_6 \overline{A}_5 \overline{A}_4 \overline{A}_3 \overline{A}_2 \overline{A}_1 \overline{A}_0 \qquad WL_{255} = A_7 A_6 A_5 A_4 A_3 A_2 A_1 A_0$$

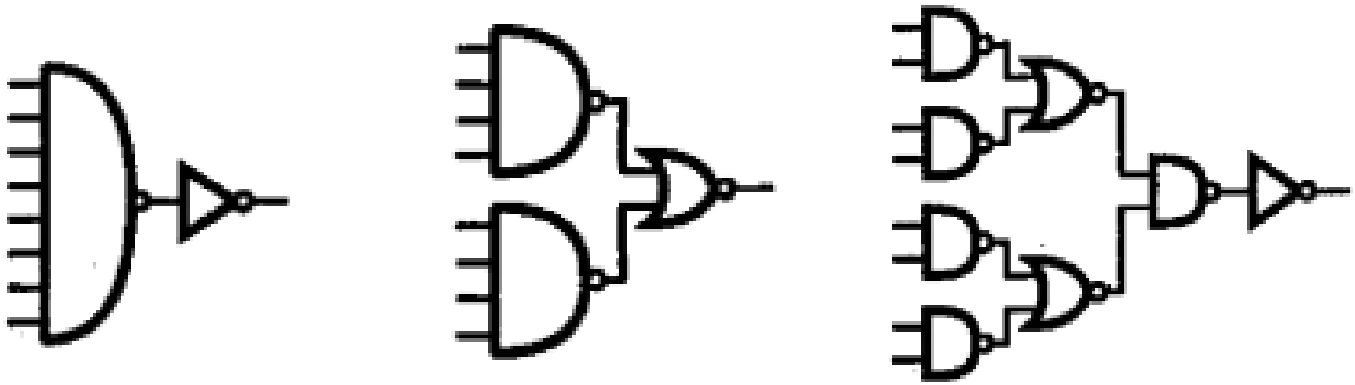
- **NOR Decoder:**
  - DeMorgan will provide us with a NOR Decoder.
  - In the previous example, we'll get 256 8-input NOR gates:

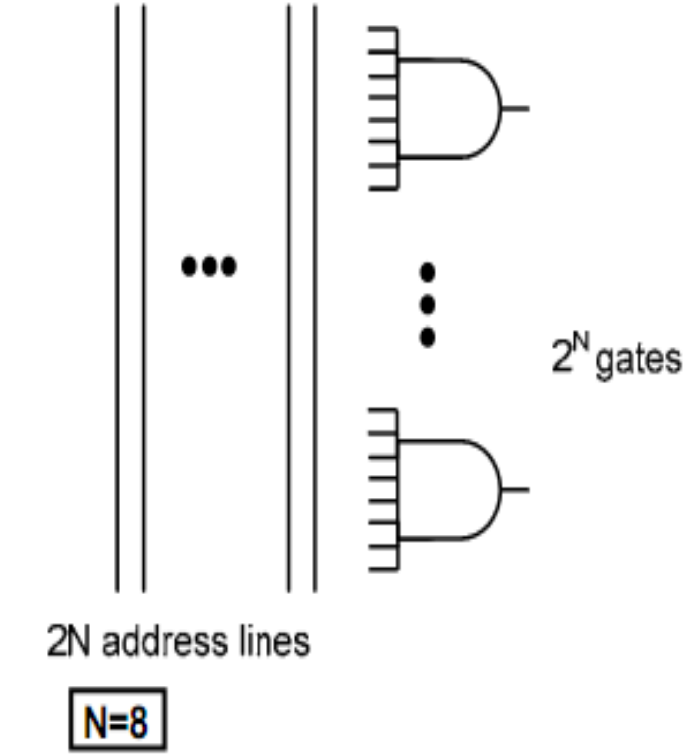
$$WL_0 = \overline{A_7 + A_6 + A_5 + A_4 + A_3 + A_2 + A_1 + A_0}$$

$$WL_{255} = \overline{\overline{A}_7 + \overline{A}_6 + \overline{A}_5 + \overline{A}_4 + \overline{A}_3 + \overline{A}_2 + \overline{A}_1 + \overline{A}_0}$$

How should we build it?

- **Let's build a row decoder for a 256x256 SRAM Array.**
  - We need 256 8-input AND Gates.
  - Each gate drives 256 bitcells
- **We have various options:**





- **Which one is best?**

Reminder: Logical Effort

$$t_{pd,i} = t_{pINV}\left(p_i\gamma + EF_i\right)$$

$$EF_i \triangleq LE_i \cdot f_i = LE_i \cdot \frac{b_i \cdot C_{in,i+1}}{C_{in,i}}$$

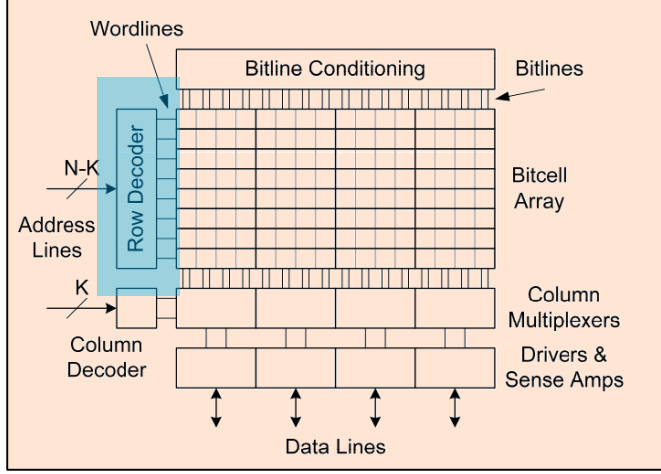
$$PE = F \cdot \prod LE \cdot B = {C_L}/{C_{in,1}} \cdot \prod LE_i \prod b_i$$

$$EF_{opt} = \sqrt[N]{PE} = \sqrt[N]{F \cdot \prod LE_i \prod b_i}$$

$$N_{opt} = \log_{EF_{opt}} PE = \log_{EF_{opt}} F \cdot LE \cdot B$$

$$t_{pd} = t_{pINV}\sum\left(p_i\gamma + EF_i\right) = t_{pINV}\left(\gamma\sum p_i + N \cdot \sqrt[N]{PE}\right)$$

Problem Setup



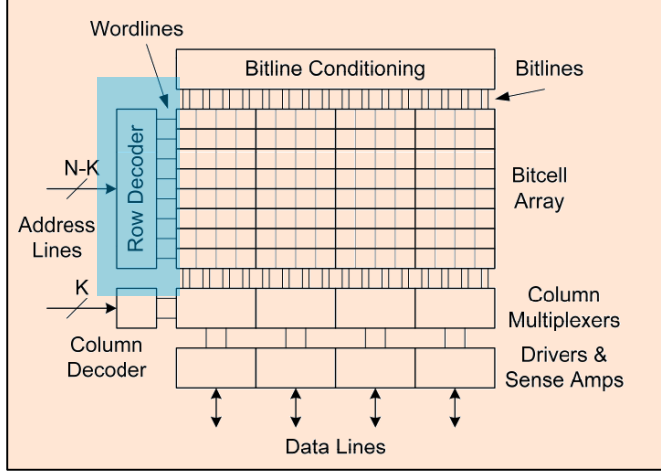
- **For LE calculation we need to start with:**
  - Output Load (C_L)
  - Input Capacitance (C_{in})
  - Branching (B)
- **What is the Load Capacitance?**
  - 256 bitcells on each Word Line

$$C_{WL} = 256 \cdot C_{Cell} + C_{Wire}$$

  - *Let's ignore the wire for now…*
- **What is the Input Capacitance?**
  - Let's assume our address drivers can drive a bit more than a bitcell, so:

$$C_{in,addr\_driver} = 4 \cdot C_{Cell}$$

Problem Setup



- **What is the Branching Effort?**
  - Lets take another look at the Boolean expressions:

$$WL_0 = \bar{A}_7 \bar{A}_6 \bar{A}_5 \bar{A}_4 \bar{A}_3 \bar{A}_2 \bar{A}_1 \bar{A}_0$$

$$WL_{255} = A_7 A_6 A_5 A_4 A_3 A_2 A_1 A_0$$

  - We see that half of the signals use A_i and half use A_i!
  - So each address driver drives 128 8-input AND gates, but only one is on the selected WL path.

$$C_{on\ path} = C_{nand}; \quad C_{off\ path} = 127 \cdot C_{nand}$$

$$B_{add\_driver} = \frac{C_{on\ path} + C_{off\ path}}{C_{on\ path}} = \frac{C_{nand} + 127 \cdot C_{nand}}{C_{nand}} = 128$$

Number of Stages

- **Altogether the path effort is:** $$PE = LE \cdot B \cdot F = LE \cdot \Pi b_i \frac{C_{WL}}{C_{address}} = LE \cdot 128 \cdot \frac{256 C_{Cell}}{4 C_{Cell}}$$
$$= LE \cdot 8k = 2^{13} \cdot LE$$

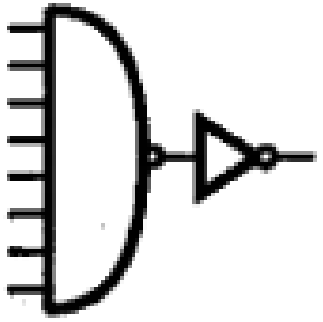
- **The best case logical effort is** $$\Pi LE = 1$$

- **So the minimum number of stages for optimal delay is:** $$PE = 2^{13}$$
$$N_{opt} = \log_{3.6} 2^{13} = 7$$

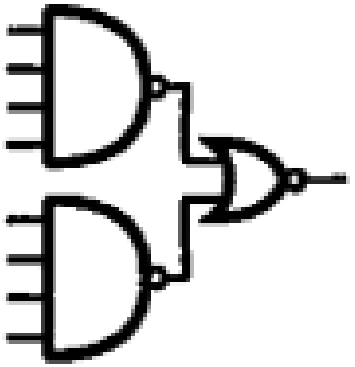
- **That's a lot of stages!**

So which implementation should we use?

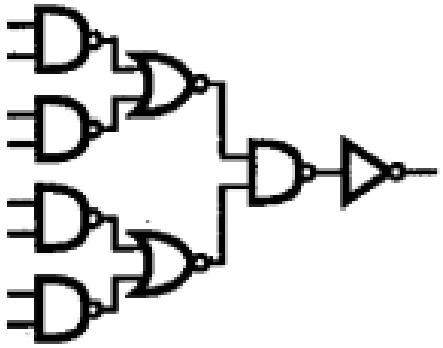
- **The one with the minimum Logical Effort:**



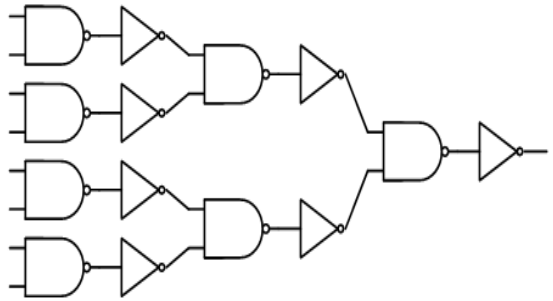
$$\Pi LE = (10/3)\cdot 1 = 10/3;$$
$$p = 8+1 = 9$$



$$\Pi LE = 2\cdot(5/3) = 10/3$$
$$p = 4+2 = 6$$



$$\Pi LE = (4/3)\cdot(5/3)\cdot(4/3)\cdot 1 = 80/27;$$
$$p = 2+2+2+1 = 7$$



$$\Pi LE = (4/3)^3 = 2.37;$$
$$p = 2\cdot 3+1\cdot 3 = 9$$

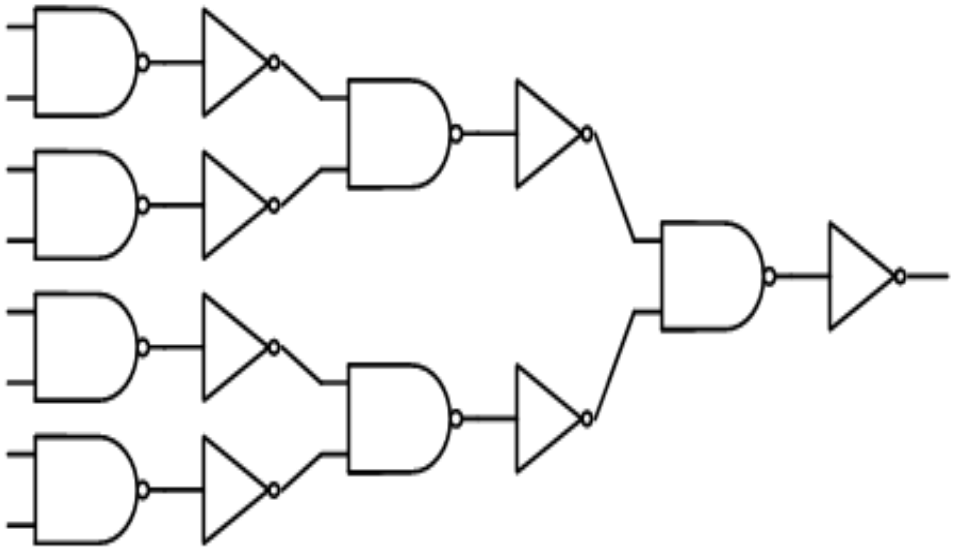
New optimal number of Stages

- **So now we can calculate the actual path effort:**

$$PE = F \cdot \Pi b_i \cdot \Pi LE_i =$$

$$= 2.37 \cdot 2^{13} = 19.418k$$

$$N_{opt} = \log_{3.6} PE = 7.7$$



- **We could add another inverter or two to get closer to the optimal number of stages…**

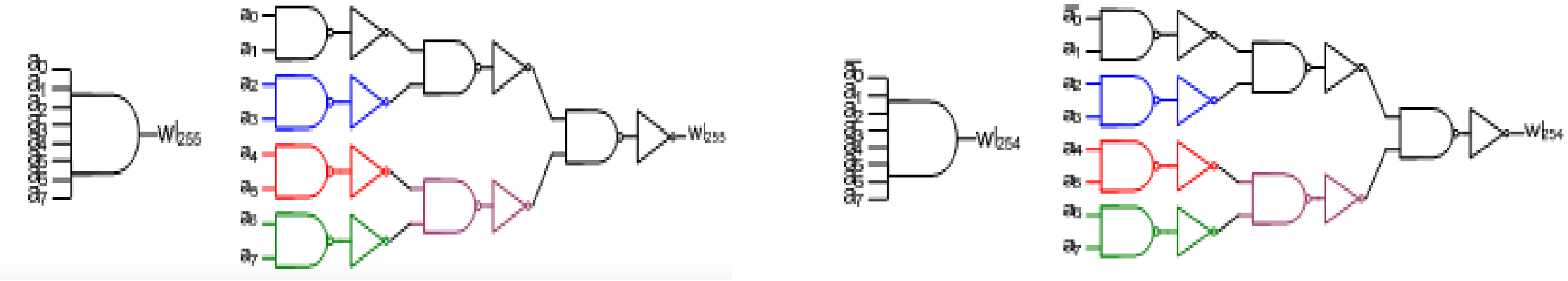
Implementation Problems

- **Address Line Capacitance:**
  - Our assumption was that $C_{\text{in,addr\_driver}} = 4C_{\text{cell}}$.
  - But each address drives 128 gates
    - That's a really long wire with high capacitance.
  - This means that we will need to buffer the address lines
    - This will probably ruin our whole analysis...

- **Bit-cell Pitch:**
  - Each signal drives one row of bitcells.
  - How will we fit 8 address signals into this pitch?

Predecoding - Concept

- **Solution:**
  - Let's look at two decoder paths: WL_{254}, WL_{255}



  - We see that there are many "shared" gates.
    - So why not share them?
  - For instance, we can use the purple output for both gates…

Predecoding - Method

- **How do we do this?**
  - If we look at the final Boolean expression, it has combinations of groups of inputs.
  - By grouping together a few inputs, we actually create a small decoder.
  - Then we just AND the outputs of all the "pre" decoders.
- **For example: Two 4:16 predecoders**

$$D = dec(A_0, A_1, A_2, A_3);\ E = dec(A_4, A_5, A_6, A_7);$$

$$WL_0 = D_0 \cdot E_0;\ WL_{255} = D_{15} \cdot E_{15};\ WL_{254} = D_{15} \cdot E_{14};$$

Predecoding - Example

- **Let's look at our example:**

$$D = dec(A_0, A_1, A_2, A_3) \qquad WL_0 = D_0 \cdot E_0$$

$$E = dec(A_4, A_5, A_6, A_7) \qquad WL_{255} = D_{15} \cdot E_{15}$$

$$WL_{254} = D_{15} \cdot E_{14}$$

- **What is our new branching effort?**
  - As before, each address drives half the lines of the small decoder.
  - Each predecoder output drives 256/16 post-decoder gates.
  - Altogether, the branching effort is:

$$B = b_{addr\_driver} \cdot b_{predecoder} = {}^{16}\!/_{2} \cdot {}^{256}\!/_{16} = 128$$

- **Same as before!**

Predecoding - Solution

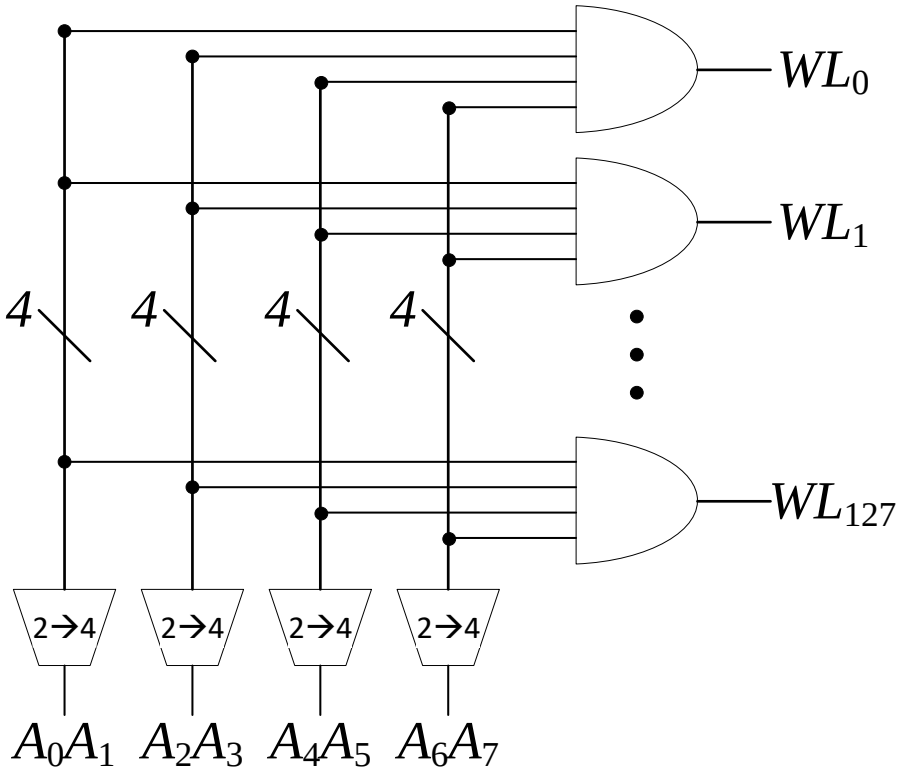
- **Why is this a better solution?**
  - Each Address driver is only driving four gates
    - less capacitance.
  - We saved a ton of area by “sharing” gates.
  - We can “Pitch Fit” 2-input NAND gates.

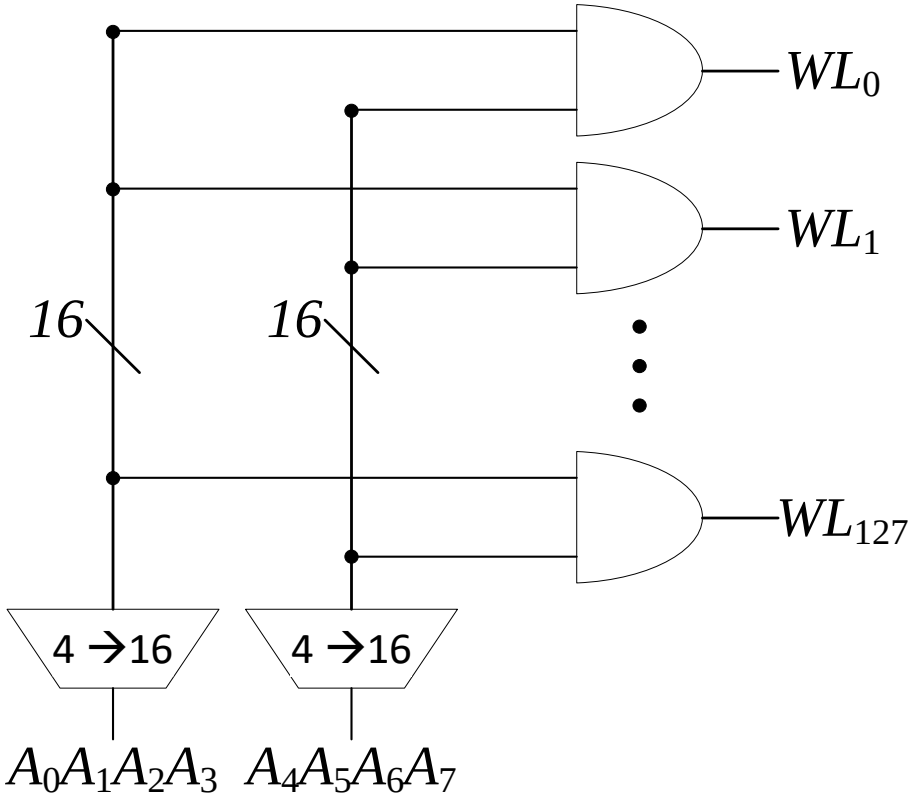
Another Predecoding Example

- **We can try using four 2-input predecoders:**
  - This will require us to use 256 4-input NAND gates.

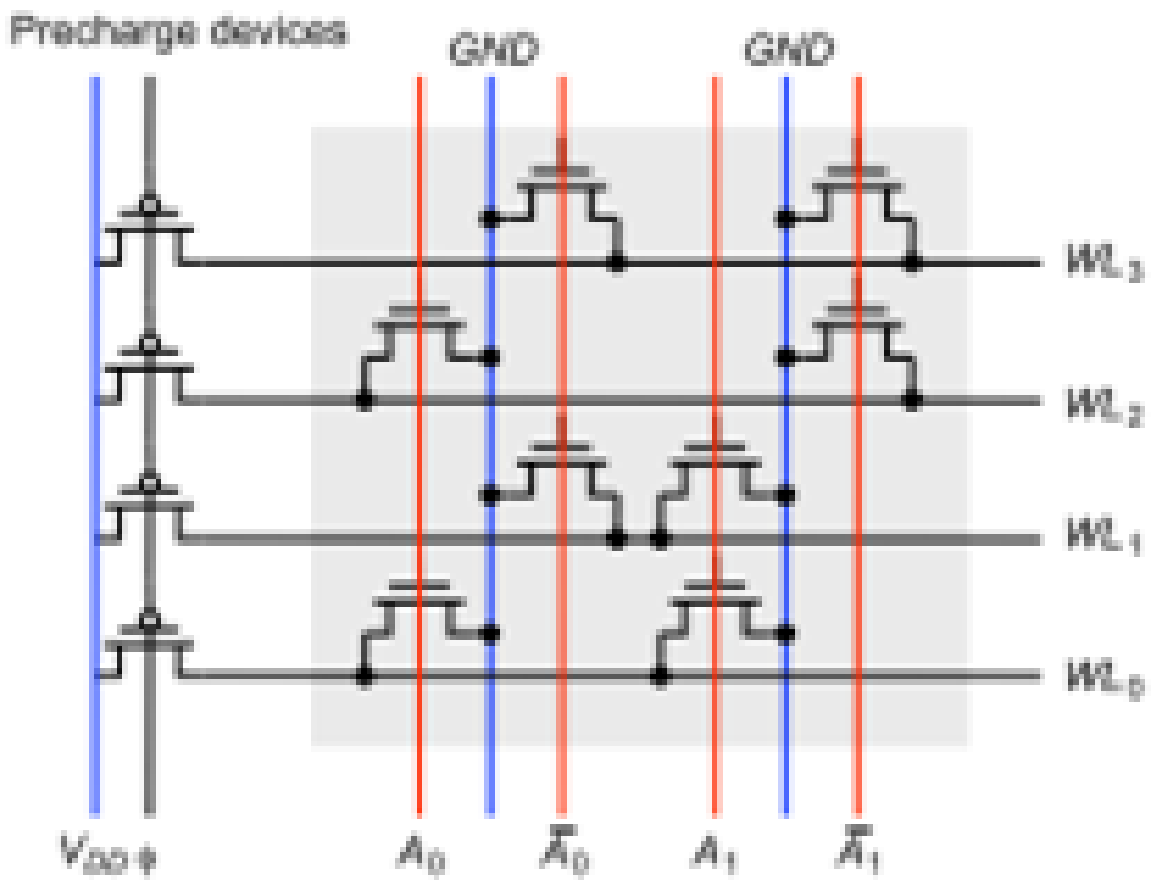
How do we choose a configuration?

- **Pitch Fitting**: **2-input NANDs vs. 4-input NAND.**
- **Switching Capacitance**: **How many wires switch at each transition?**
- **Stages Before the large cap**: **Distribution of the load along the delay.**
- **Conclusion**: **Usually do as much predecoding as possible!**

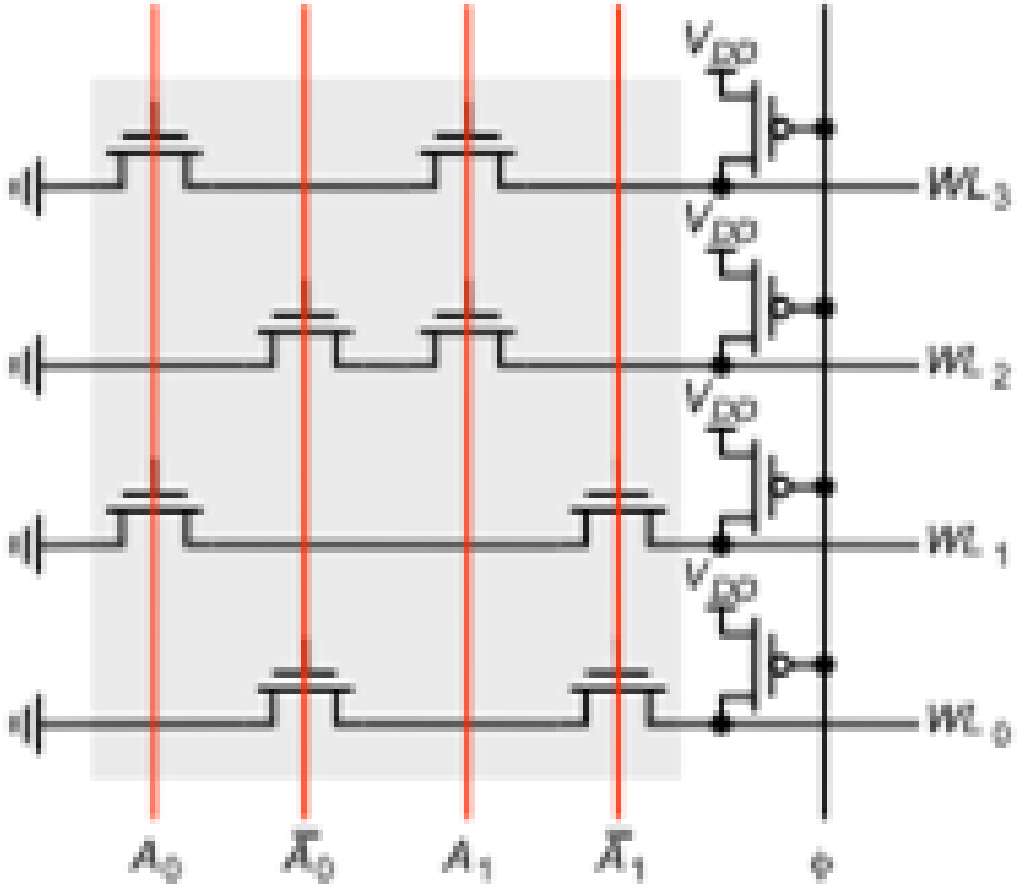




Alternative Solution: Dynamic Decoders



2-input NOR decoder

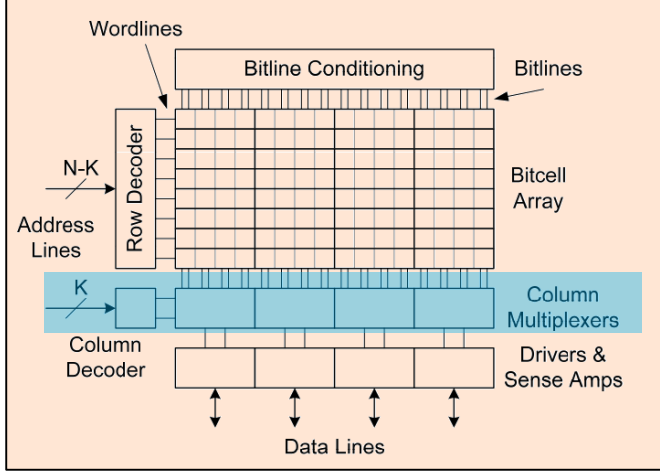


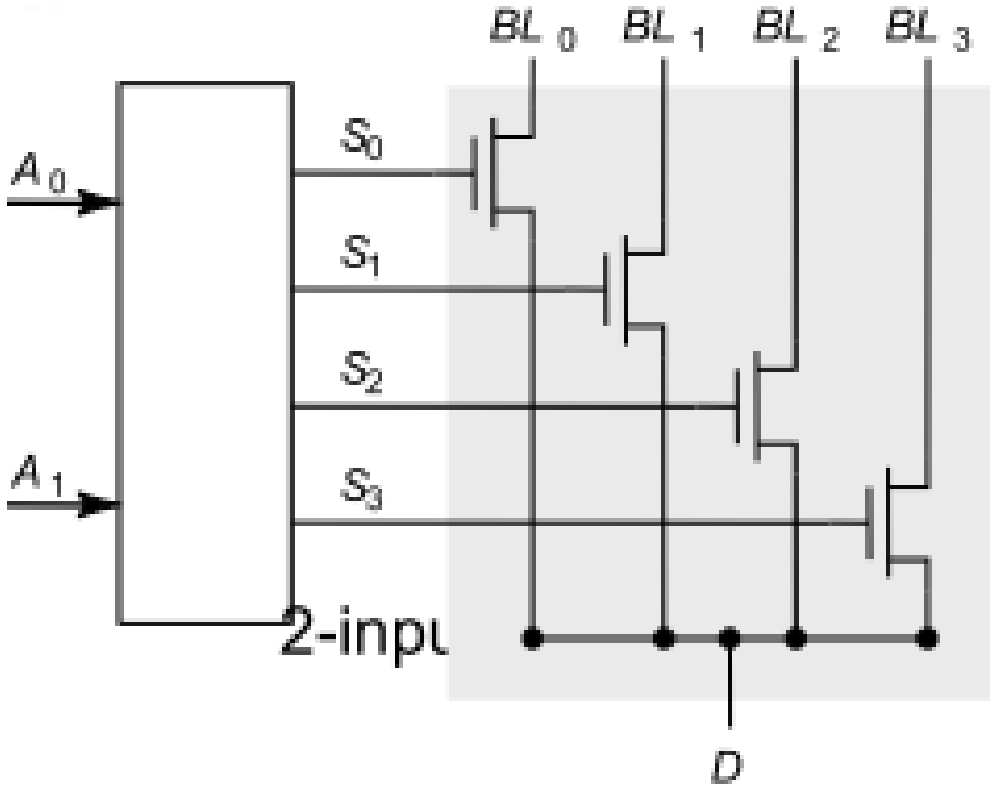
2-input NAND decoder

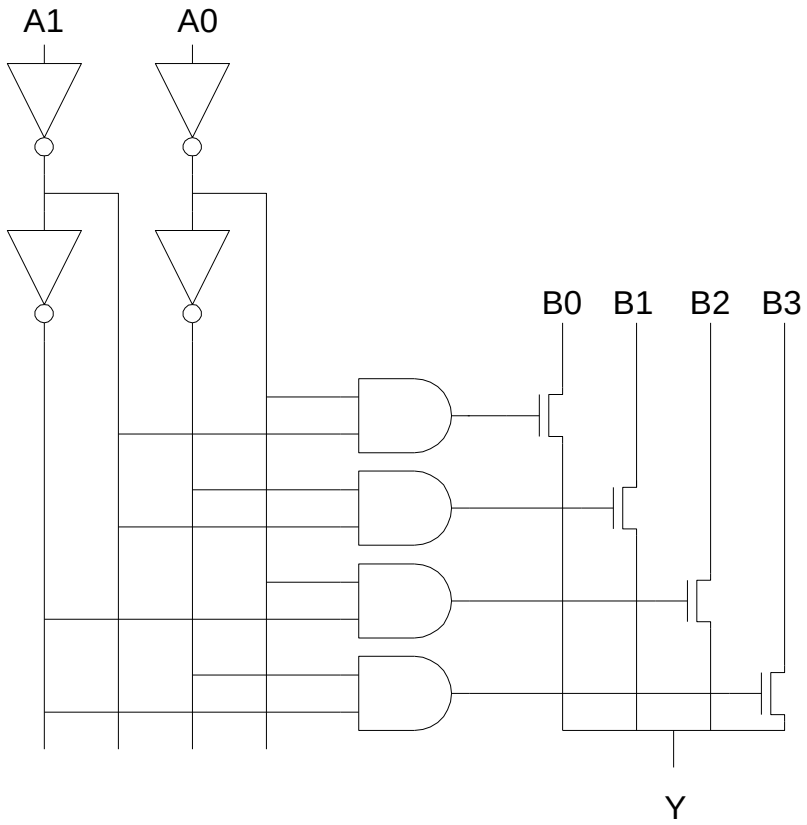
Column Multiplexer

Column Multiplexer

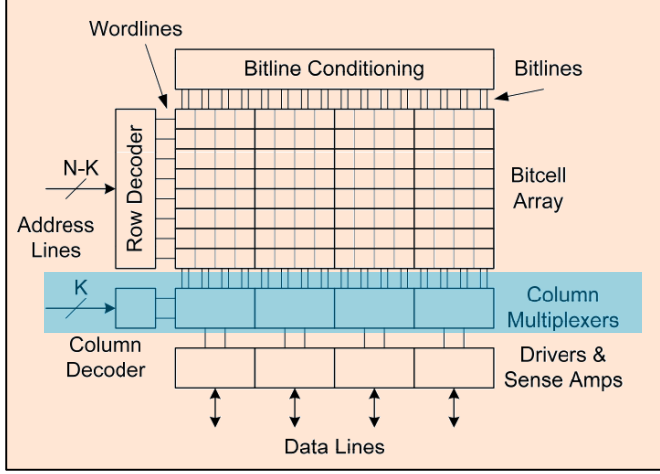
- **First option – PTL Mux with decoder**
  - Fast – only 1 transistor in signal path.
  - Large transistor Count



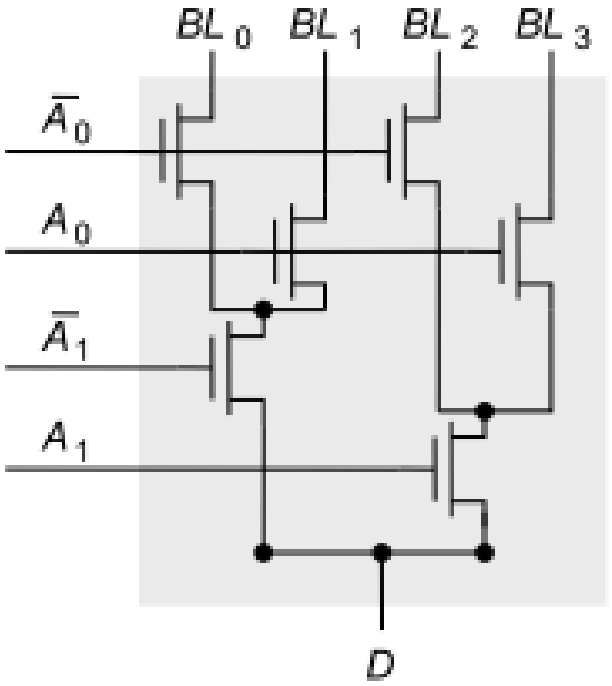


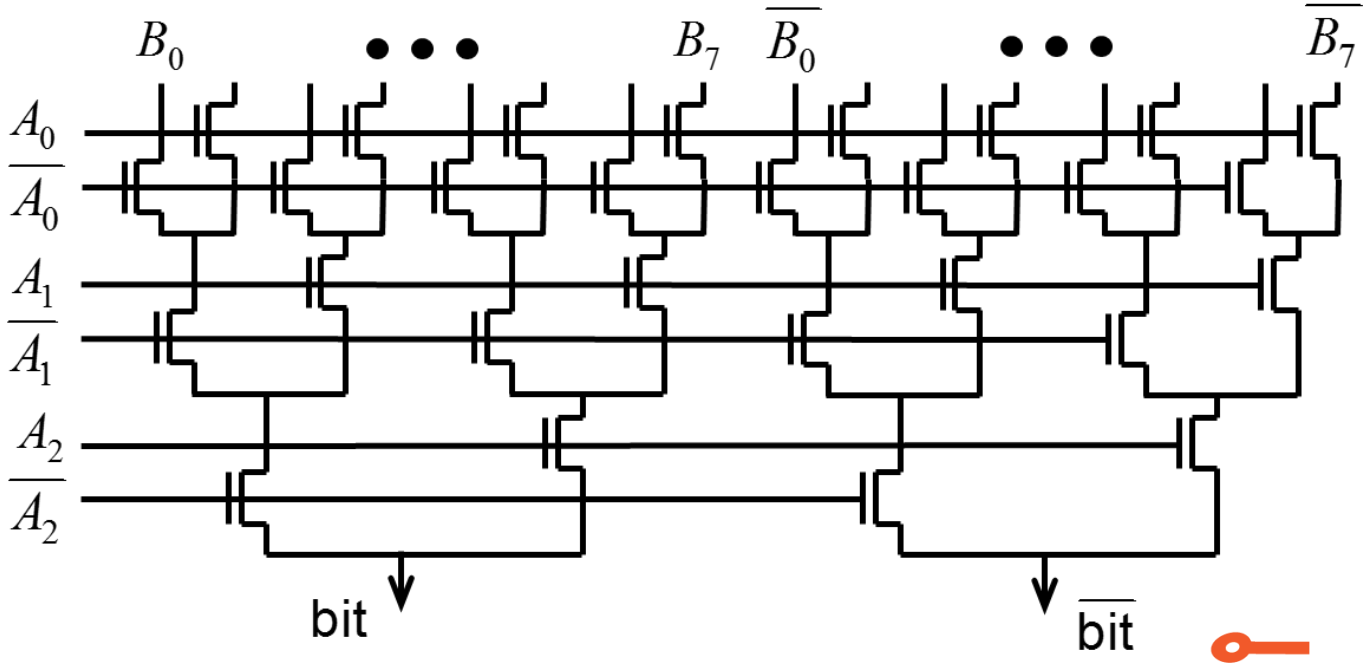


4 to 1 tree decoder



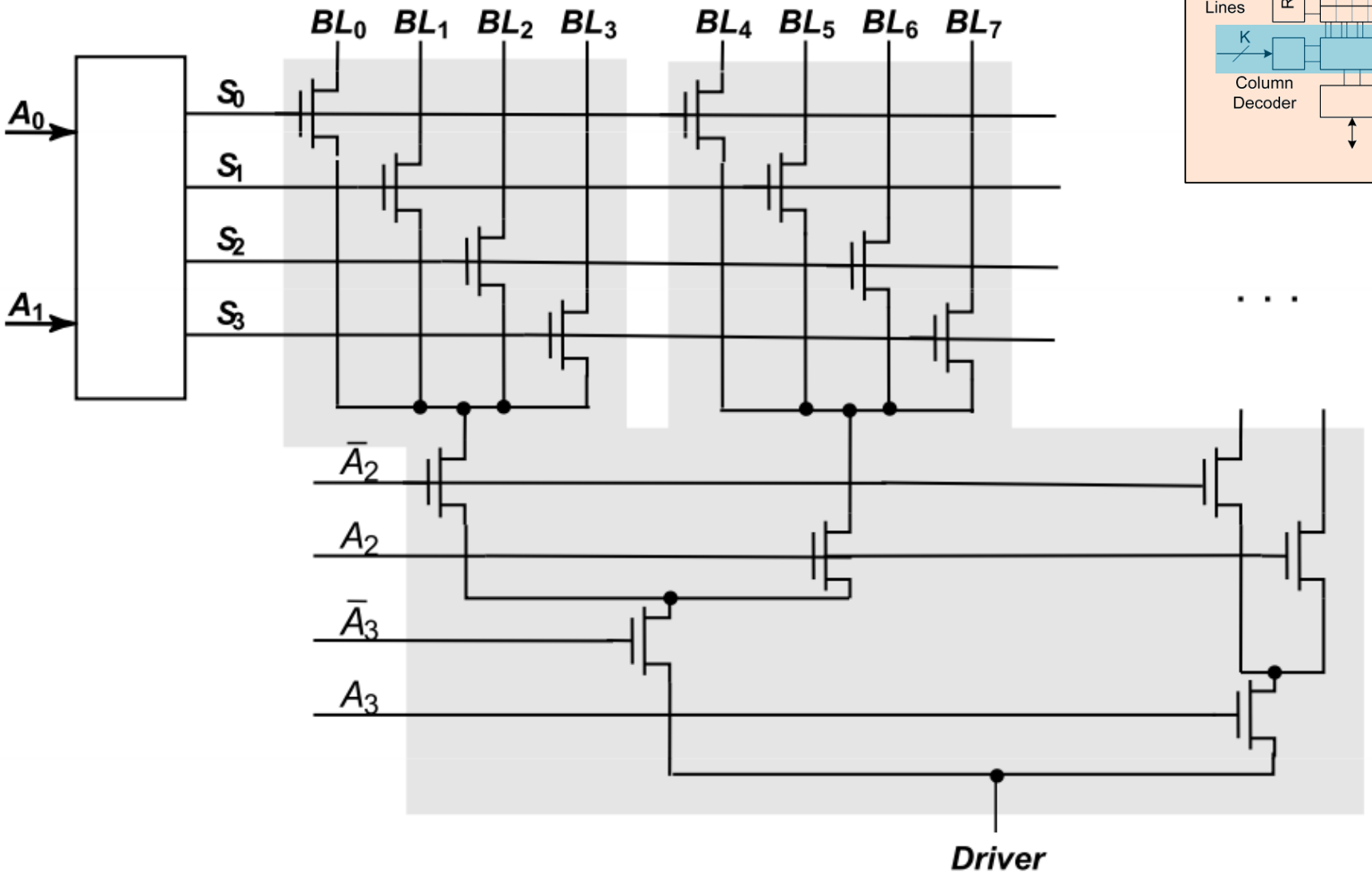
- **Second option – Tree Decoder**
  - For $2k$:1 Mux, it uses k series transistors.
  - Delay increases quadratically
  - No external decode logic → big area reduction.

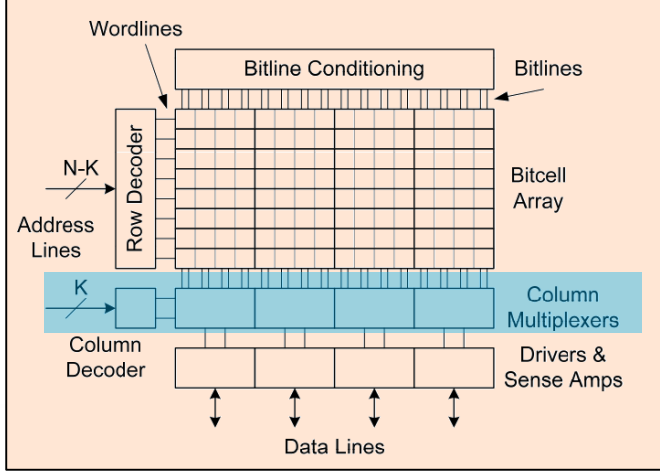




To sense Amps and Write Circuits

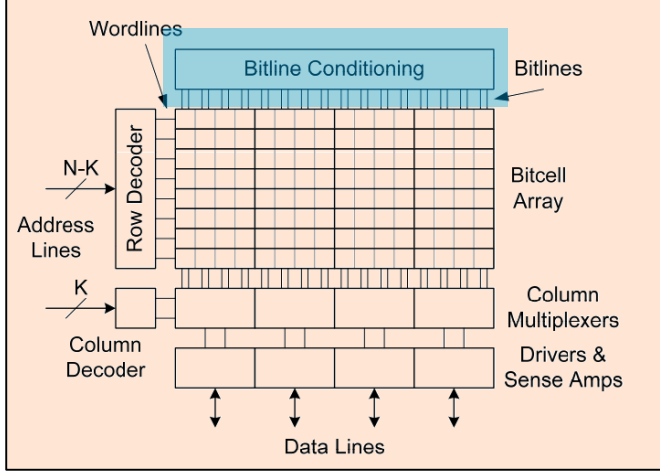
Combining the Two



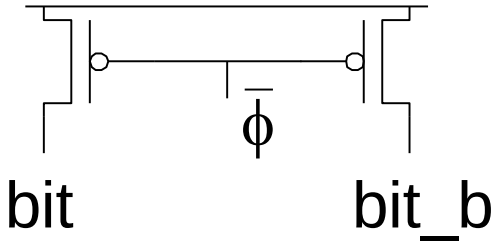


Precharge and Sense Amp

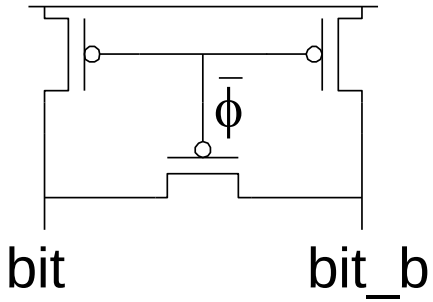
Precharge Circuitry



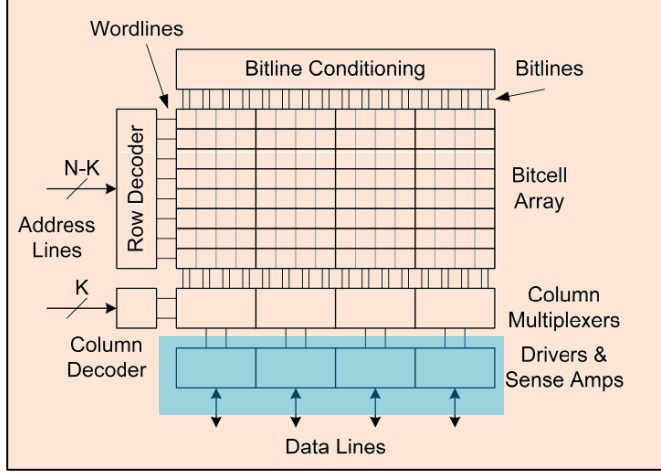
- **Precharge bitlines high before reads**



- **Equalize bitlines to minimize voltage difference when using sense amplifiers**



Sense Amplifiers



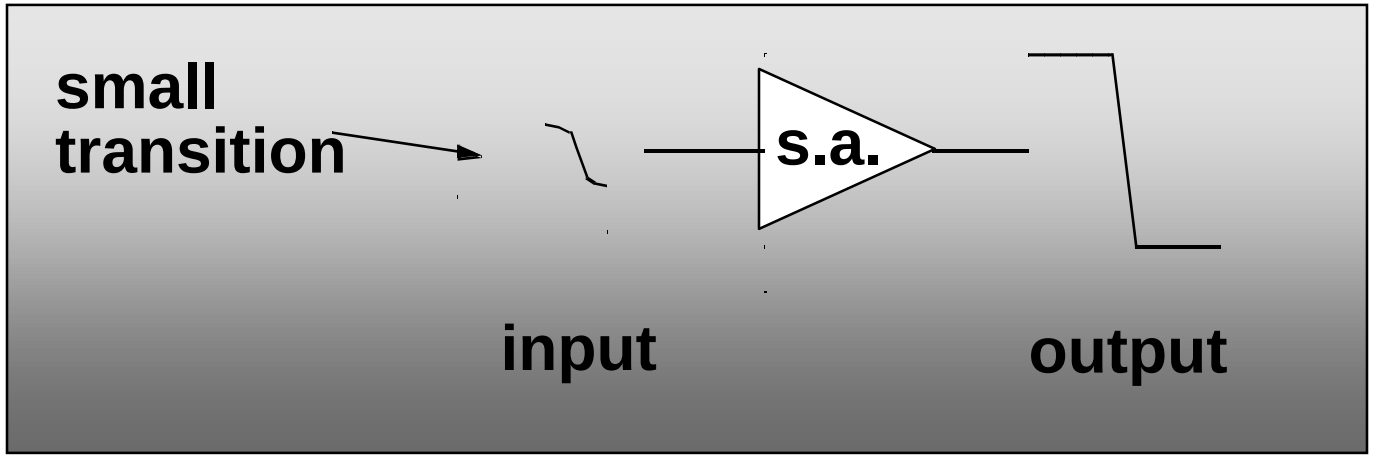
$$t_p = \frac{C \cdot \Delta V}{I_{av}}$$

make ΔV as small as possible

large

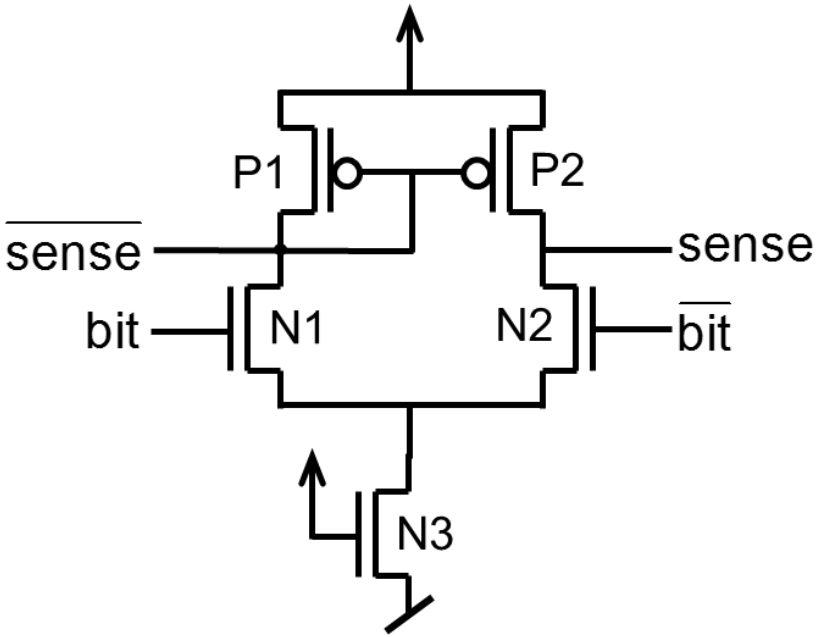
small

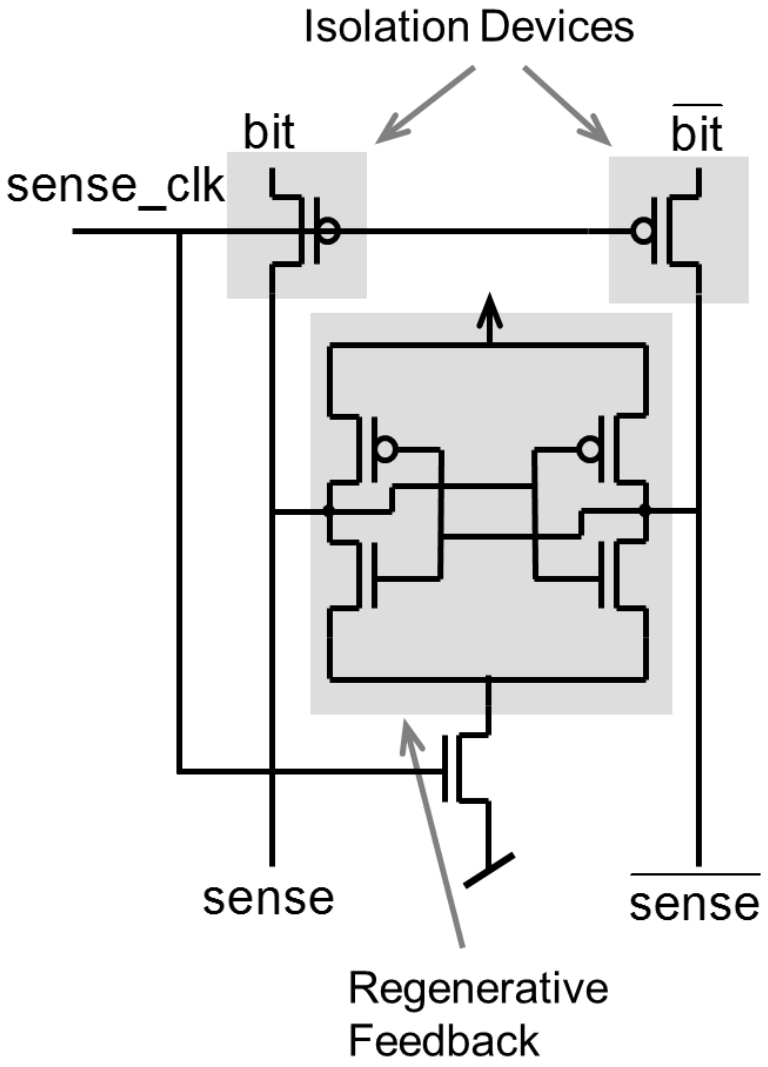
Idea: Use Sense Amplifer



Differential Sense Amplifier

- **Non-clocked Sense Amp has high static power.**
- **Clocked sense amp saves power**
- **Requires *sense_clk* after enough bitline swing**
- **Isolation transistors cut off large bitline capacitance**





Further Reading

- Rabaey, et al. “Digital Integrated Circuits” (2nd Edition)
- Elad Alon, Berkeley ee141 (online)
- Weste, Harris, “CMOS VLSI Design (4th Edition)”