EECS151 : Introduction to Digital Design and ICs

Lecture 23 – SRAM

Bora Nikolić

Intel's Process Roadmap to 2025: with 4nm, 3nm, 20A and 18A!?
Ian Cutress, Anandtech, July 2021

2020, Intel 10nm SuperFin (10SF): Current generation technology in use with Tiger Lake and Intel's Xe-LP discrete graphics solutions (SG1, DG1). The name stays the same.

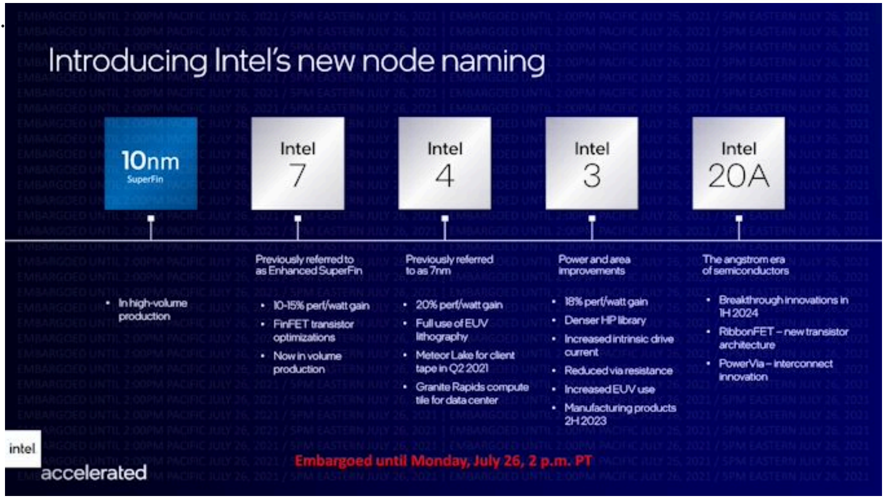
2021 H2, Intel 7: Previously known as 10nm Enhanced Super Fin or 10ESF. Alder Lake and Sapphire Rapids will now be known as Intel 7nm products, showcasing a 10-15% performance per watt gain over 10SF due to transistor optimizations. Alder Lake is currently in volume production.

2022 H2, Intel 4: Previously known as Intel 7nm. Intel earlier this year stated that its Meteor Lake processor will use a compute tile based on this process node technology, and the silicon is now back in the lab being tested. Intel expects a 20% performance per watt gain over the previous generation, and the technology uses more EUV, mostly in the BEOL. Intel's next Xeon Scalable product, Granite Rapids, will also use a compute tile based on Intel 4.

2023 H2, Intel 3: Previously known as Intel 7+. Increased use of EUV and new high density libraries. This is where Intel's strategy becomes more modular – Intel 3 will share some features of Intel 4, but enough will be new enough to describe this a new full node, in particular new high performance libraries

2024, Intel 20A: Previously known as Intel 5nm. Moving to double digit naming, with the A standing for Ångström, or 10A is equal to 1nm. Few details, but this is where Intel will move from FinFETs to its version of Gate-All-Around (GAA) transistors called RibbonFETs.

2025, Intel 18A: Not listed on the diagram above, but Intel is expecting to have an 18A process in 2025. 18A will be using ASML's latest EUV machines, known as High-NA machines, which are capable of more accurate photolithography. Intel has stated to us that it is ASML's lead partner when it comes to High-NA, and is set to receive the first production model of a High-NA machine.



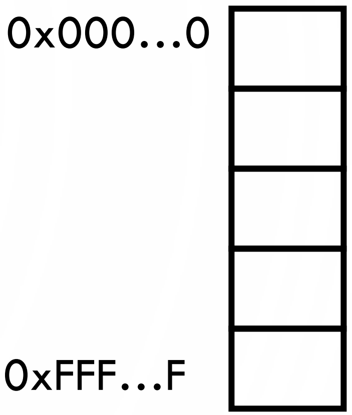
Review

- Latches are based on positive feedback
- Clk-Q delay calculated similarly to combinational logic
- Setup, hold defined as D-Clk times that correspond to Clk-Q delay increases
- Flip-flop is typically a latch pair

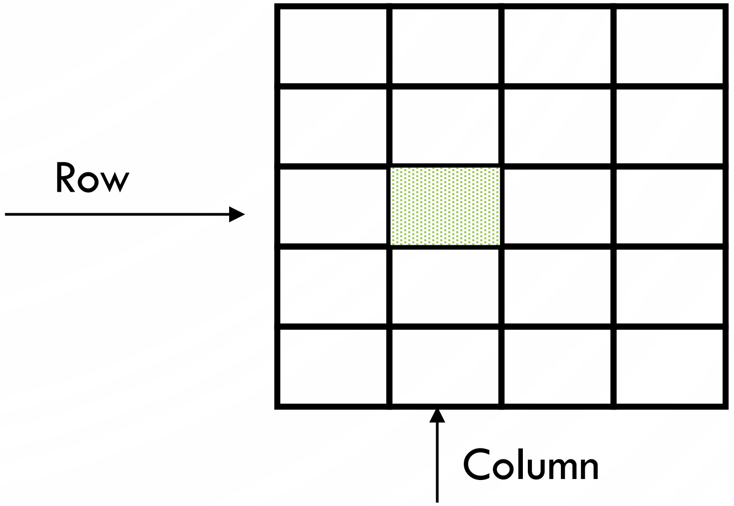
SRAM

Random Access Memory Architecture

- Conceptual: Linear array of addresses
  - Each box holds some data
  - Not practical to physically realize – millions of 32b/64b words

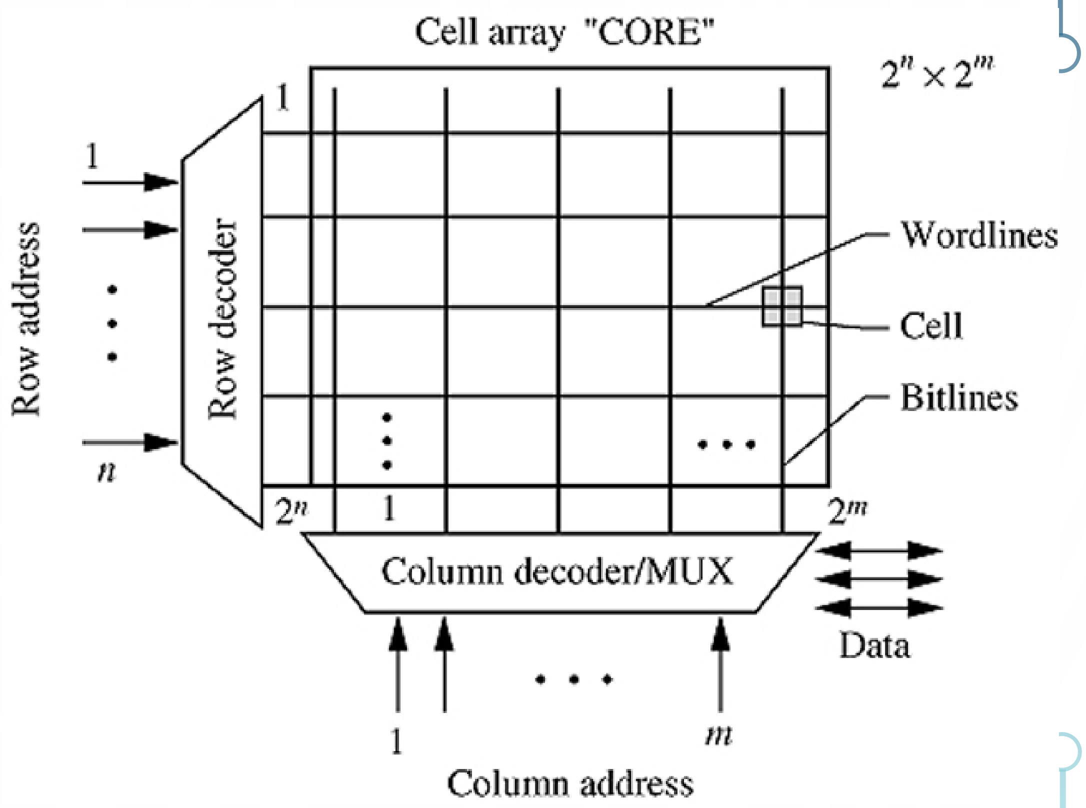


- Create a 2-D array
  - Decode Row and Column address to get data

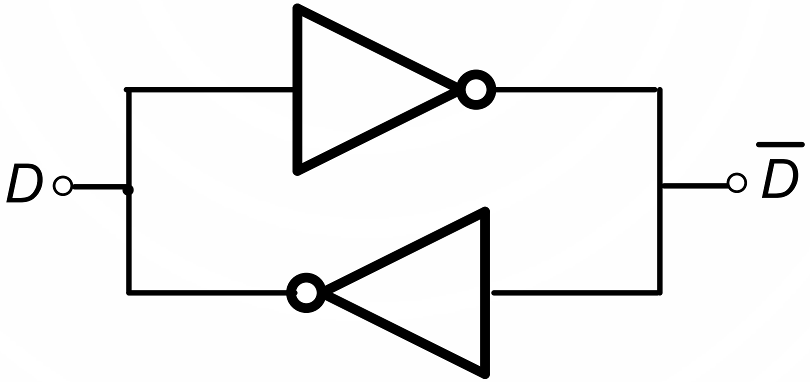


Basic Memory Array

- Core
  - Wordlines to access rows
  - Bitlines to access columns
  - Data multiplexed onto columns
- Decoders
  - Addresses are binary
  - Row/column MUXes are 'one-hot' - only one is active at a time



Basic Static Memory Element

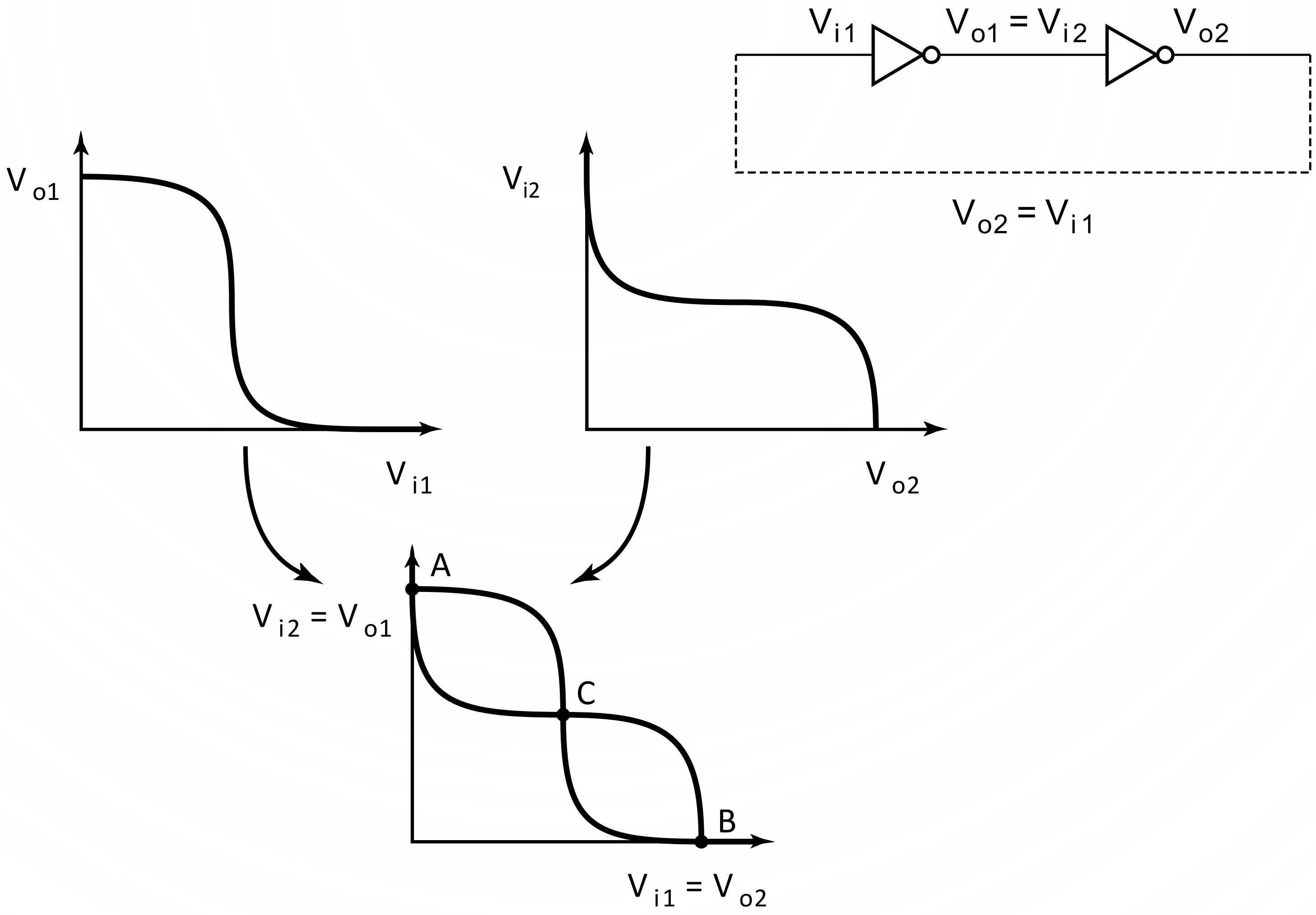


- If D is high, $\overline{D}$ will be driven low
  - Which makes D stay high

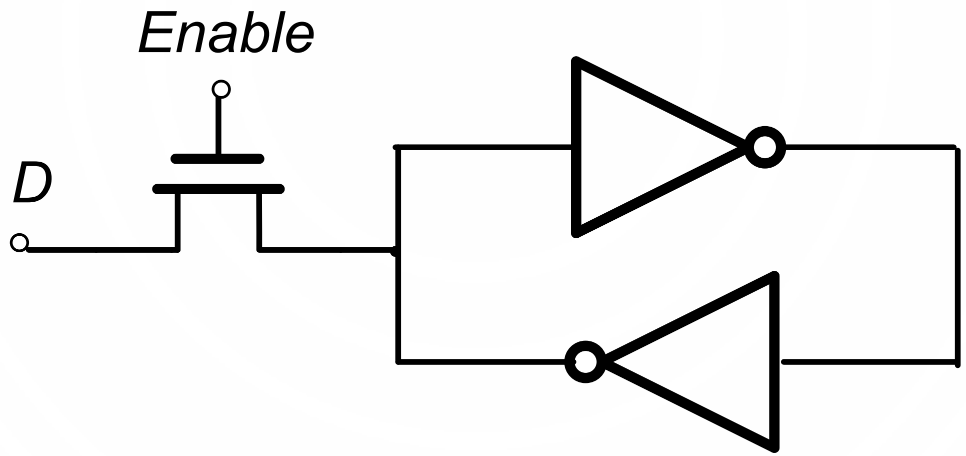
- Positive feedback
- Same principle as in latches

Positive Feedback: Bi-Stability

- As in latches



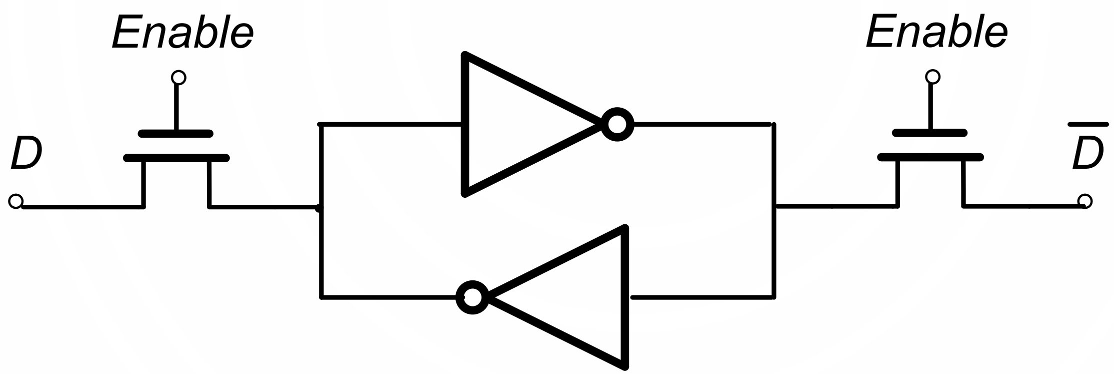
Writing into a Cross-Coupled Pair



- This is a 5T SRAM cell
    - Access transistor must be able to overpower the feedback; therefore must be large
    - Easier to write a 0, harder to write 1
- Can implement as a transmission gate as well; single-ended 6T cell

- There is a better solution…

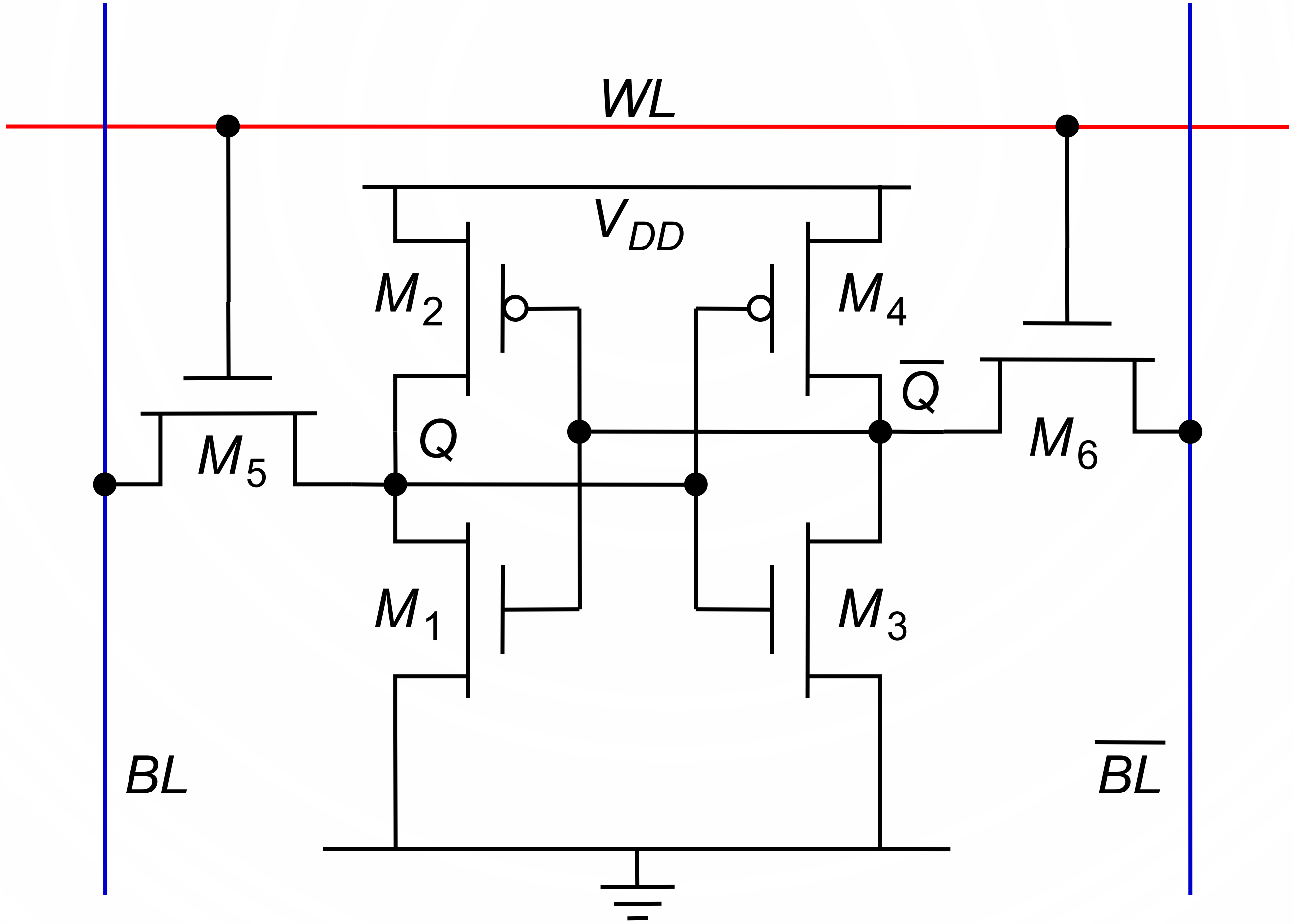
SRAM Cell



Since it is easier to write a 0 through NMOS, write only 0s, but on opposite sides!

When reading, measure the difference

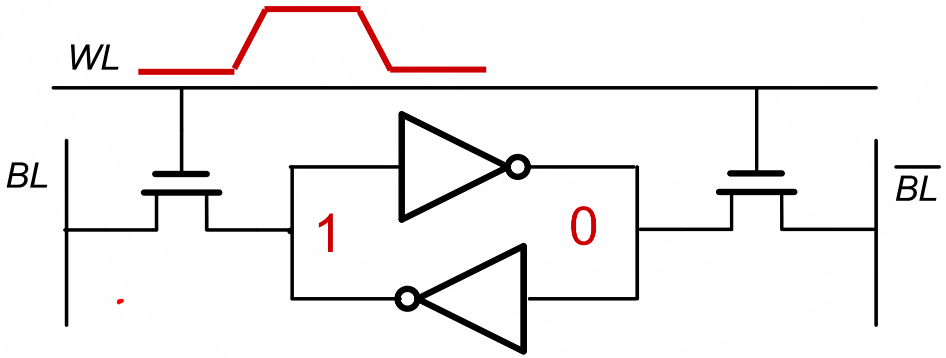
6-transistor CMOS SRAM Cell



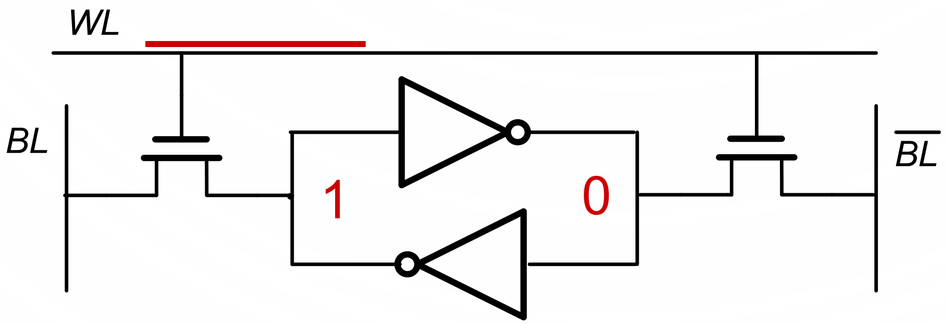
- Wordline (WL) enables read/write access for a row
- Data is written/read differentially through shared BL, $\overline{\text{BL}}$

SRAM Operation

Write

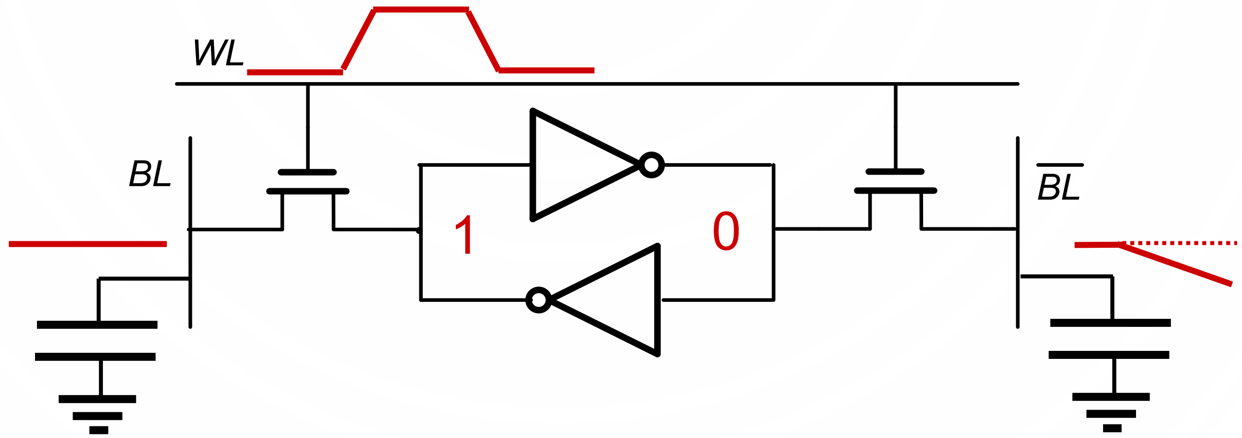


Hold



SRAM Operation

Read

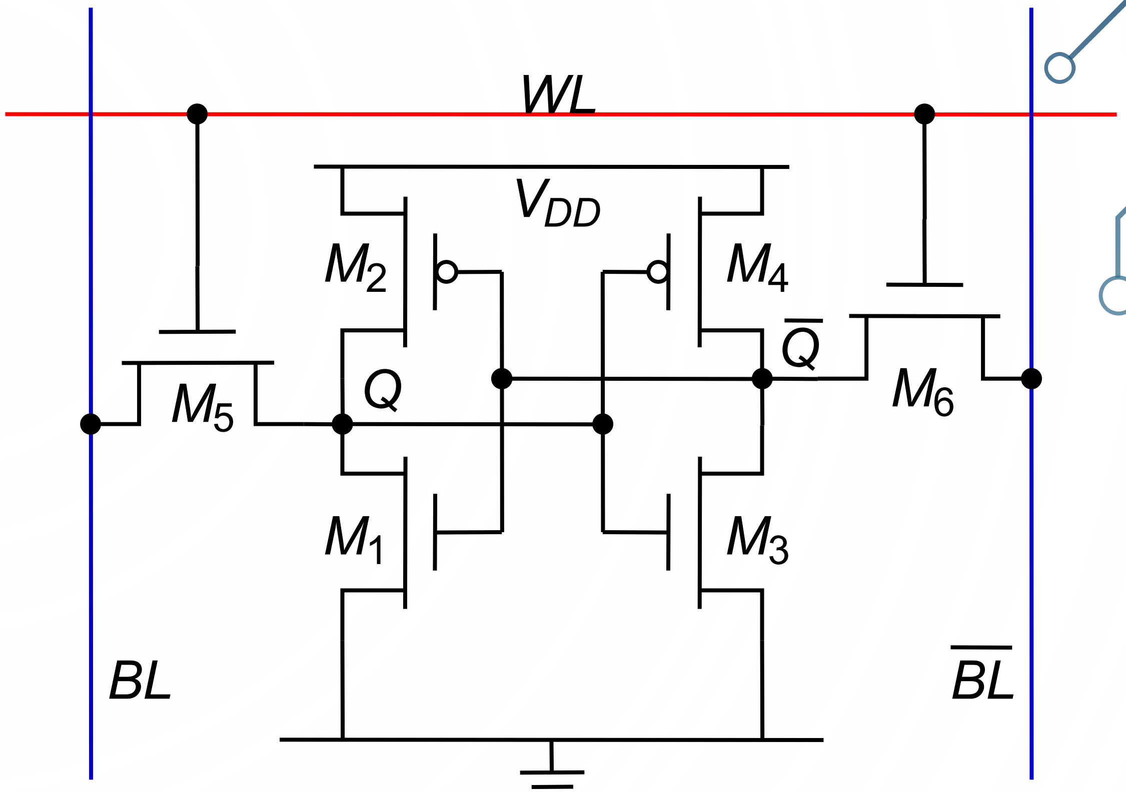


SRAM read in non-destructive

- Reading the cell should not destroy the stored value

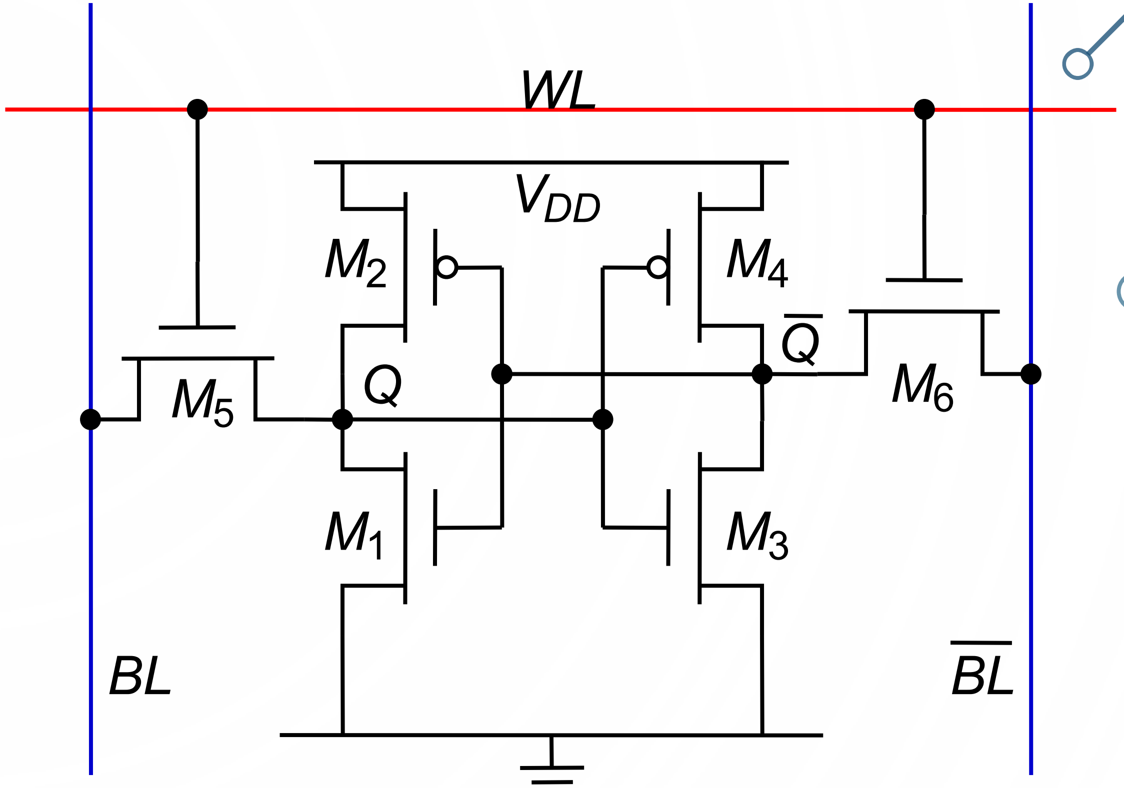
Sizing SRAM Cell

- Read stability: Cell should not change value during read
  - Q = 0: M_5, M_1 both on
  - Voltage divider between M_5, M_1
  - V_Q should stay low, not to flip M_4-M_3 inverter
  - $(W/L)_1 > (W/L)_5$
- Typically $(W/L)_1 = 1.5\ (W/L)_5$
  - In finFETs: $(W/L)_1 = 2(W/L)_5$
- Read speed: Both M_5 and M_1

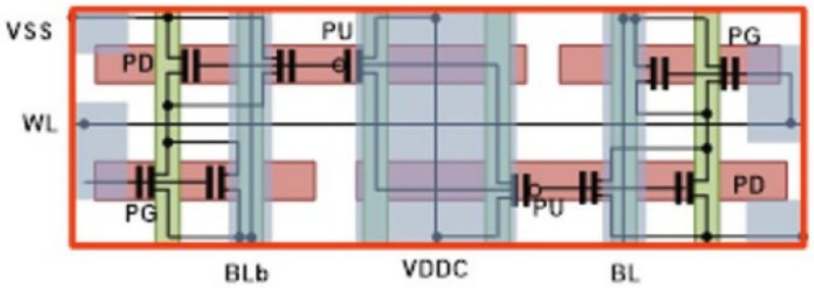


Sizing SRAM Cell

- Writeability: Cell should be writeable by pulling BL low
  - Q = 1, M_5, M_2 both on
  - Voltage divider between M_5, M_2
  - V_Q should pull belowthe switching point of M_4-M_3 inverter
  - $(W/L)_5 > (W/L)_2$
- Typically $(W/L)_5 = (W/L)_2$ in planar
  - In finFETs: $(W/L)_5 = 2(W/L)_2$
  - 1:2:2 and 1:2:3 sizing

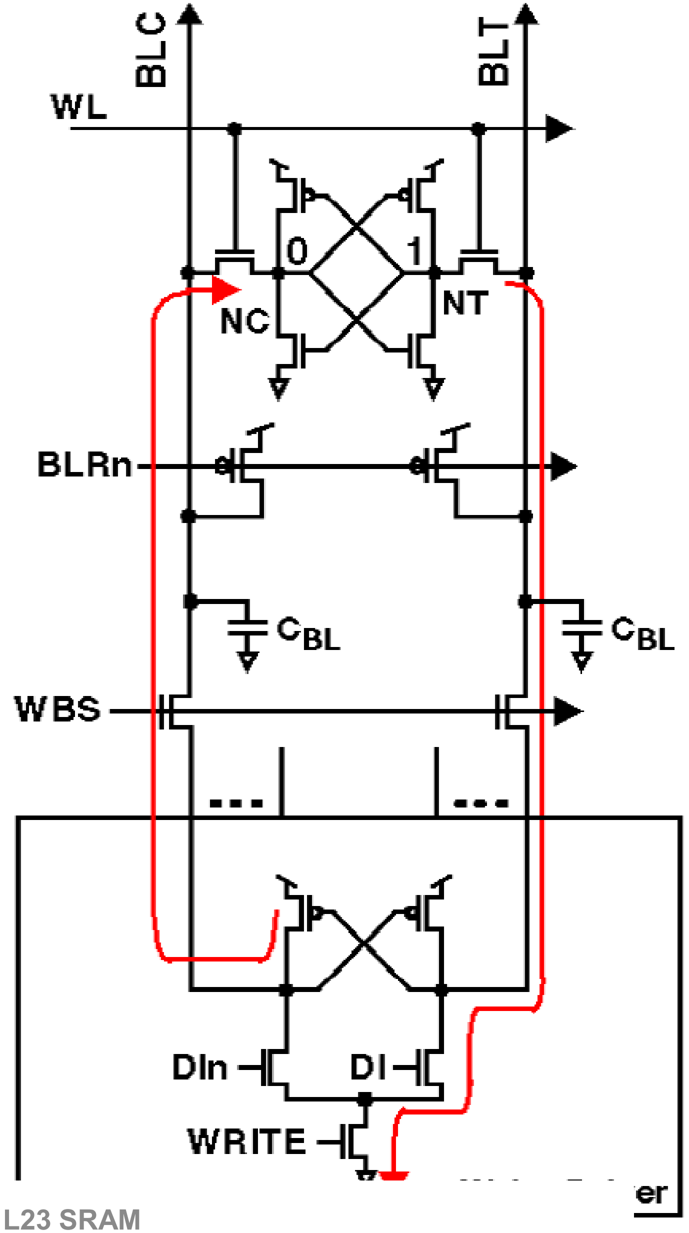


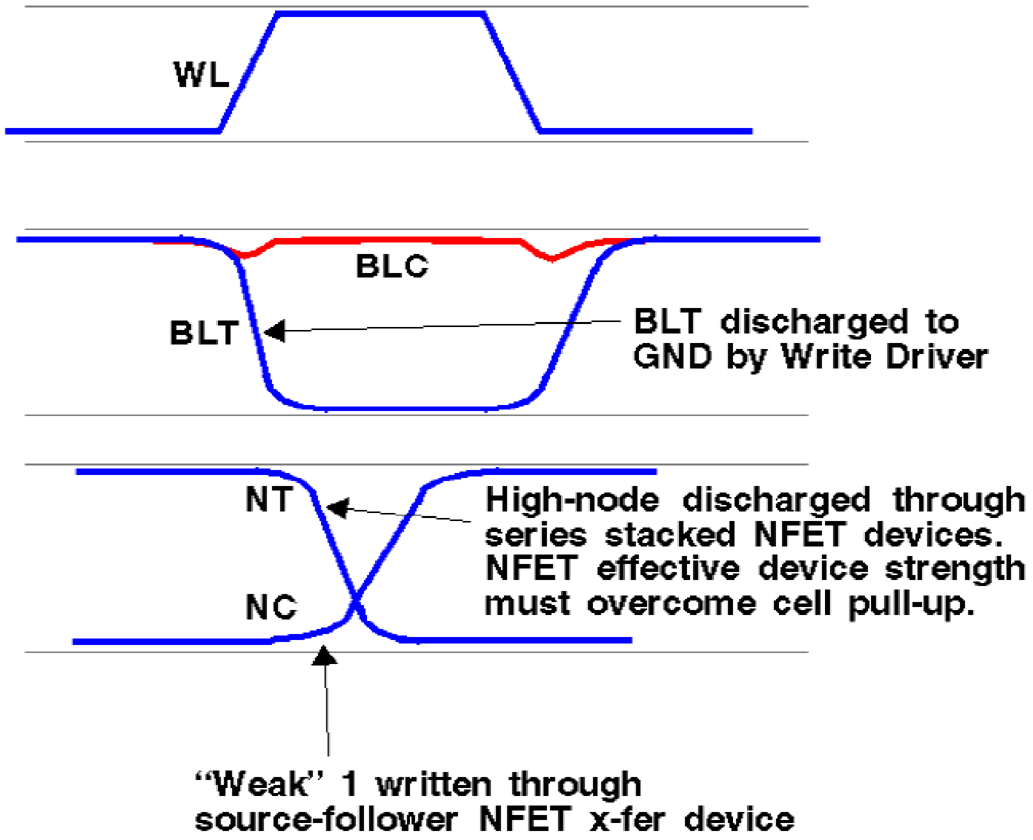
6T High-Current (HC) bitcell
0.049 um[2] (1:2:2)



Song, ISSCC'16

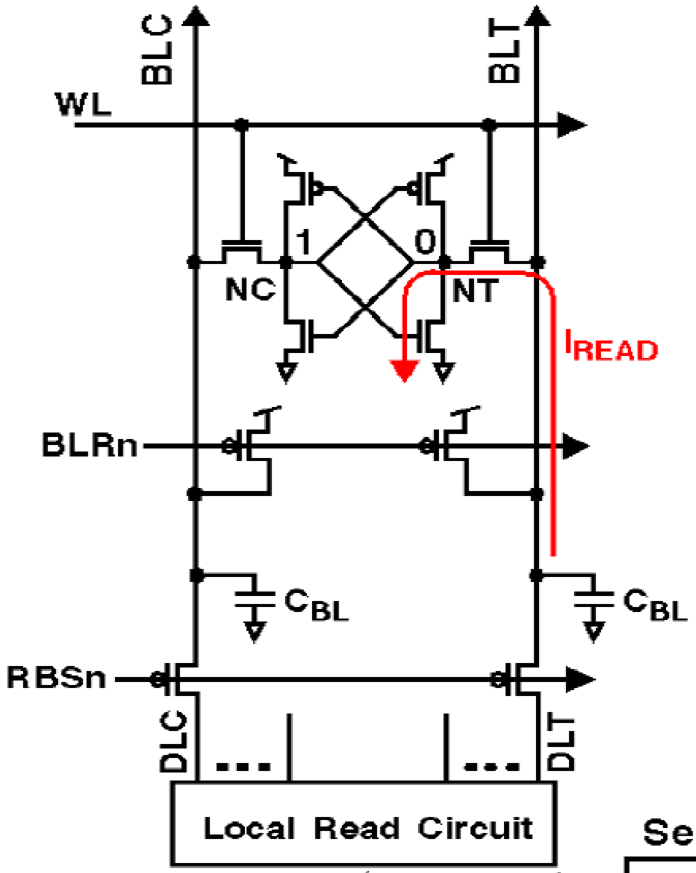
SRAM Column: Write

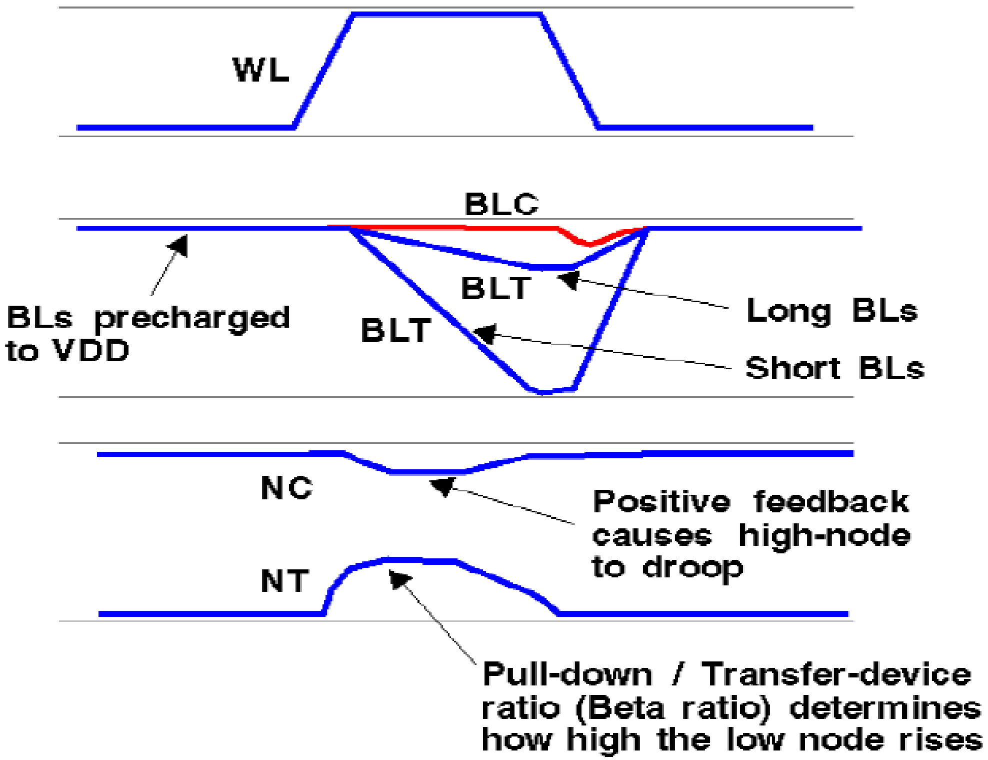




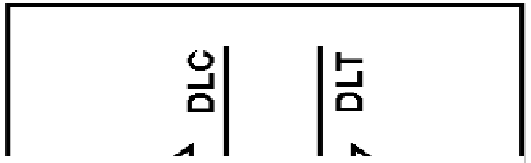
DM 2006

SRAM Column: Read

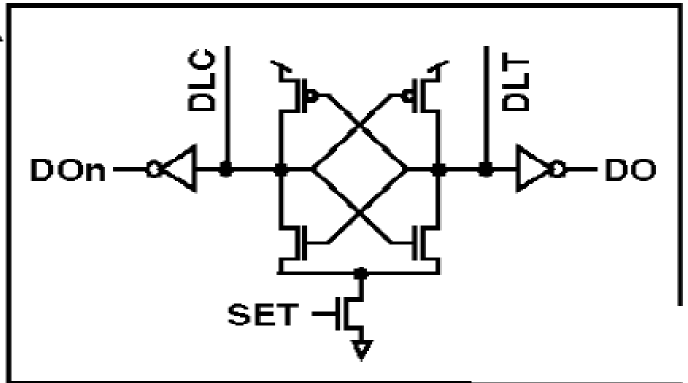




Domino: Short BLs, SOI

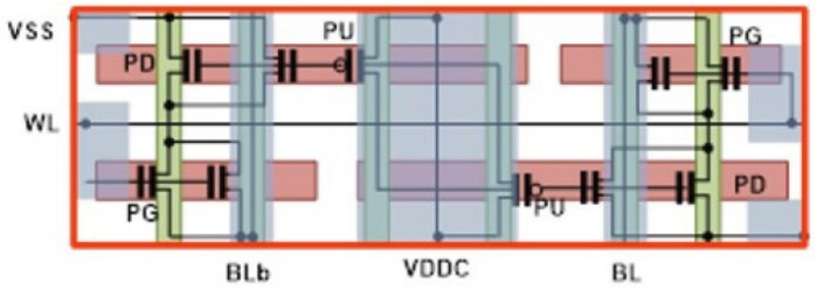


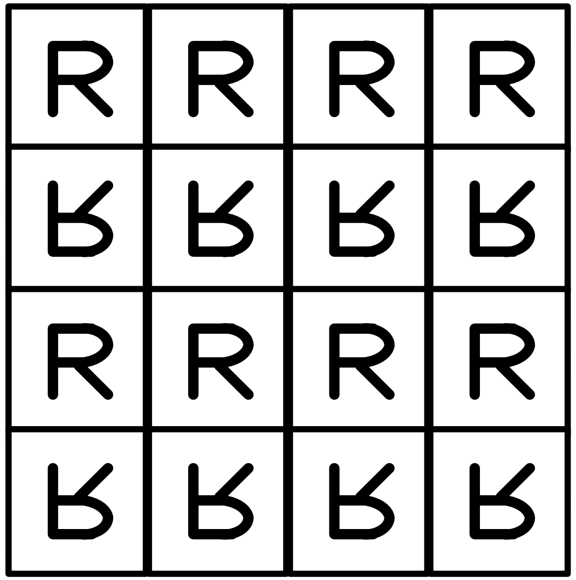
Sense Amplifier: Longer BLs

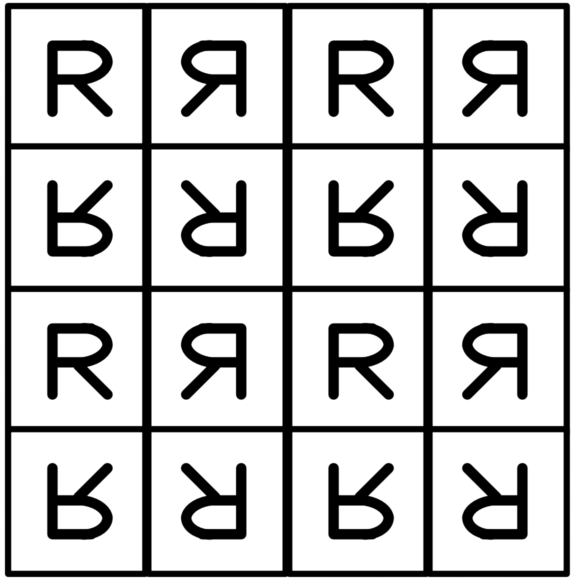


DM 2006

SRAM Array Layout





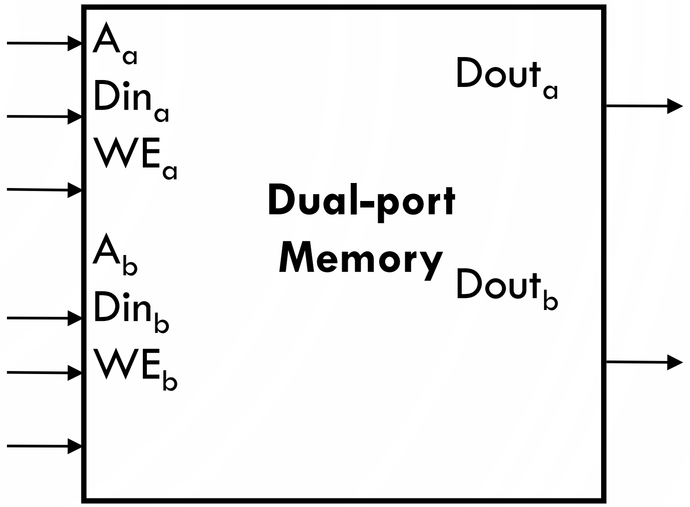


Administrivia

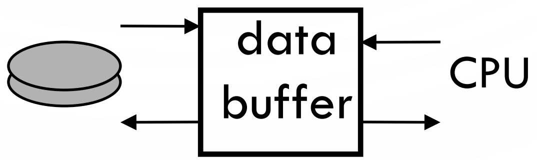
- Homework 10 posted on Friday, due 11/22
  - No homework during Thanksgiving
- Project checkpoints #3 this week

Multi-Ported Memory

- Motivation:
  - Consider CPU core register file:
    - 1 read or write per cycle limits processor performance.
    - Complicates pipelining. Difficult for different instructions to simultaneously read or write regfile.
    - Single-issue pipelined CPUs usually needs 2 read ports and 1 write port (2R/1W).
    - Superscalar processors have more (e.g. 6R/3W)



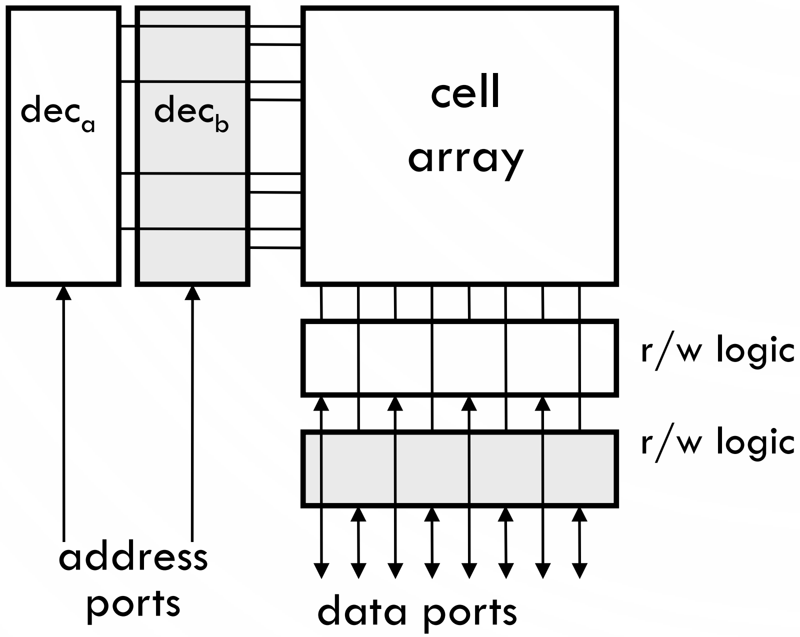
  – I/O data buffering:



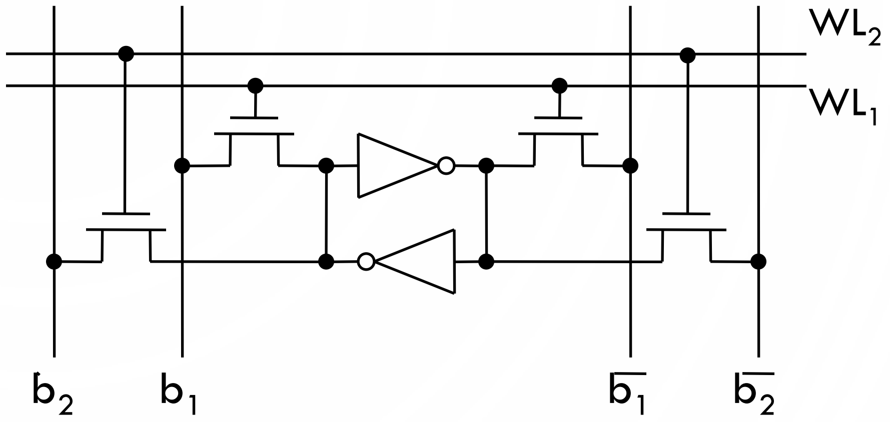
- dual-porting allows both sides to simultaneously access memory at full bandwidth.

Dual-Ported Memory Internals

- Add decoder, another set of read/write logic, bits lines, word lines:



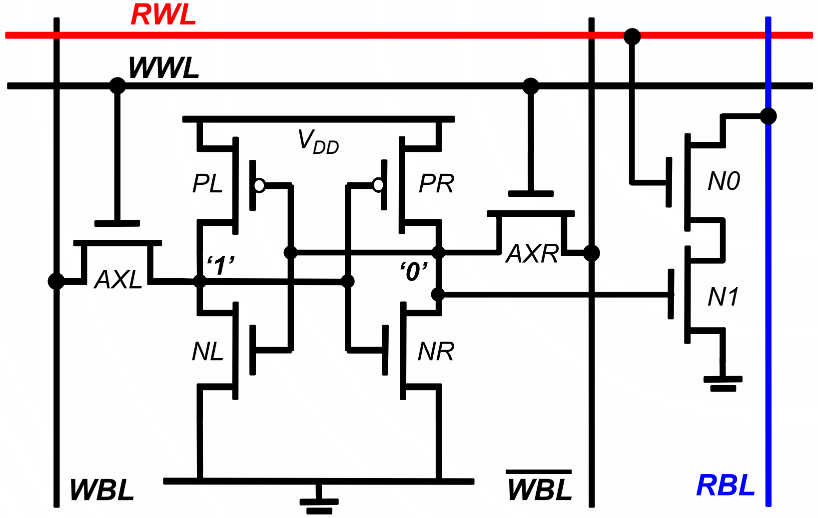
- Example cell: SRAM



- Repeat everything but cross-coupled inverters.
- This scheme extends up to a couple more ports, then need to add additional transistors.

1R/1W 8T SRAM

8-T SRAM



- Dual-port read/write capability
- Single-cycle read and write, timed appropriately
- Often found in register files, first level (L1) of cache

True or False

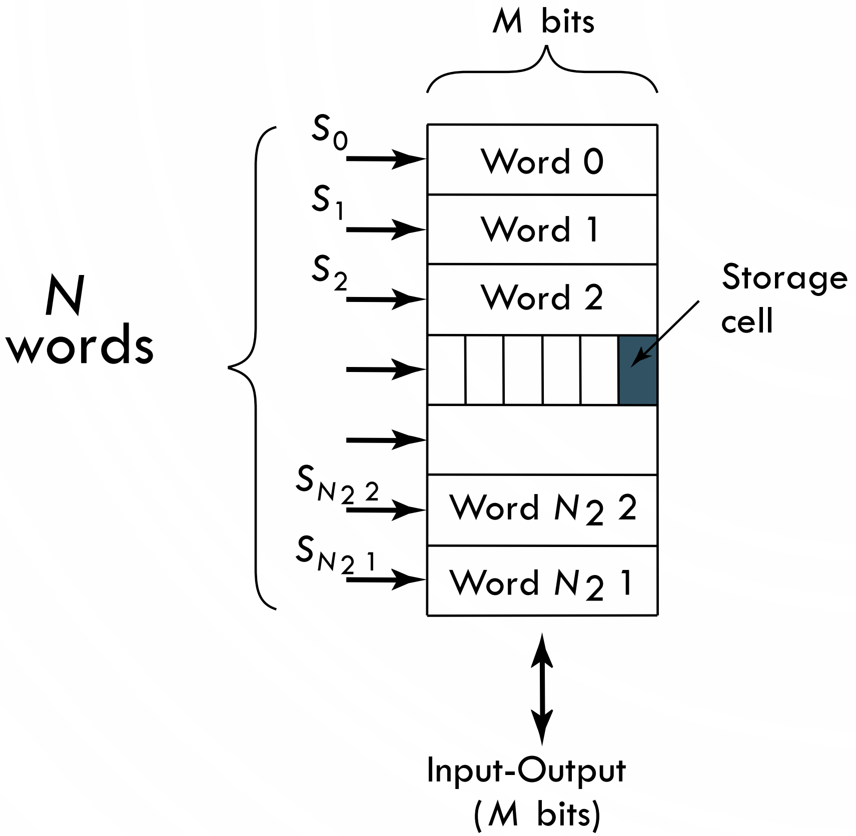
www.yellkey.com/pull

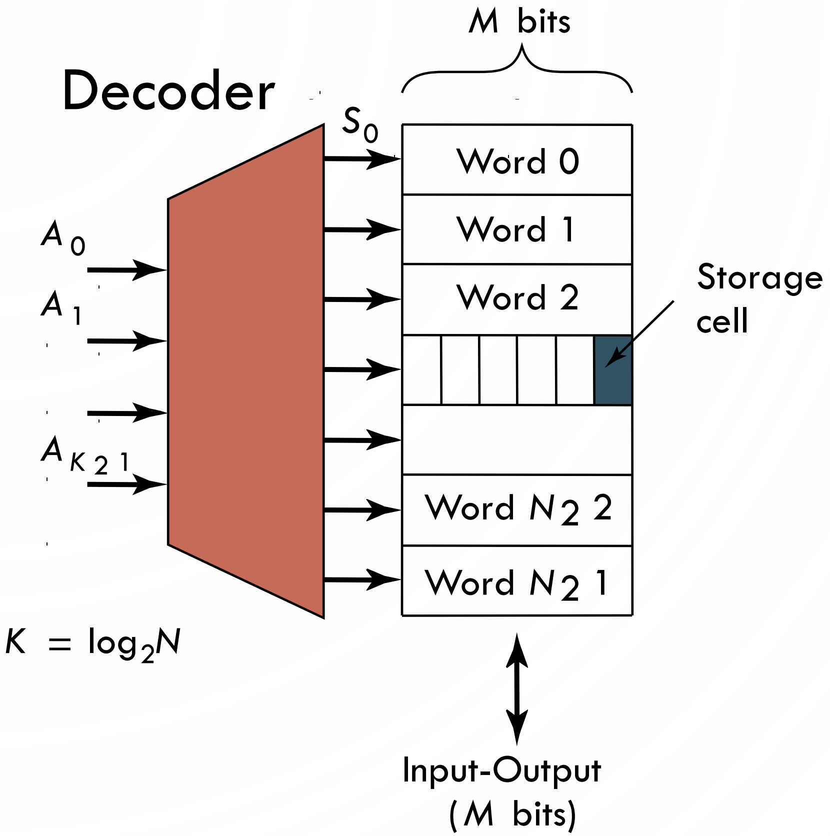
1	F	F	F
2	F	F	T
3	F	T	F
4	F	T	T
5	T	F	F
6	T	F	T
7	T	T	F
8	T	T	T

1. Transistor leakage doesn’t affect SRAM read speed
2. One should write into an SRAM cell by pulling BL high
3. One can only write into some cells of a selected WL

Memory Decoders

Decoders



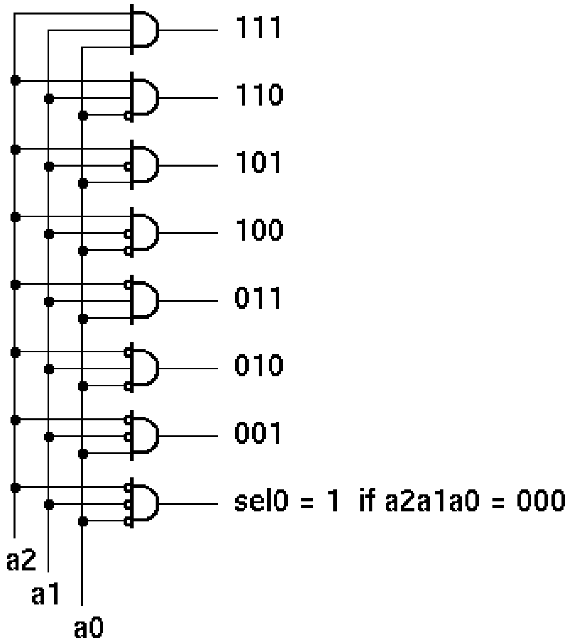


Intuitive architecture for N x M memory
Too many select signals:
N words = N select signals

Decoder reduces the number of select signals
$K = \log_2 N$

Row Decoders

Collection of 2^M complex logic gates
Organized in regular and dense fashion



(N)AND Decoder

$$WL_0 = \overline{A_0}\,\overline{A_1}\,\overline{A_2}\,\overline{A_3}\,\overline{A_4}\,\overline{A_5}\,\overline{A_6}\,\overline{A_7}\,\overline{A_8}\,\overline{A_9}$$

$$WL_{511} = A_0A_1A_2A_3A_4A_5A_6A_7A_8\overline{A_9}$$

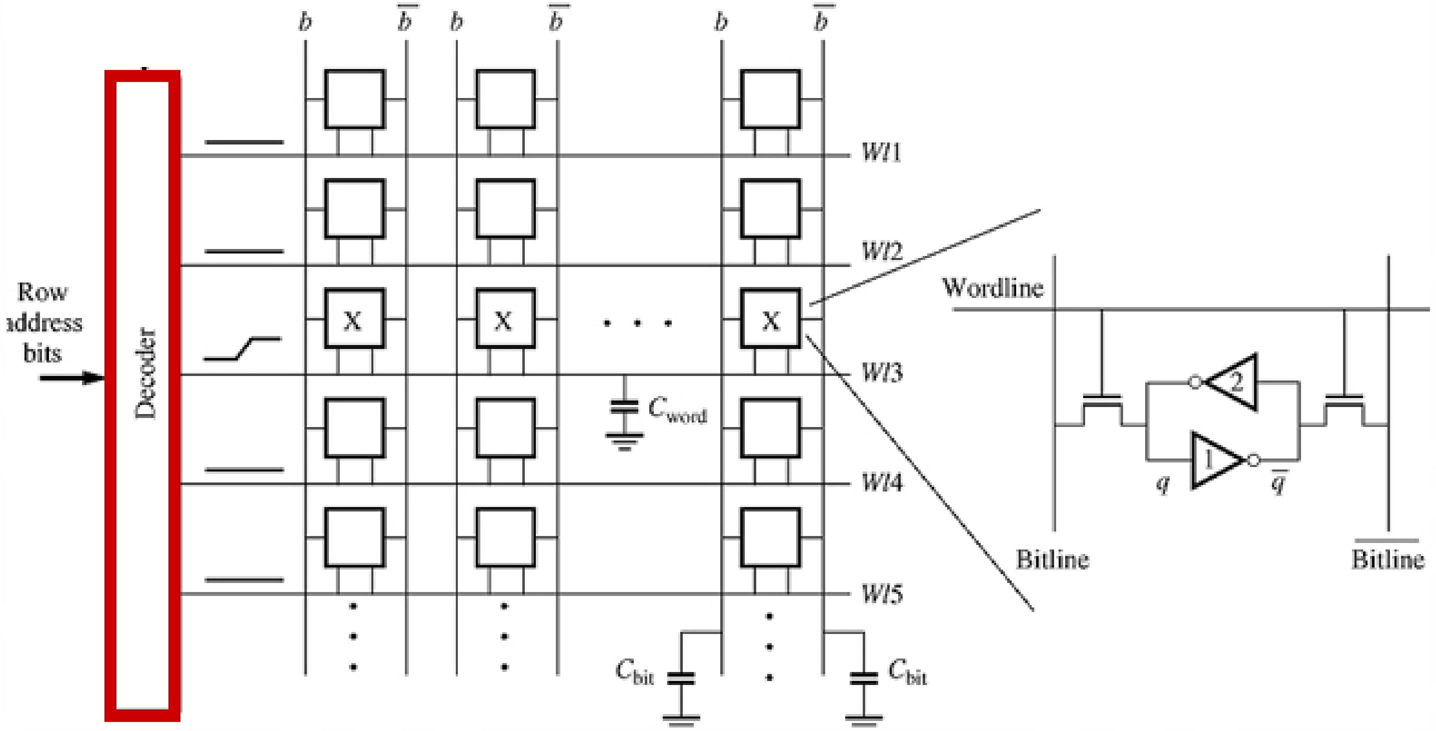
NOR Decoder

$$WL_0 = \overline{A_0 + A_1 + A_2 + A_3 + A_4 + A_5 + A_6 + A_7 + A_8 + A_9}$$

$$WL_{511} = \overline{\overline{A_0} + \overline{A_1} + \overline{A_2} + \overline{A_3} + \overline{A_4} + \overline{A_5} + \overline{A_6} + \overline{A_7} + \overline{A_8} + A_9}$$

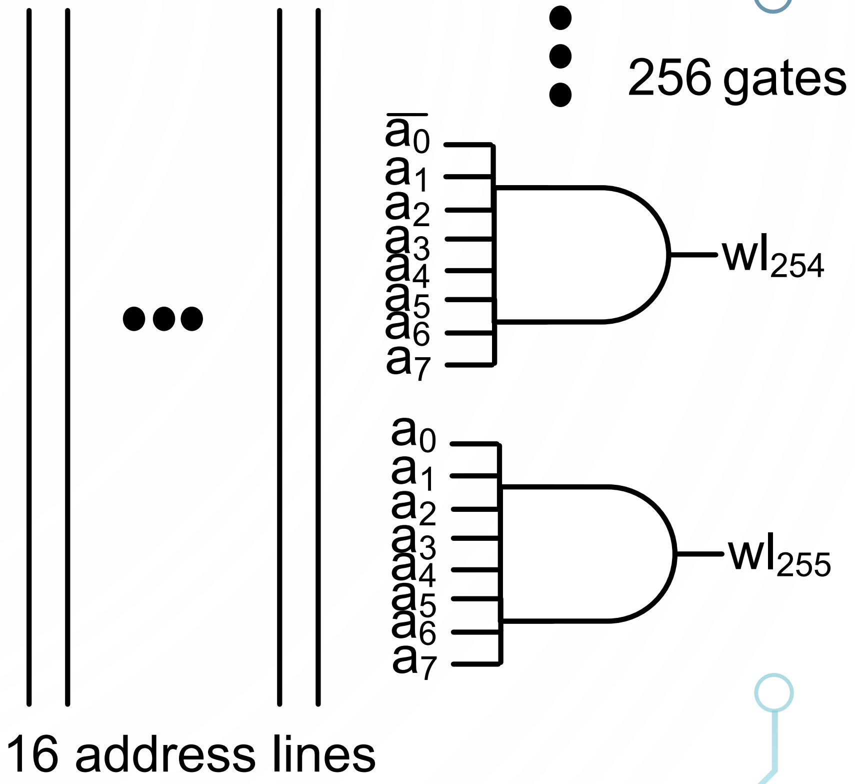
Decoder Design Example

- Look at decoder for 256x256 memory block (8KBytes)



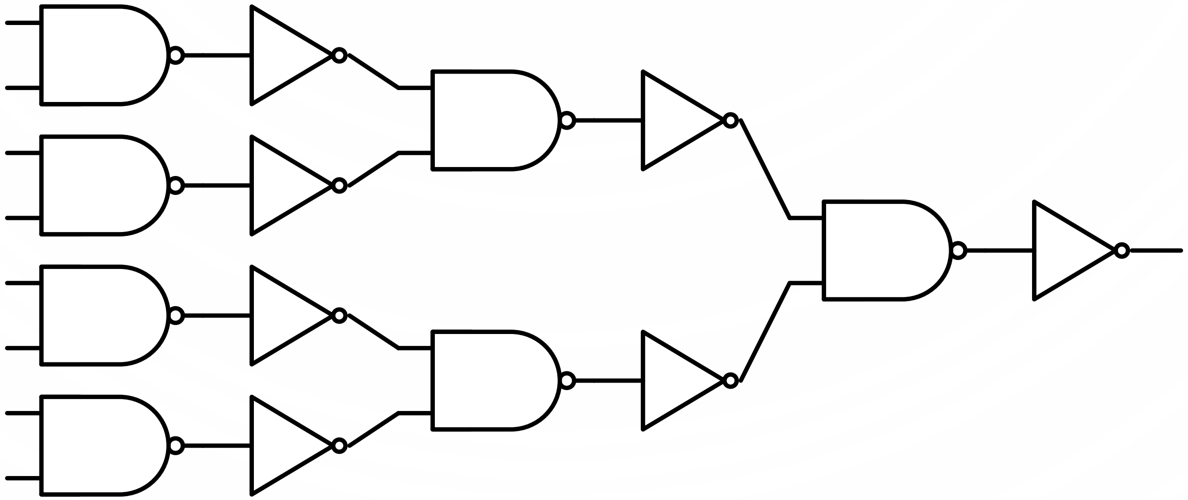
Possible Decoder

- 256 8-input AND gates
  - Each built out of a tree of NAND gates and inverters
- Need to drive a lot of capacitance (SRAM cells)
  - What's the best way to do this?



Possible AND8

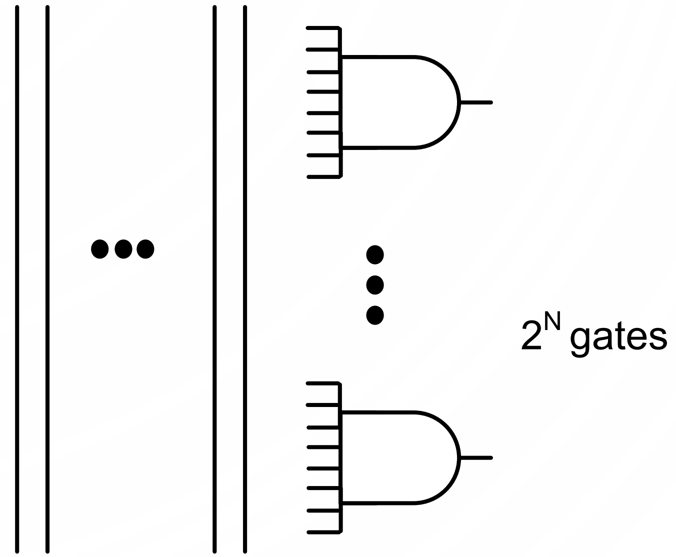
- Build 8-input NAND gate using 2-input gates and inverters
- Is this the best we can do?
- Is this better than using fewer NAND4 gates?



Problem Setup

- Goal: Build fastest possible decoder with static CMOS logic

- What we know
  - Basically need 256 AND gates, each one of them drives one word line



Problem Setup (1)

- Each wordline has 256 cells connected to it
- $C_{WL} = 256 * C_{cell} + C_{wire}$
  - Ignore wire for now
- Assume that decoder input capacitance is $C_{address} = 4 * C_{cell}$

Problem Setup (2)

- Each address bit drives $2^8/2$ AND gates
  - A0 drives ½ of the gates, A0_b the other ½ of the gates
- Total fanout on each address wire is:

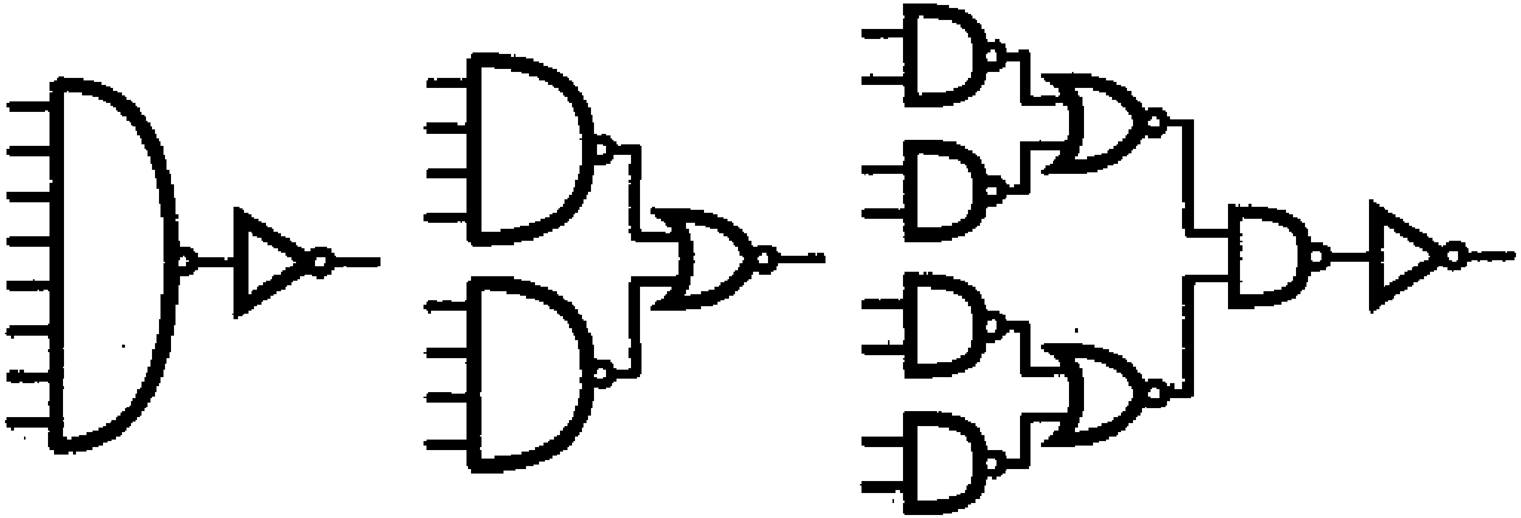
$$F = \Pi B \frac{C_{load}}{C_{in}} = 128 \frac{\left(256 C_{cell}\right)}{4 C_{cell}} = 2^7 \frac{\left(2^8 C_{cell}\right)}{2^2 C_{cell}} = 2^{13}$$

Decoder Fan-Out

- F of 2^{13} means that we will want to use more than $\log_4(2^{13}) = 6.5$ stages to implement the AND8

- Need many stages anyways
  - So what is the best way to implement the AND gate?
  - Will see next that it's the one with the most stages and least complicated gates

8-Input AND



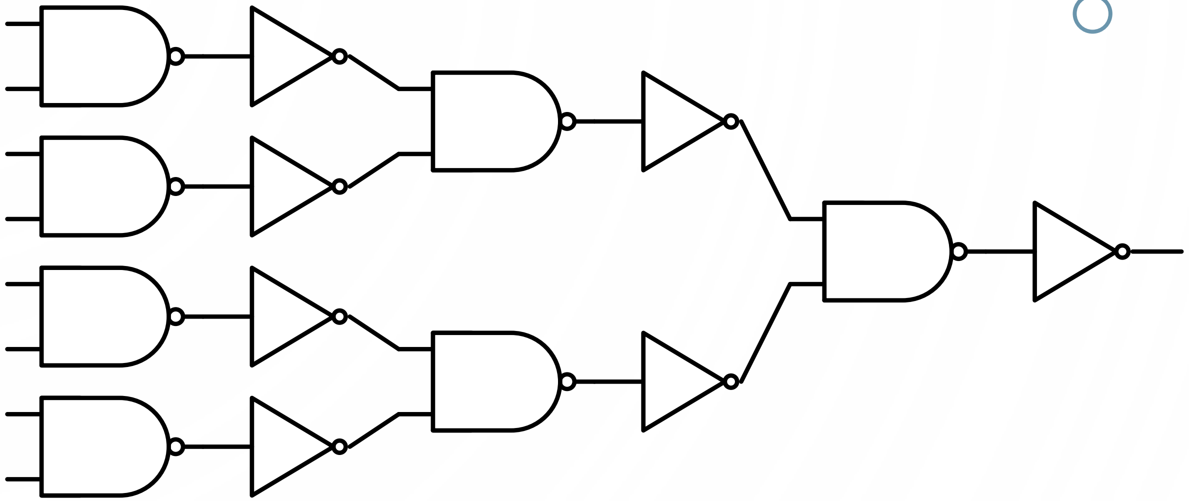
g : 9/2 1
G = 9/2
P = 8 + 1

g : 5/2 3/2
G = 15/4
P = 4 + 2

g : 3/2 3/2 3/2 1
G = 27/8
P = 2 + 2 + 2 + 1

8-Input AND

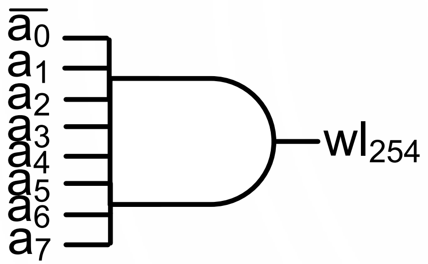
- Using 2-input NAND gates
  - 8-input gate takes 6 stages
- Total G is $(3/2)^3 \approx 3.4$
- So H is $3.4 * 2^{13}$ – optimal N of ~7.4

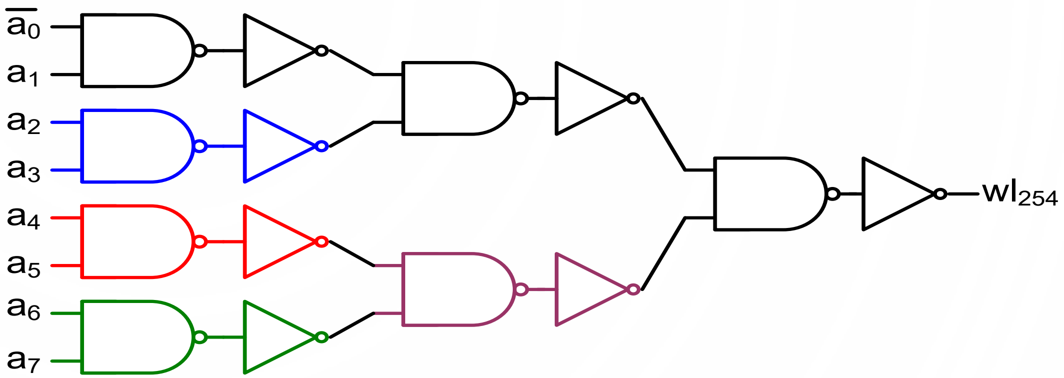


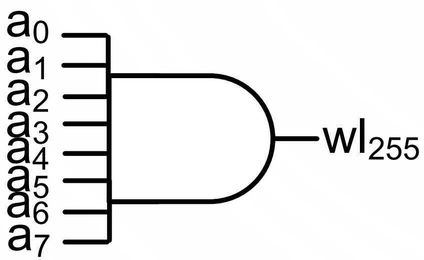
Decoder So Far

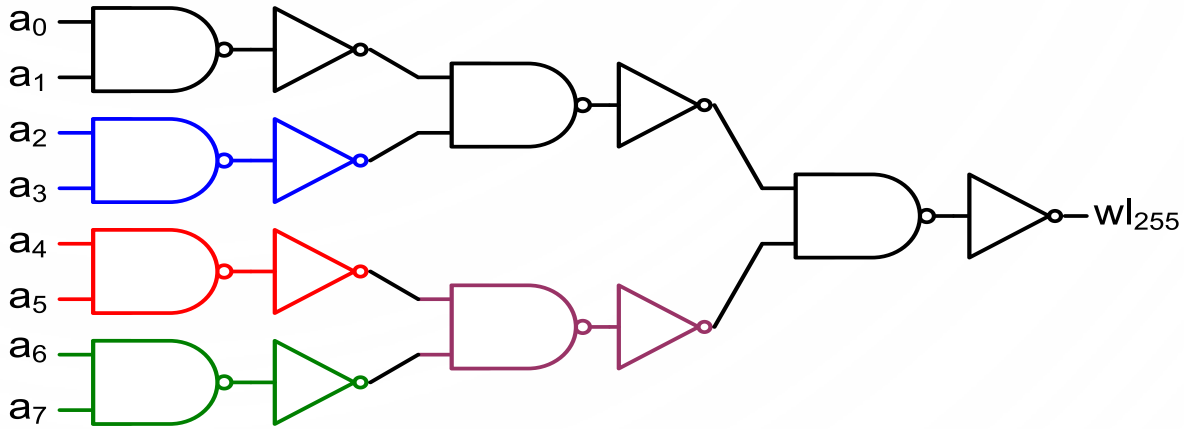
- 256 8-input AND gates
  - Each built out of tree of NAND gates and inverters
- Issue:
  - Every address line has to drive 128 gates (and wire) right away
  - Forces us to add buffers just to drive address inputs

Look Inside Each AND8 Gate







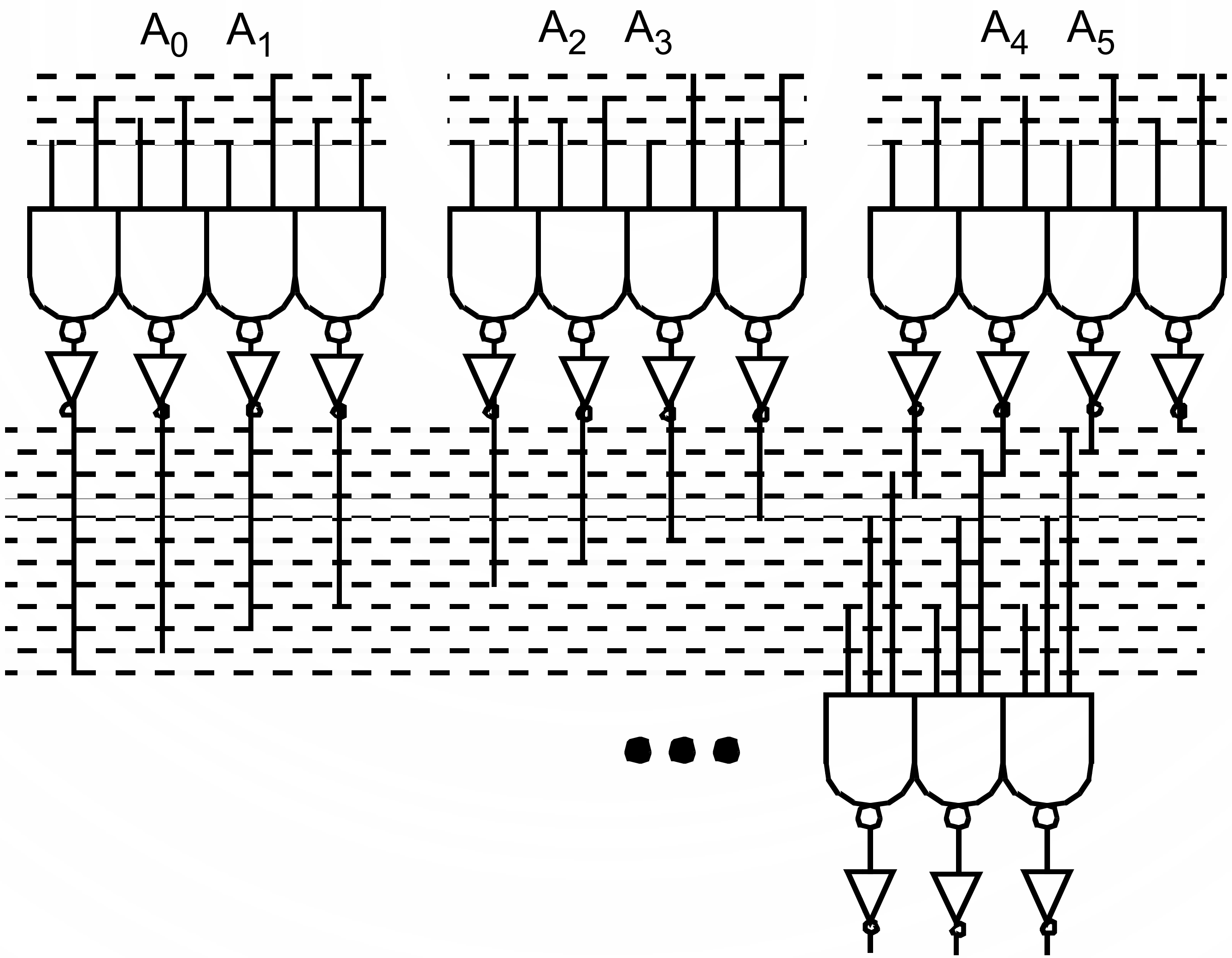


Predecoders

- Use a single gate for each of the shared terms
  - E.g., from A_0, $\overline{A_0}$, A_1, and $\overline{A_1}$, generate four signals: A_0A_1, $\overline{A_0}A_1$, $A_0\overline{A_1}$, $\overline{A_0}\,\overline{A_1}$

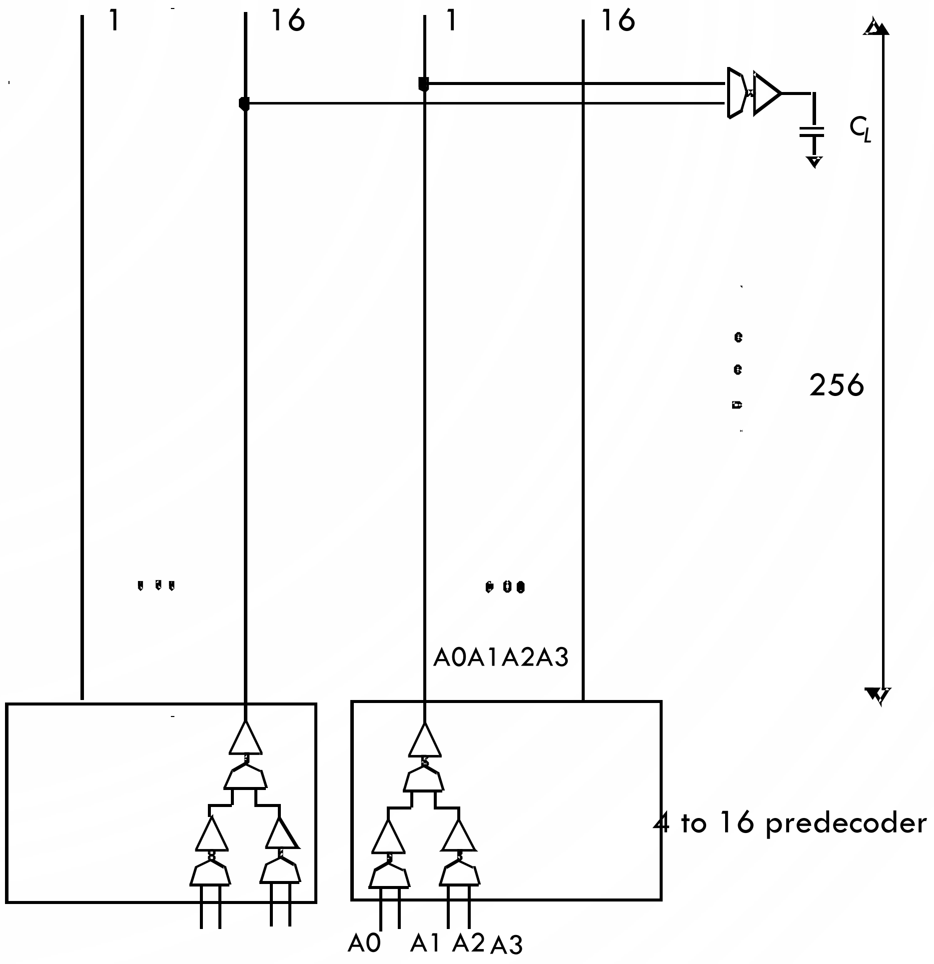
- In other words, we are decoding smaller groups of address bits first
  - And using the “predecoded” outputs to do the rest of the decoding

Predecoder and Decoder



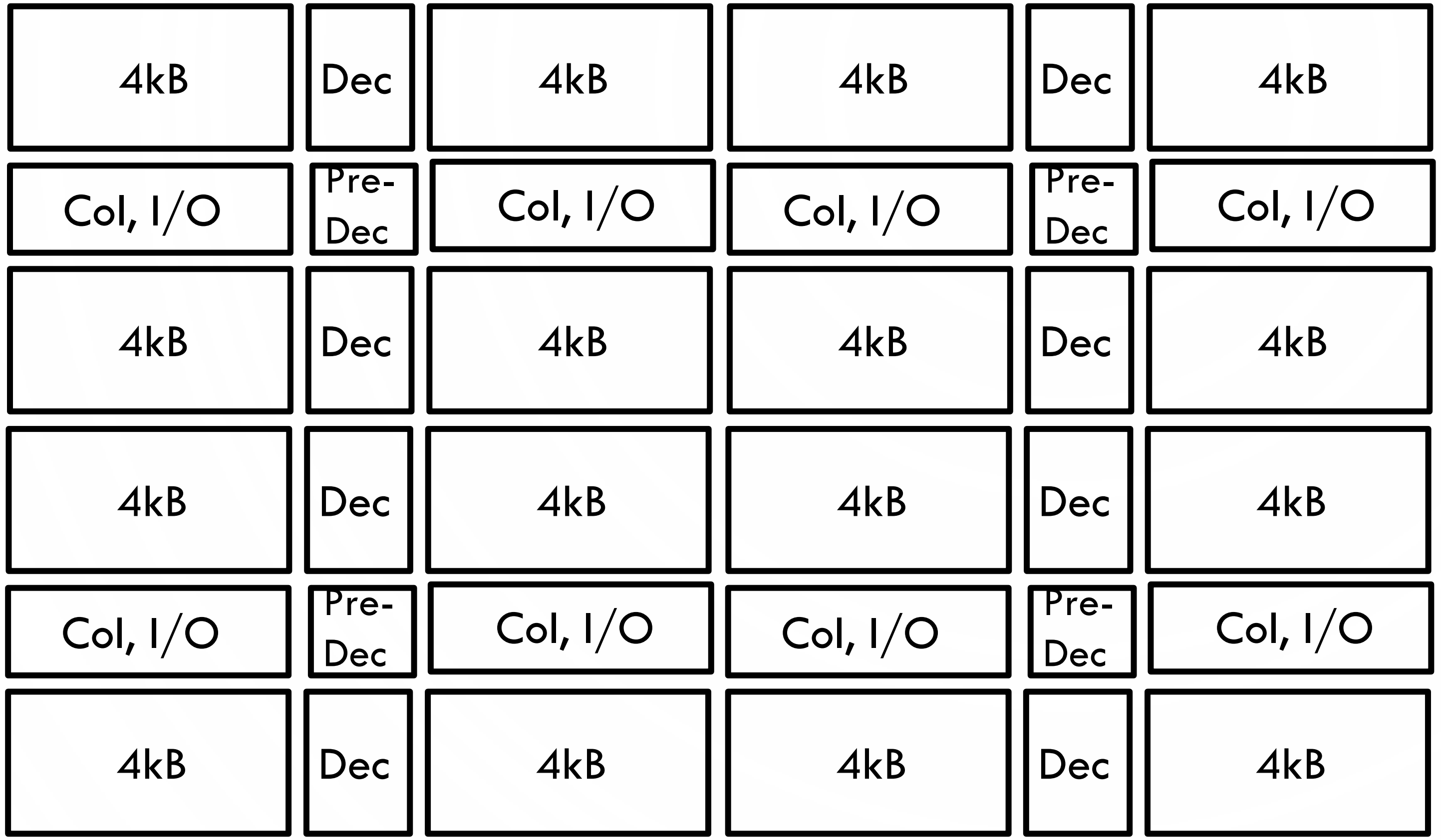
Predecode Options

- Larger predecode usually better:
- More stages before the long wires
  - Decreases their effect on the circuit
- Fewer number of long wires switches
  - Lower power
- Easier to fit 2-input gate into cell pitch



Building Larger Arrays

Building Larger Custom Arrays

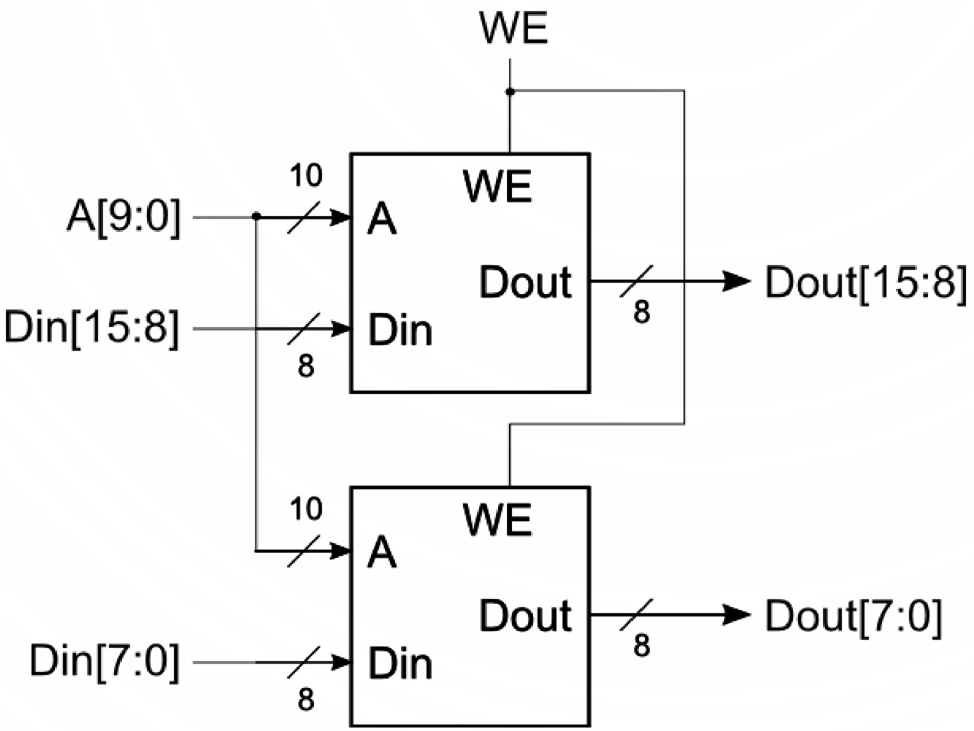


- Each subarray is 2-8kB
- Hierarchical decoding
- Peripheral overhead is 30-50%
- Delay is wire dominated
- Scratchpads, caches, TLBs

Cascading Memory-Blocks

How to make larger memory blocks out of smaller ones.

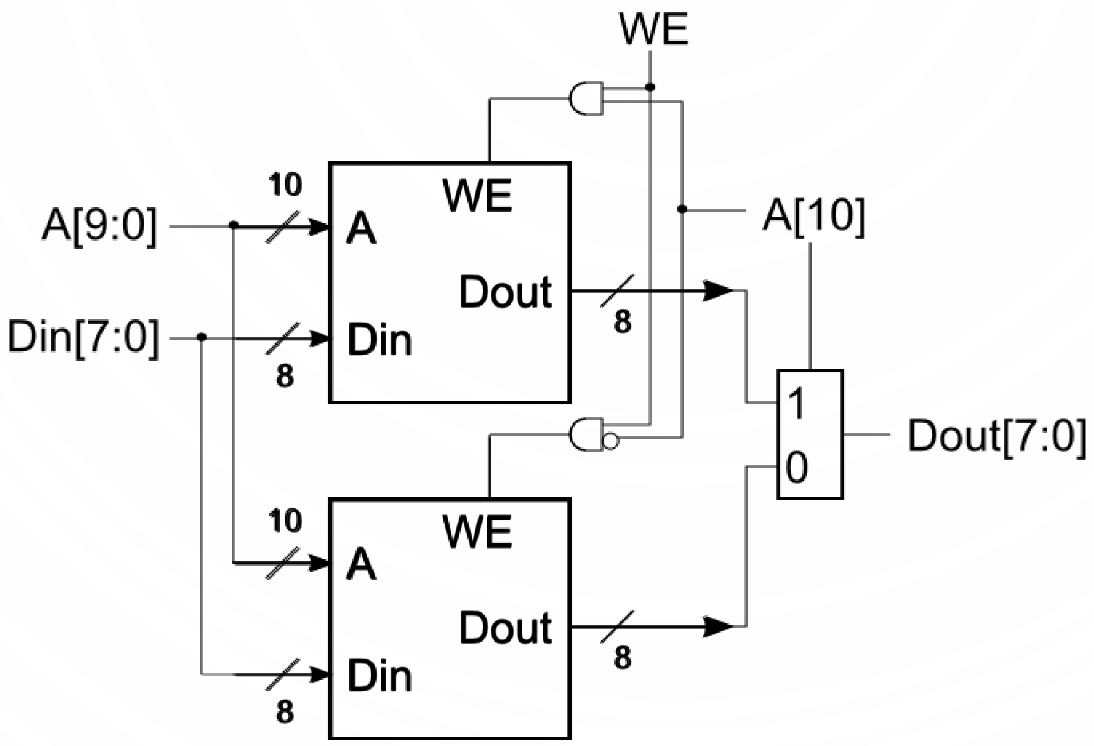
Increasing the width. Example: given 1Kx8, want 1Kx16



Cascading Memory-Blocks

How to make larger memory blocks out of smaller ones.

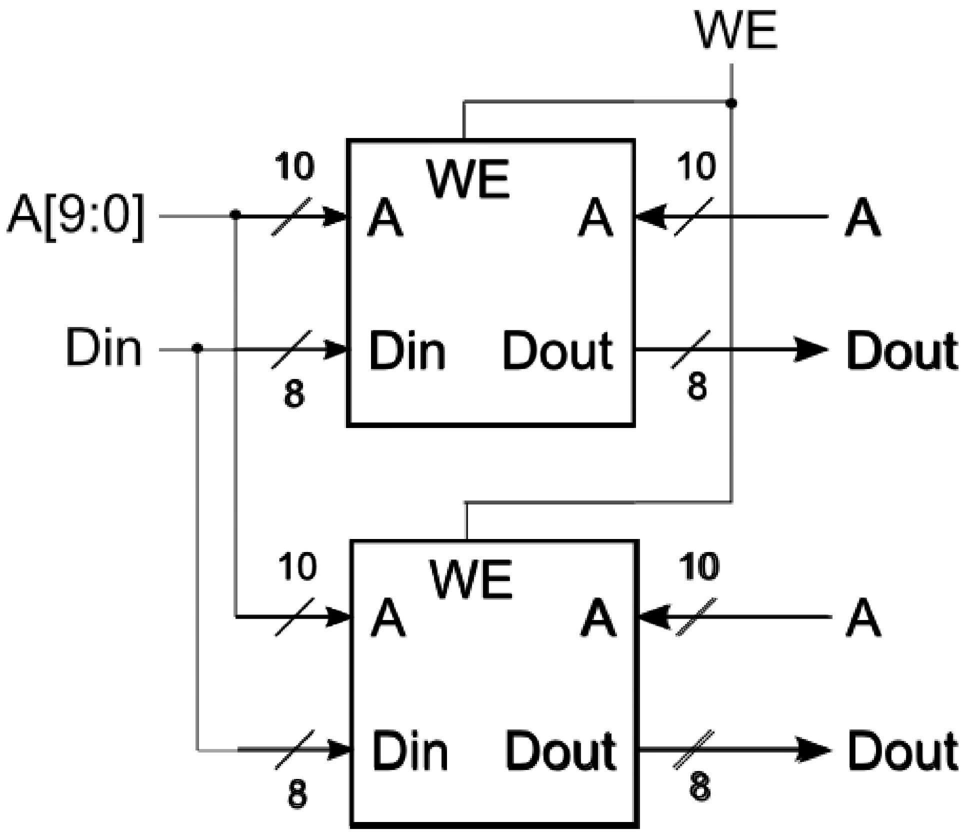
Increasing the depth. Example: given 1Kx8, want 2Kx8



Adding Ports to Primitive Memory Blocks

Adding a read port to a simple dual port (SDP) memory.

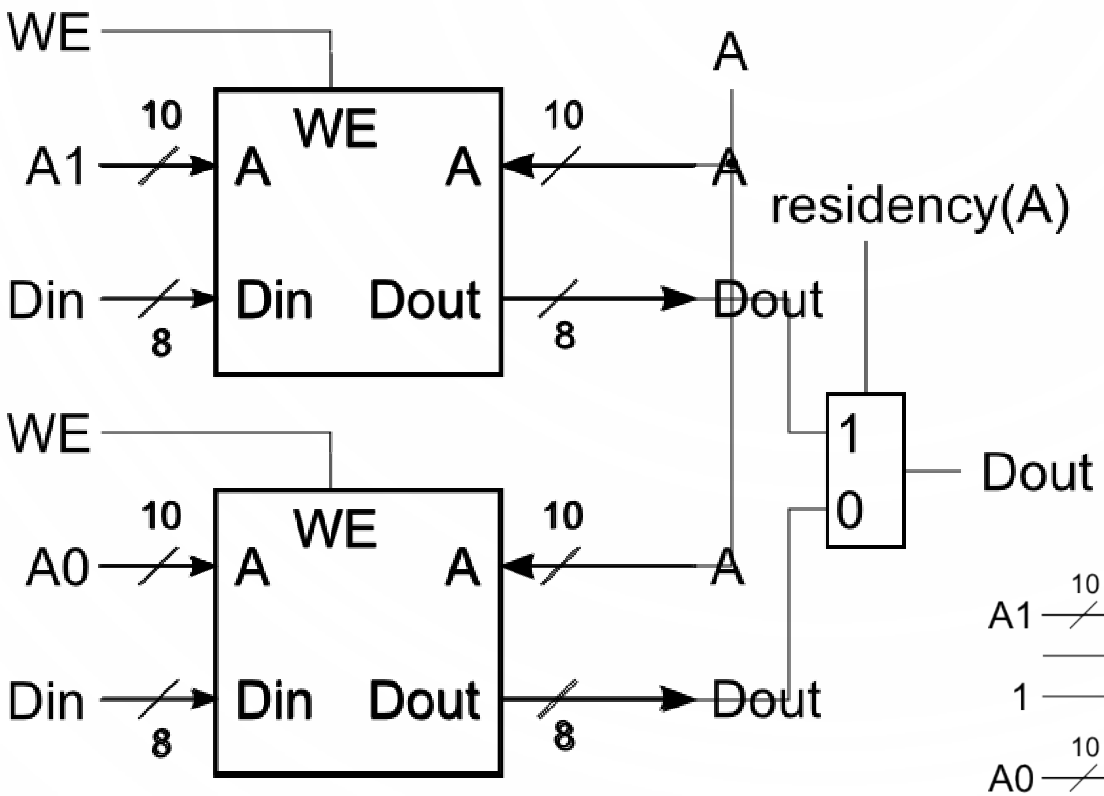
Example: given 1Kx8 SDP, want 1 write & 2 read ports.

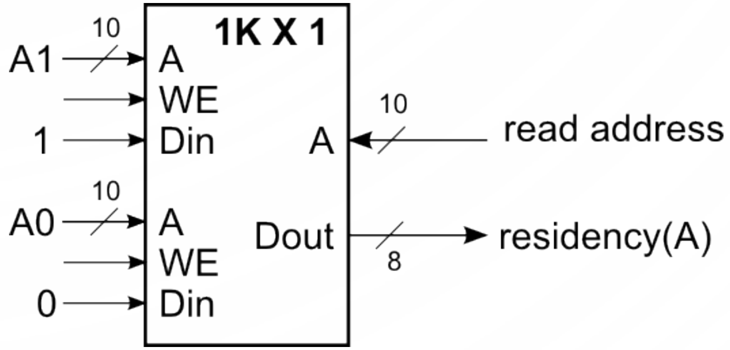


Adding Ports to Primitive Memory Blocks

How to add a write port to a simple dual-port memory.

Example: given 1Kx8 SDP, want 1 read & 2 write ports.





Review

- Dense memories are built as arrays of memory elements
  - SRAM is a static memory
- SRAM has unique combination of density, speed, power
- SRAM cells sized for stability and writeability
- SRAM and regfile cells can have multiple R/W ports
- Memory decoding is done hierarchically
  - Wire-limited in large arrays