Slide Set 9

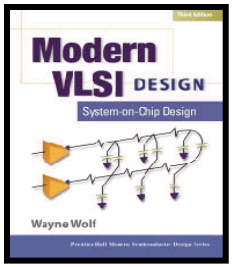
Memory

Steve Wilton
Dept. of ECE
University of British Columbia
stevew@ece.ubc.ca

.

Overview

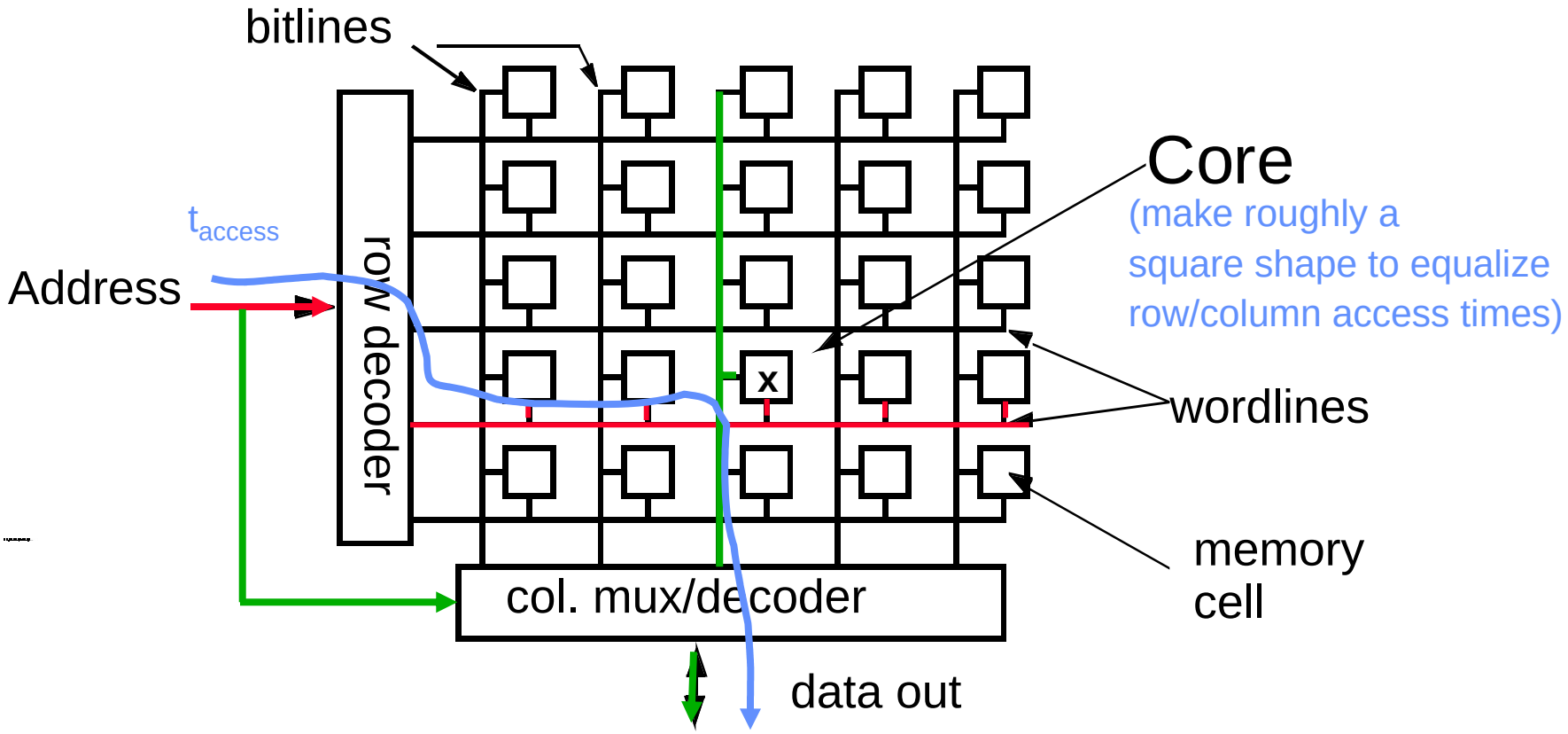
Wolf 6.7



Memories are one of the most useful VLSI building blocks. One reason for their utility is that memory arrays can be extremely dense. This density results from their very regular wiring.

Memories come in many different types (RAM, ROM, EEPROM) and there are many different types of cells, but the basic idea and organization is pretty similar. We will look at the most common memory cell that is used today, a 6T sRAM cell, and then look at the other components needed to build complete memory system.

Memory Array



It has N^2 elements and only 2N wires. It is an easy way to use millions of transistors. The layout is quite dense since they are composed of snap-together cells.

Memory Cell Options

Often need to have either large number of bits or high-speed operation:

- In some cases more bits are needed: use simple cell design but complex control signals and refresh during the operation
- In other cases, speed is critical: use a complex cell, but keep control as simple as possible; also no need to refresh during operation
- If you have enough cells, it is ok to make peripheral circuits more complex

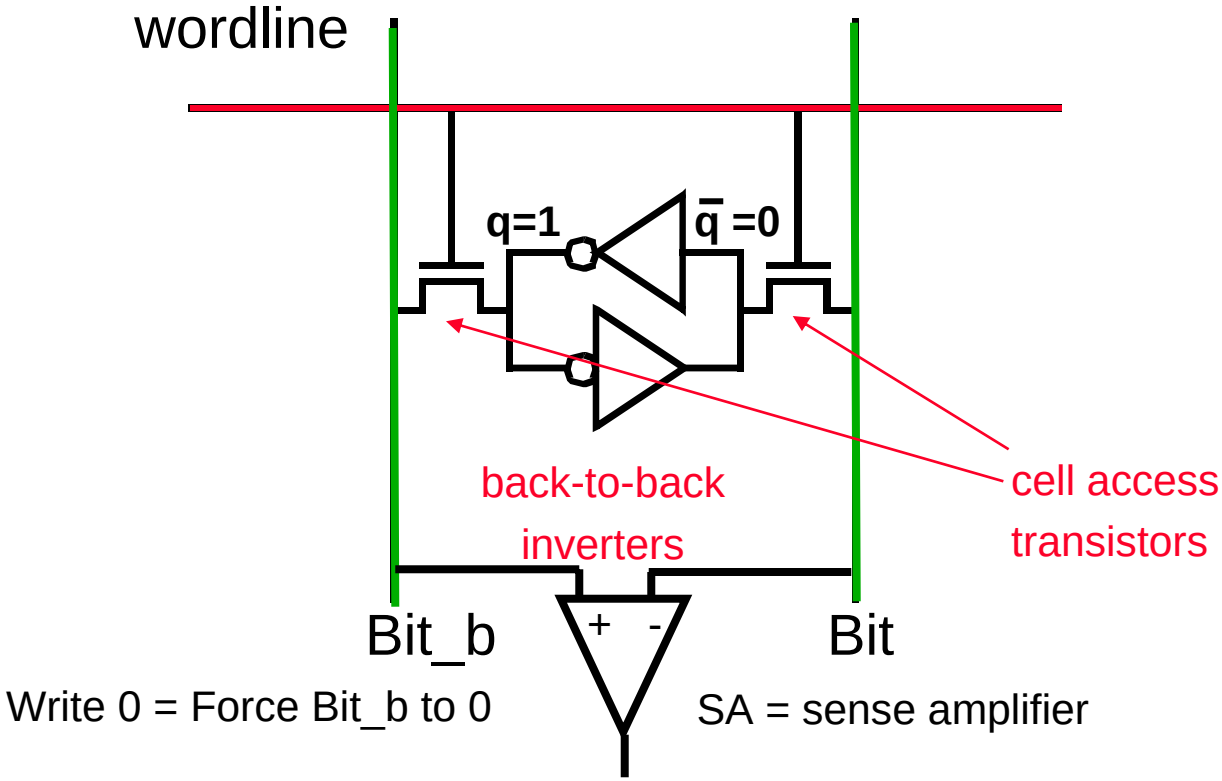
Leads to many innovative cell designs:

- 6T RAM cells
- 4T RAM cell with poly loads
- 1T DRAM cell
- And lots of strange layouts

We will look at the 6T RAM, which is the key to all memory cells

6T Static RAM Cell

Uses only six transistors:

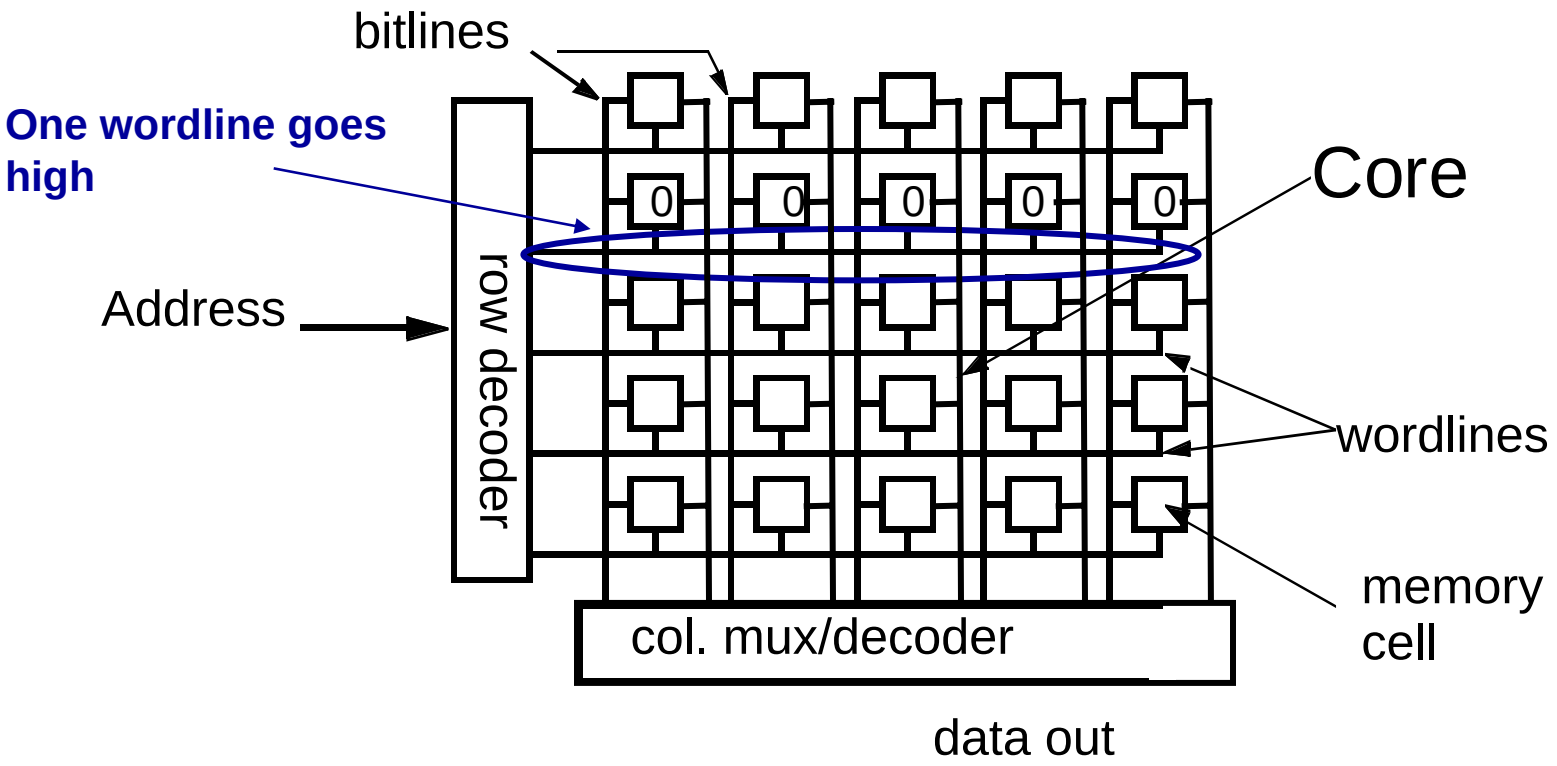


Read and write use the same port. There is one wordline and two bit lines. The bit lines carry the data. The cell is small since it has a small number of wires.

SRAM Cell Operation

Read:

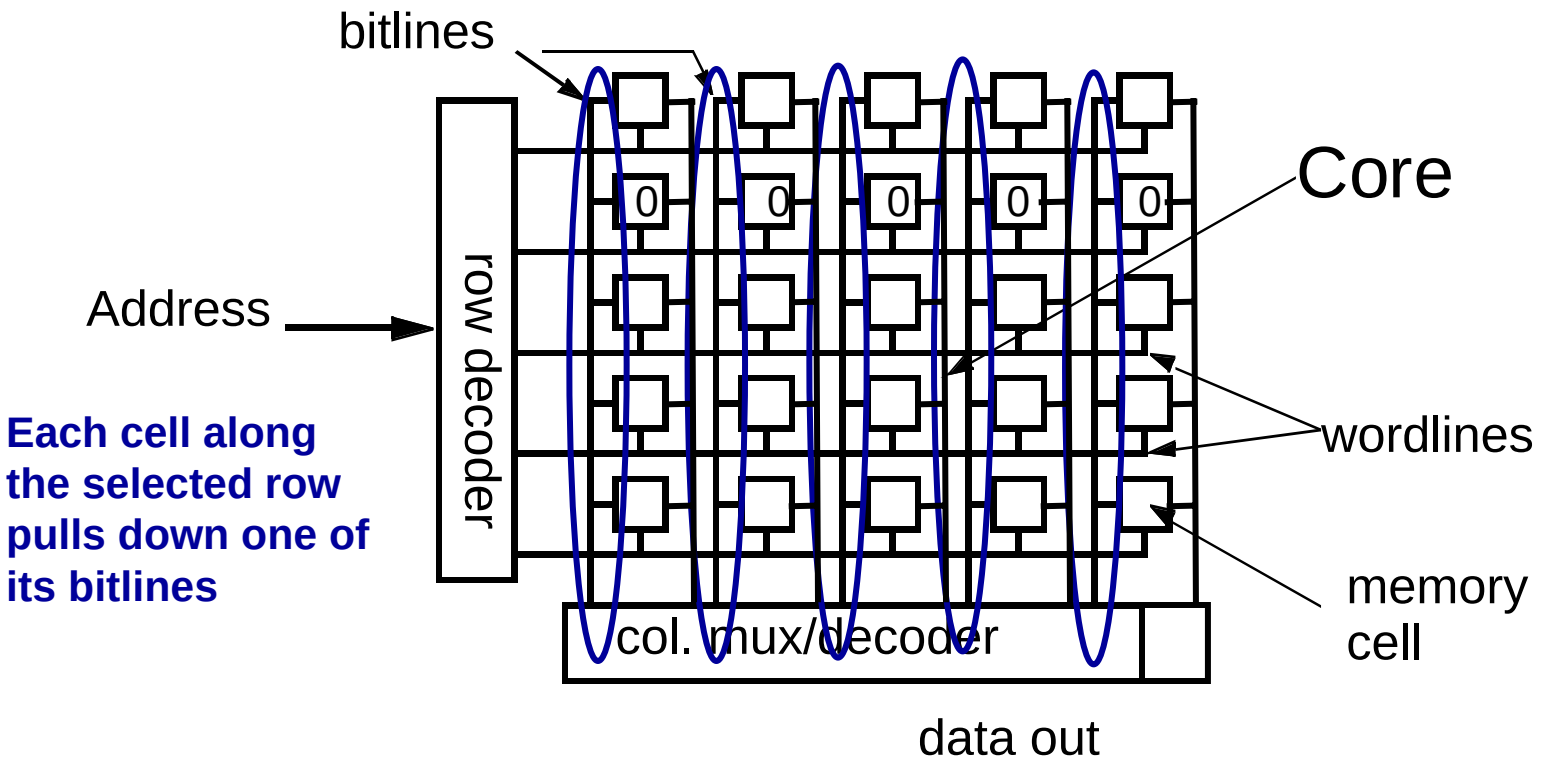
- Both Bit and $\overline{\text{Bit}}$ must start high. One wordline (one row) goes high. The cell will pull one of the lines low



SRAM Cell Operation

Read:

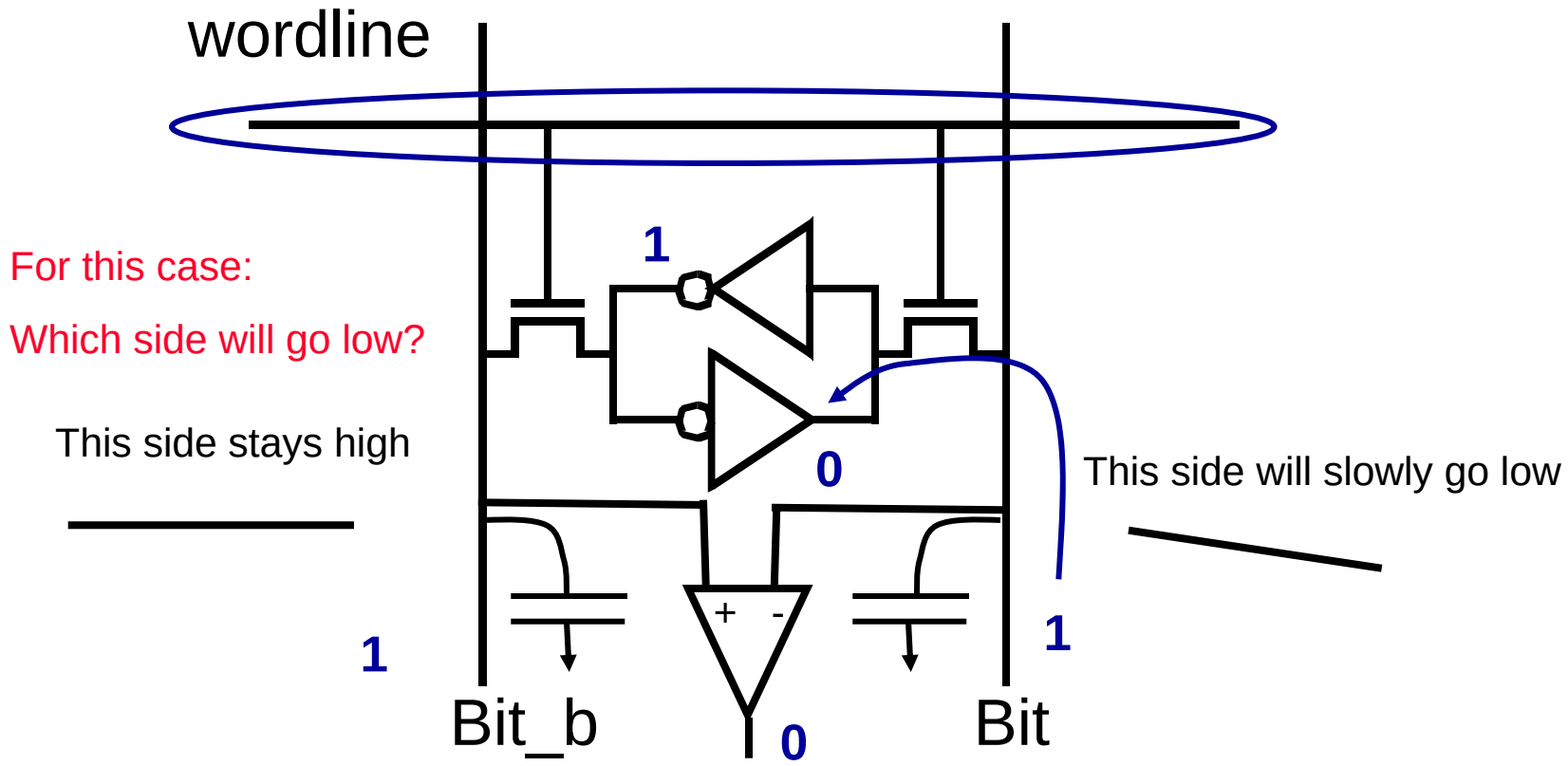
- Both Bit and $\overline{\text{Bit}}$ must start high. One wordline (one row) goes high. The cell will pull one of the lines low



SRAM Cell Operation

Read:

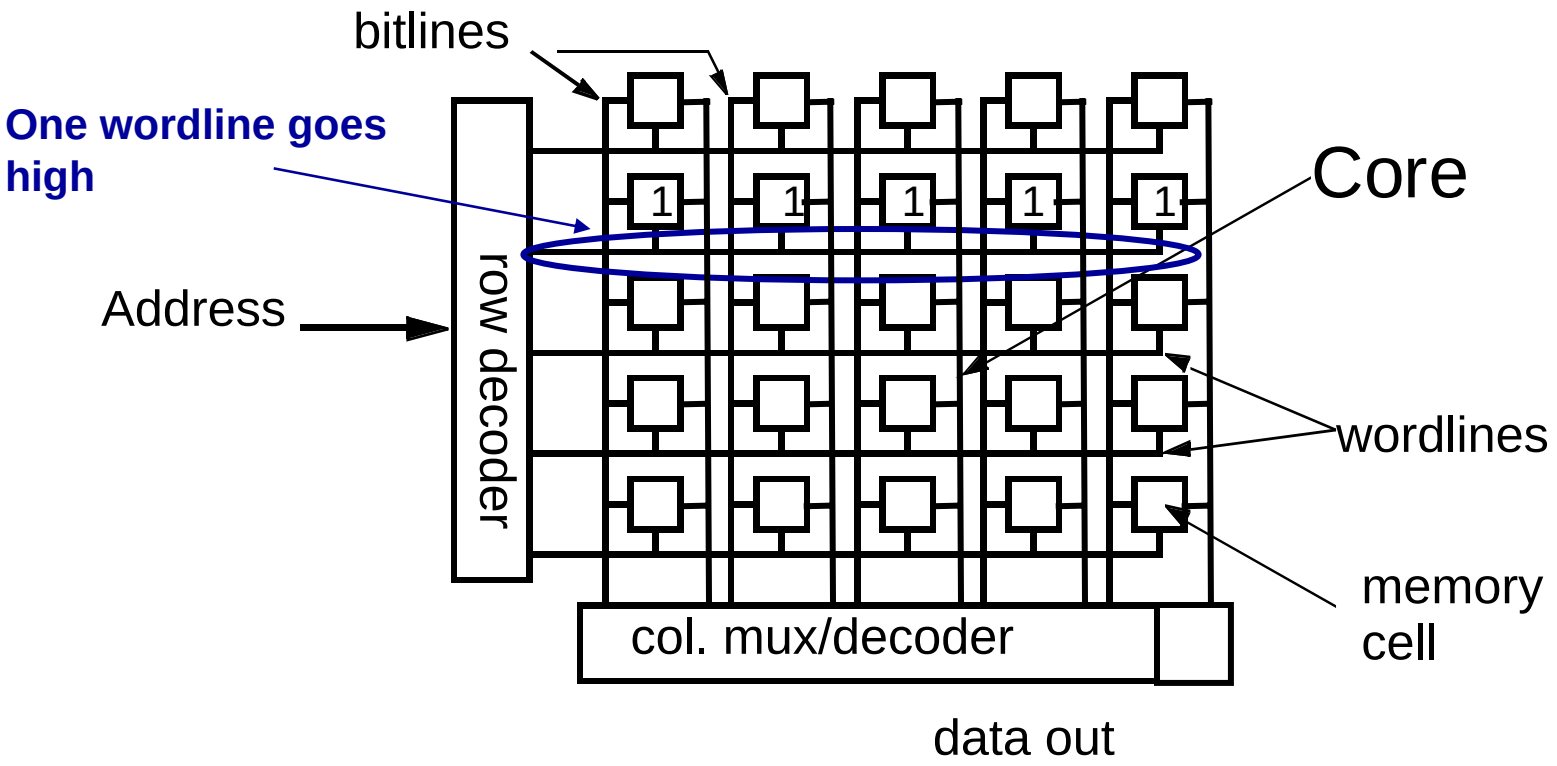
- Both Bit and $\overline{\text{Bit}}$ must start high. One wordline (one row) goes high. The cell will pull one of the lines low



SRAM Cell Operation

Write:

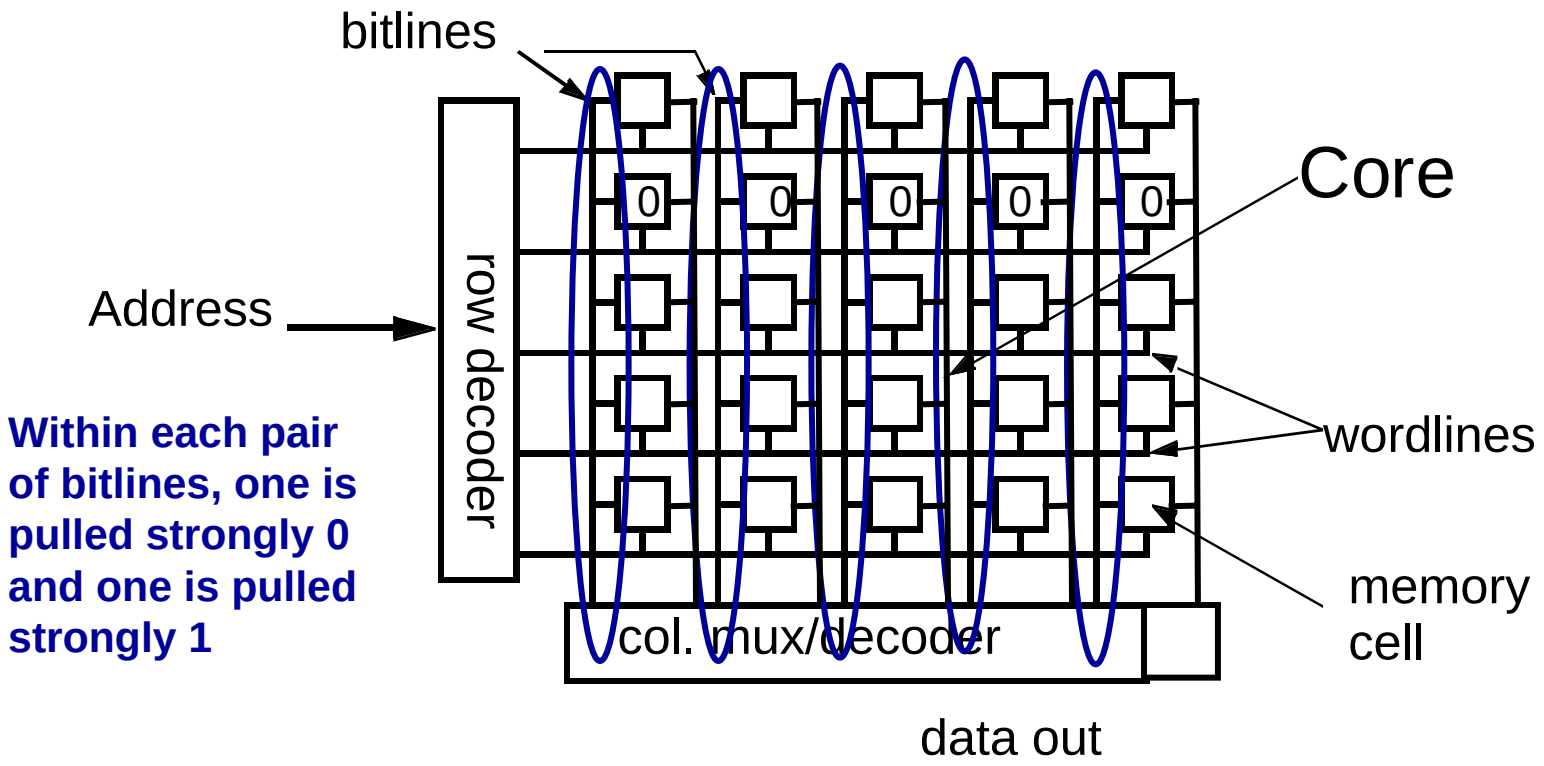
- One (Bit or $\overline{\text{Bit}}$) is forced low, the other is high. This low value overpowers the pMOS in the inverter, and this will write the cell.



SRAM Cell Operation

Write:

- One (Bit or $\overline{\text{Bit}}$) is forced low, the other is high. This low value overpowers the pMOS in the inverter, and this will write the cell.

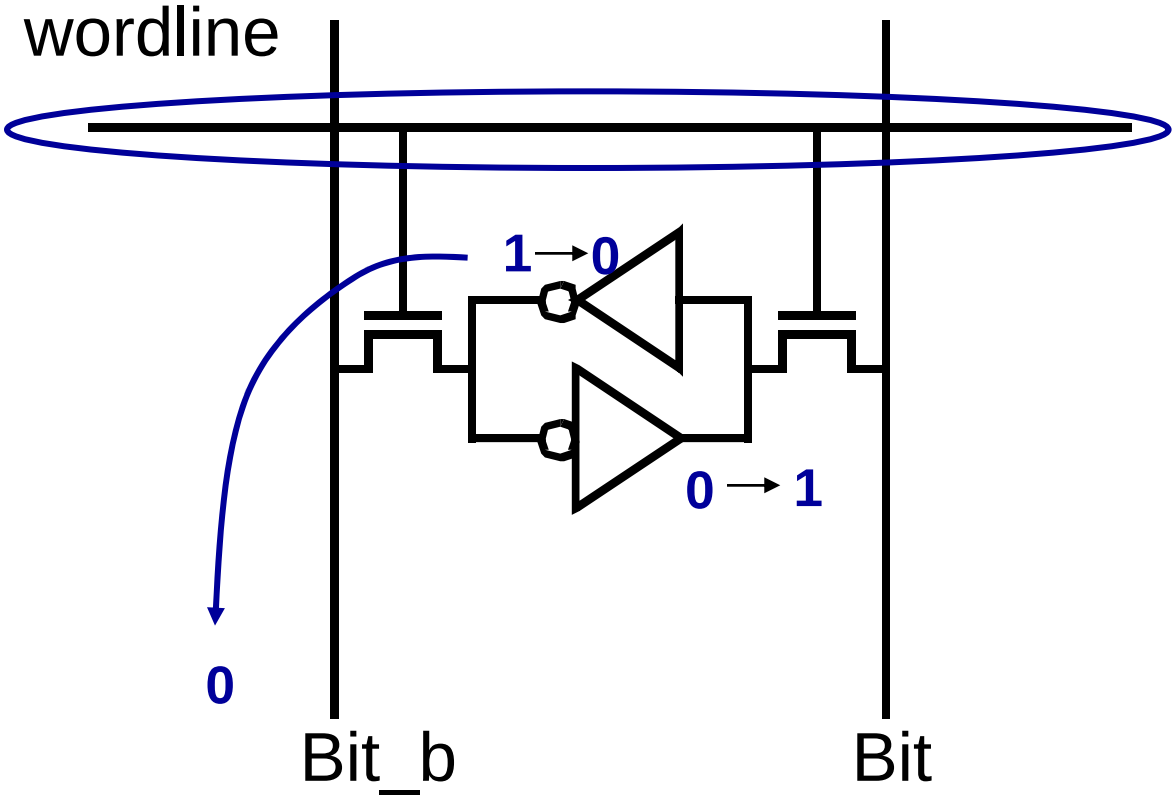


Within each pair of bitlines, one is pulled strongly 0 and one is pulled strongly 1

SRAM Cell Operation

Write:

- One (Bit or $\overline{\text{Bit}}$) is forced low, the other is high. This low value overpowers the pMOS in the inverter, and this will write the cell.

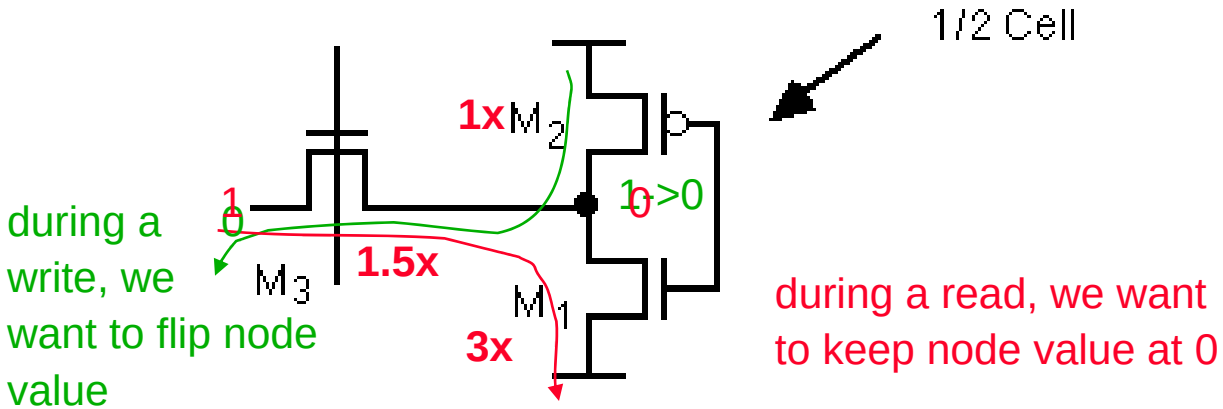


SRAM Cell Design

For the cell to work correctly, a “0”on the bit line must over power the pMOS pull up (for a proper write operation), but a “1” on the bit line must not over power the pull down (for a proper read operation)

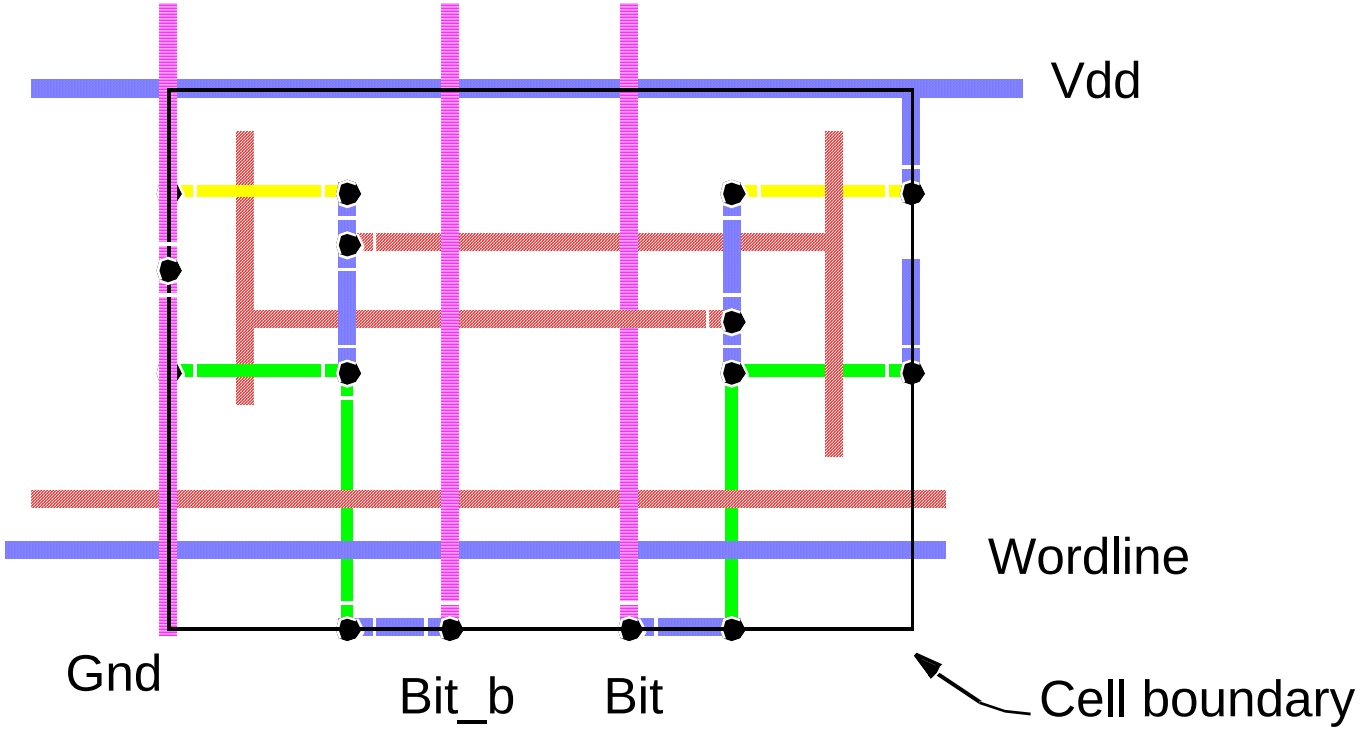
For the write case, M3 is passing a zero, so for it to overpower the pMOS it must be at least as wide (preferably 1.5x as wide). This gives a 2-3:1 current ratio between the nMOS and the pMOS.

For the read case, M3 is passing a one so it is somewhat weaker. Still M3 should be 1.5 to 2x smaller than M1 to make sure a read does not disturb the value of the cell.



SRAM Cell Layout

There are many clever SRAM layouts. This is a common one:



This layout is fairly dense, since the most of the contacts (bitline, Vdd, Gnd) are shared. Also, a clever cross-coupling method is used.

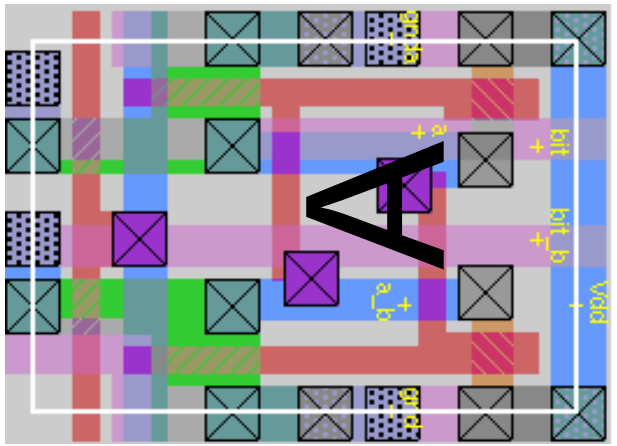
SRAM Layout

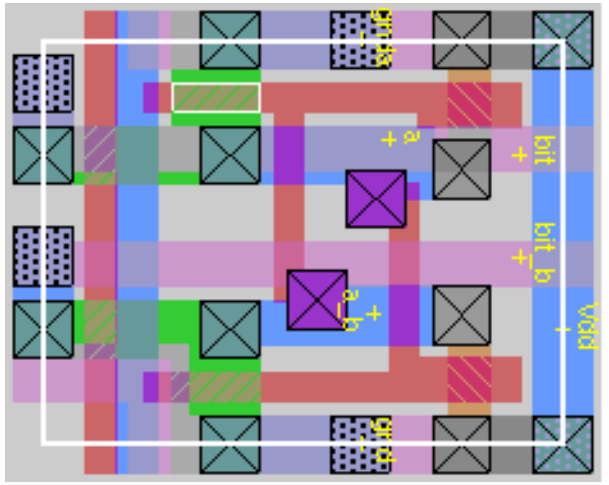
A conservative cell:

- substrate and well connects in each cell
- It has a wordline poly contact in each cell
- pMOS transistors are very weak (3:3)
- nMOS pulldown is 8:2
- All the boundaries are shared
- 41 x 28, about 1/4 the size of latch cell

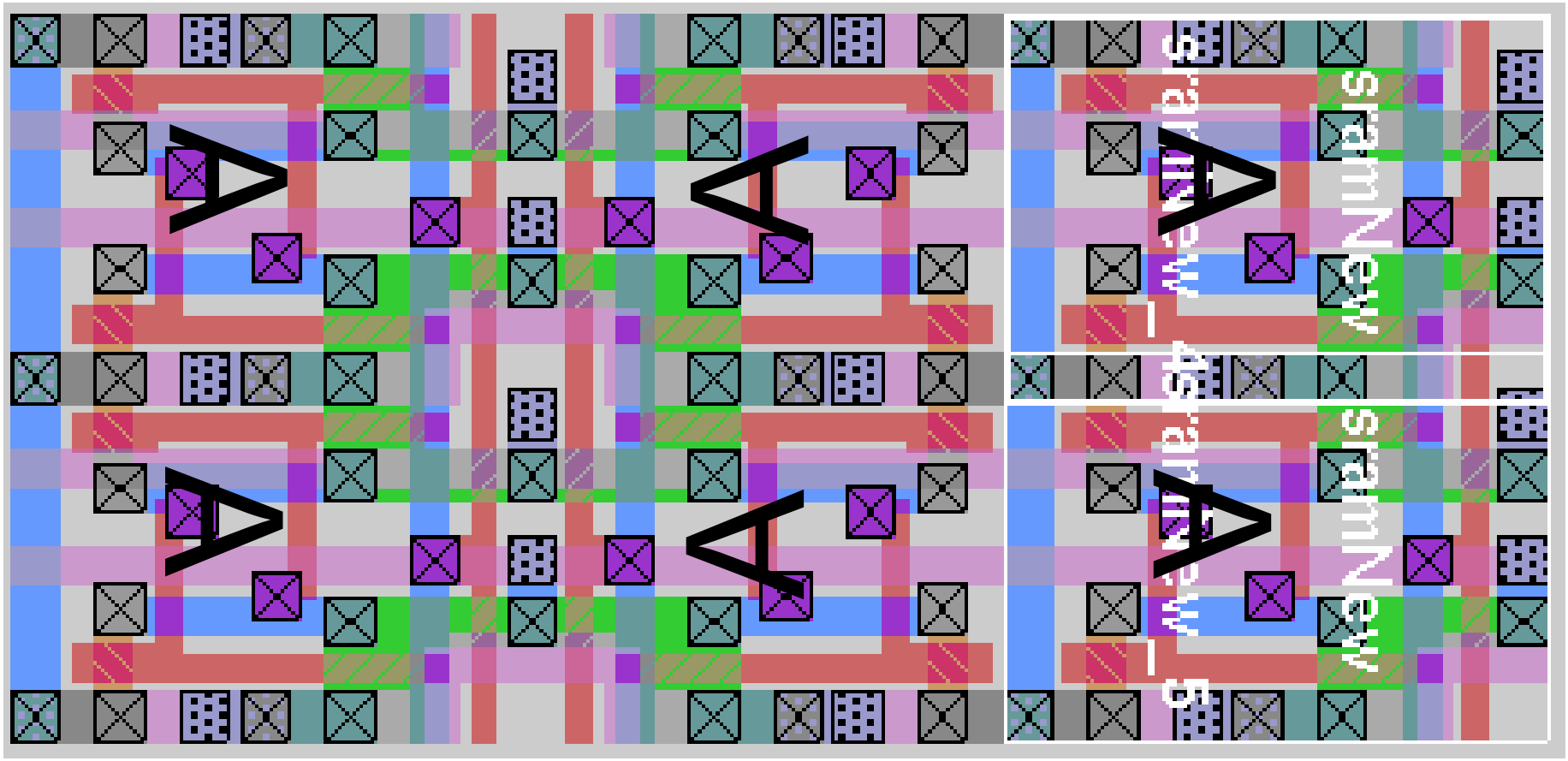
A slightly smaller cell

- Only nwell contact in cell
- pMOS transistors are very weak (3:3)
- nMOS pulldown is 6:2
- 36 x 28





SRAM Array

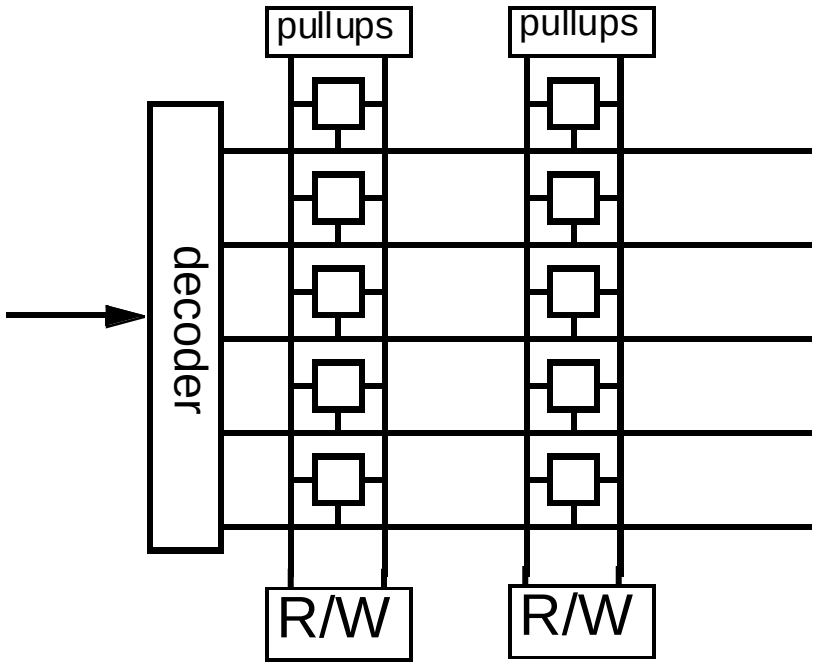


This is an array of 3 cells wide and 2 high.

- Shows how the contacts are shared

Peripheral Circuits

We need to build the I/O circuits for read/write, decoders and wordline drive circuits, and the column select and bitline drive circuits. Lets look at building each of these circuits for CMOS memories.



Bitline I/O Circuit (precharge for a read)

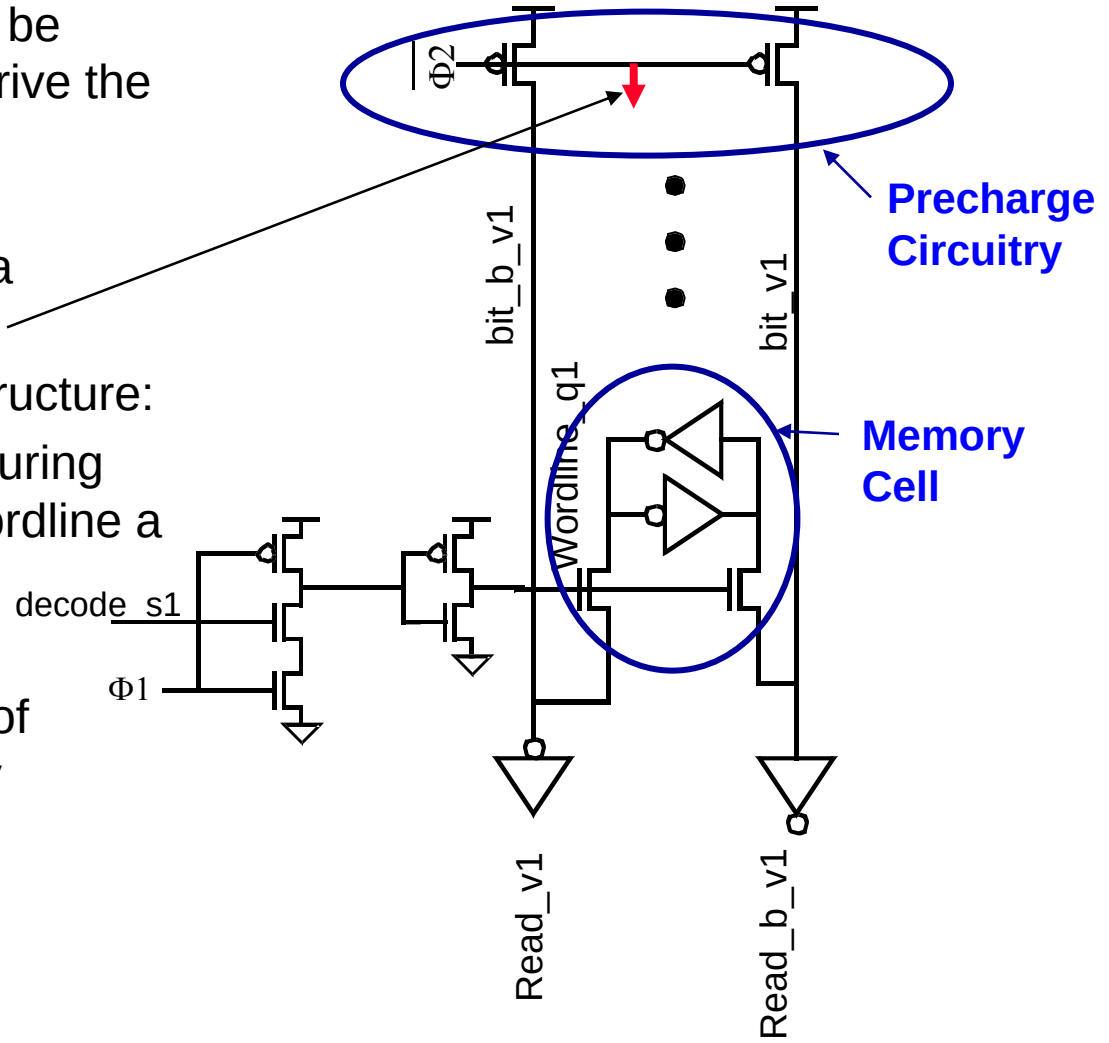
For reads both bitlines must be high, for write you need to drive the bitlines to the correct value.

- Bitlines need to be precharged, or use a pseudo nMOS load

We will use a precharged structure:

- To avoid a conflict during precharge, make wordline a qualified clock.

Bitlines are like the outputs of normal precharge gates - _v signals

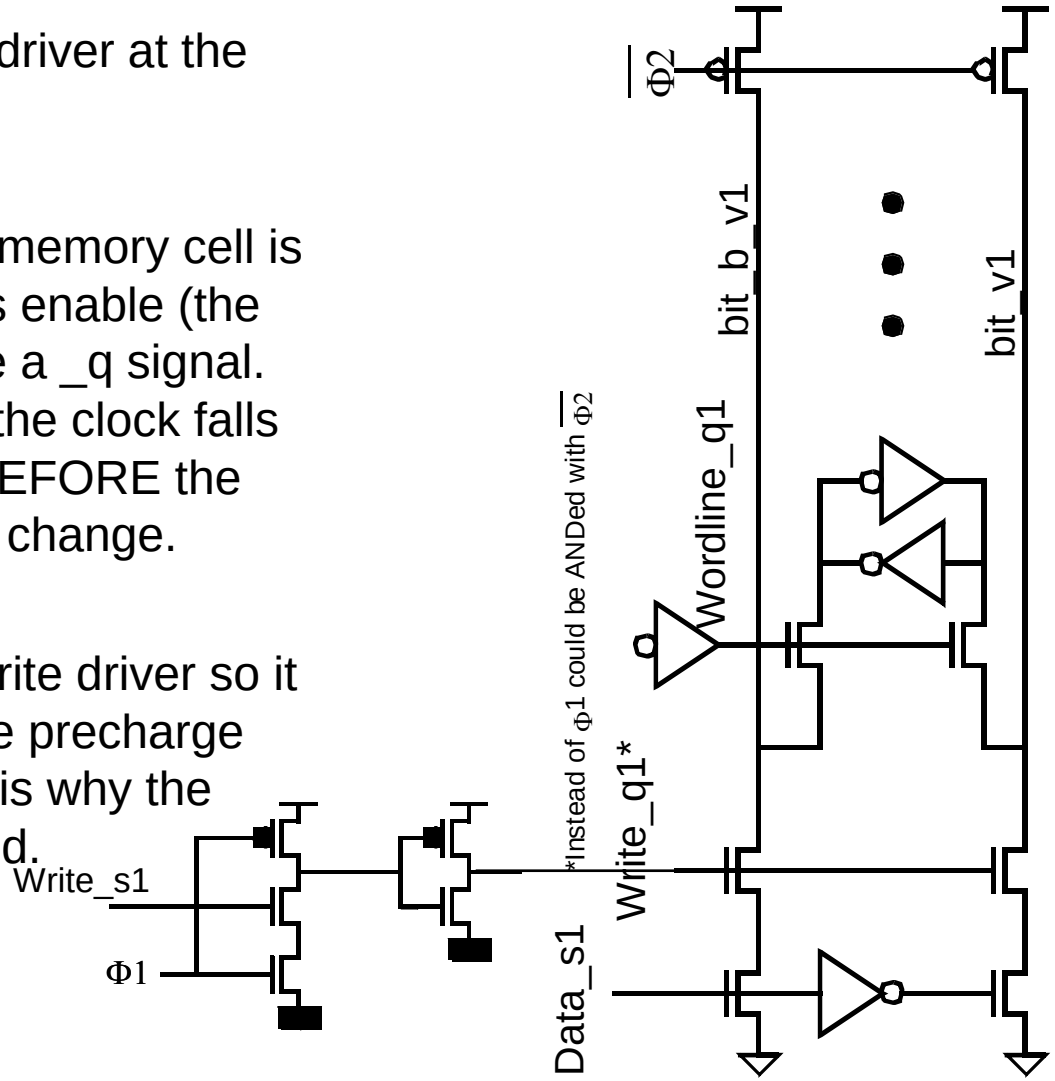


Bitline I/O Circuit (for a write)

We've added a write driver at the bottom.

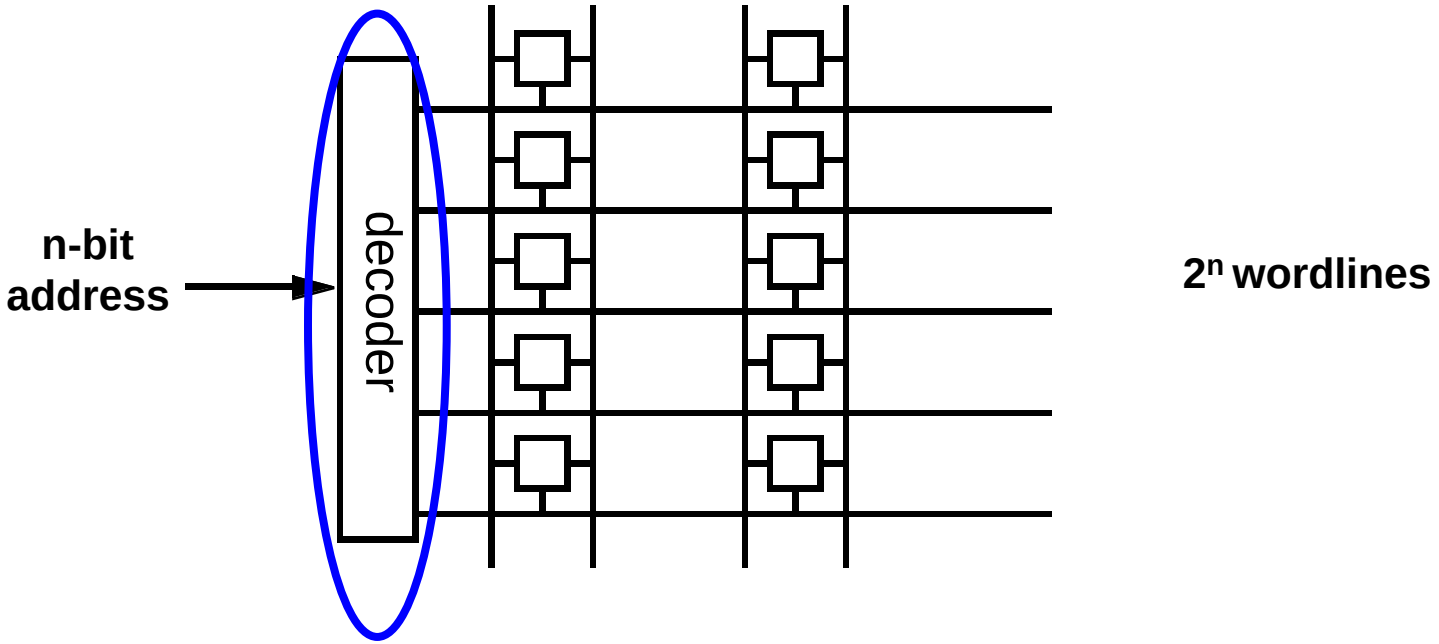
Notice that since the memory cell is a storage element, its enable (the wordline) needs to be a _q signal. That will ensure that the clock falls latching in the data BEFORE the data has a chance to change.

Need to isolate the write driver so it does not fight with the precharge (power issue), which is why the write signal is qualified.



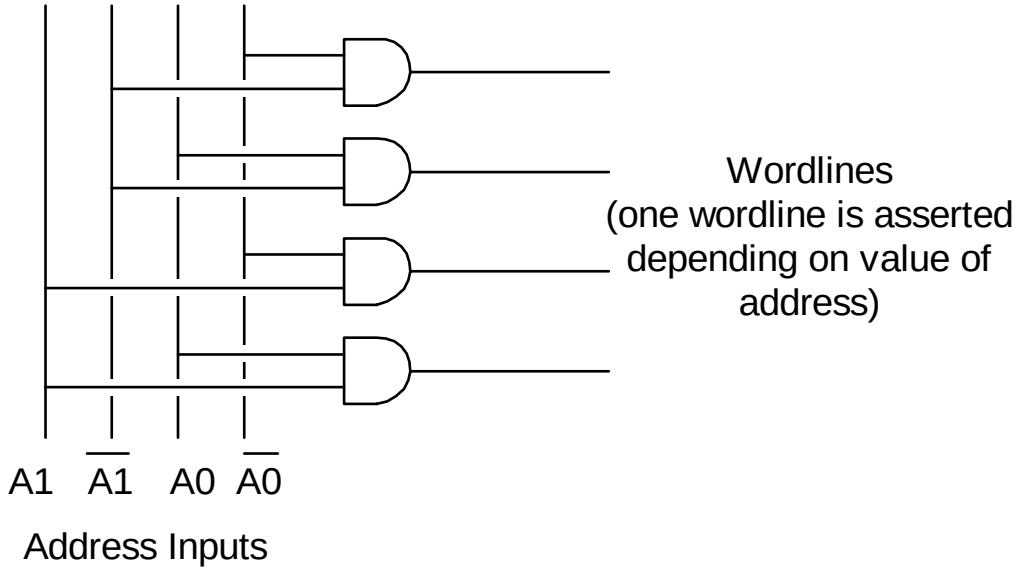
Decoders

A decoder decodes an n-bit address. The outputs of the decoder are the wordlines. Since there are n address bits, there are 2^n wordlines. Depending on the address, exactly one wordline is asserted (which activates exactly one row in the memory).



Decoders

A decoder is just a structure that contains a number of AND gates, where each gate is enabled for a different input value.

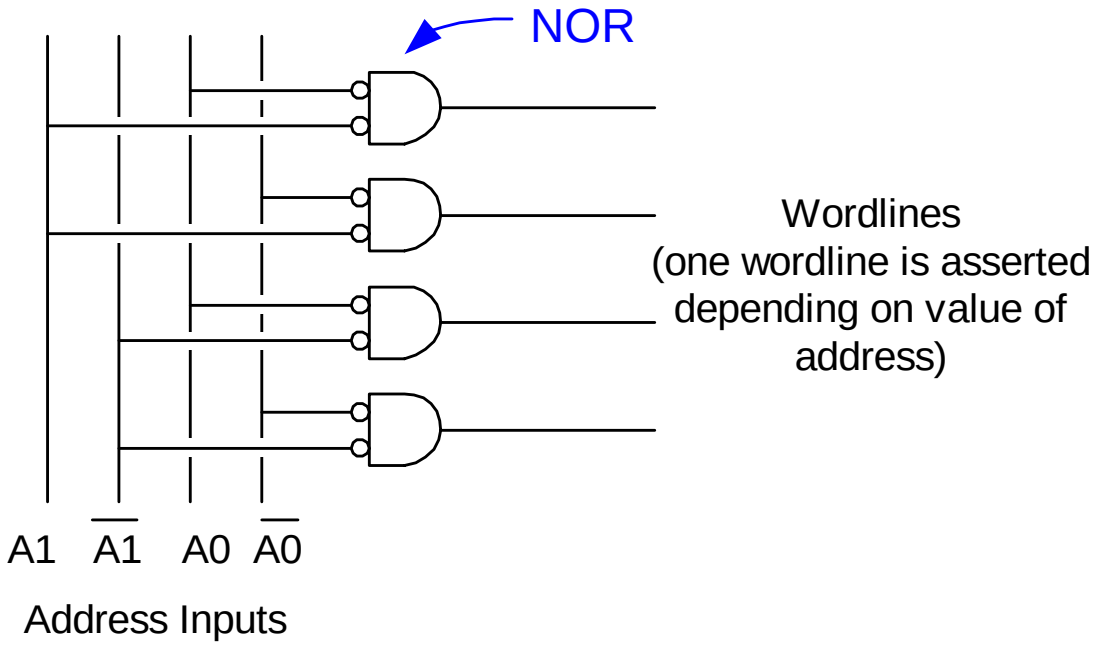


For a n-bit to 2^n decoder, we need to build 2^n, n-input AND gates. And we want to build these AND gates so they layout nicely (in a regular way)

Large Fanin AND Gates

In CMOS building this type of gate causes a problem, since large fanin implies a series stack. We will see a little later in the notes that the best way to do this is to use a two-level decoder by predecoding the inputs.

In nMOS the problem was easy, large fanin NOR gates work well. So a collection of NOR gates solves the problem very nicely.



CMOS Decoders

In CMOS, a large fanin gate implies a series stack. So we need to build a decoder that does not use a large fanin gate. But how? Use a 2-level decoder.

An n-bit decoder requires 2n wires

A0, $\overline{A0}$, A1, $\overline{A1}$, …

Each gate is an n-bit NOR (NAND gate)

Could predecode the inputs

Send $\overline{A0}\ \overline{A1}$, A0 $\overline{A1}$, $\overline{A0}$ A1, A0 A1, $\overline{A2}\ \overline{A3}$ …

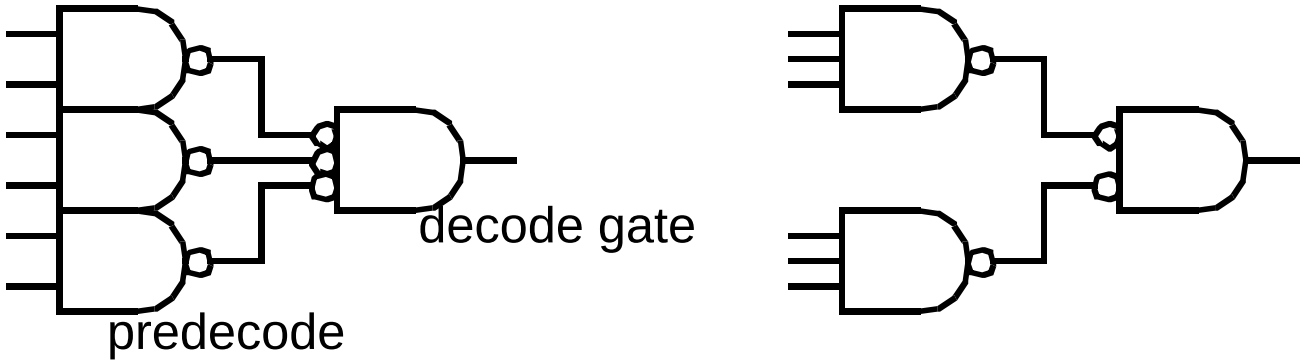
Instead of A0, $\overline{A0}$, A1, $\overline{A1}$, …

Maps 4 wires into 4 wires that need to go to the decoder

Reduces the number of inputs to the decode gate by a factor of two.

Predecode

Predecode is just like what we did when we needed to make a single six input AND gate. Did it in a few levels:



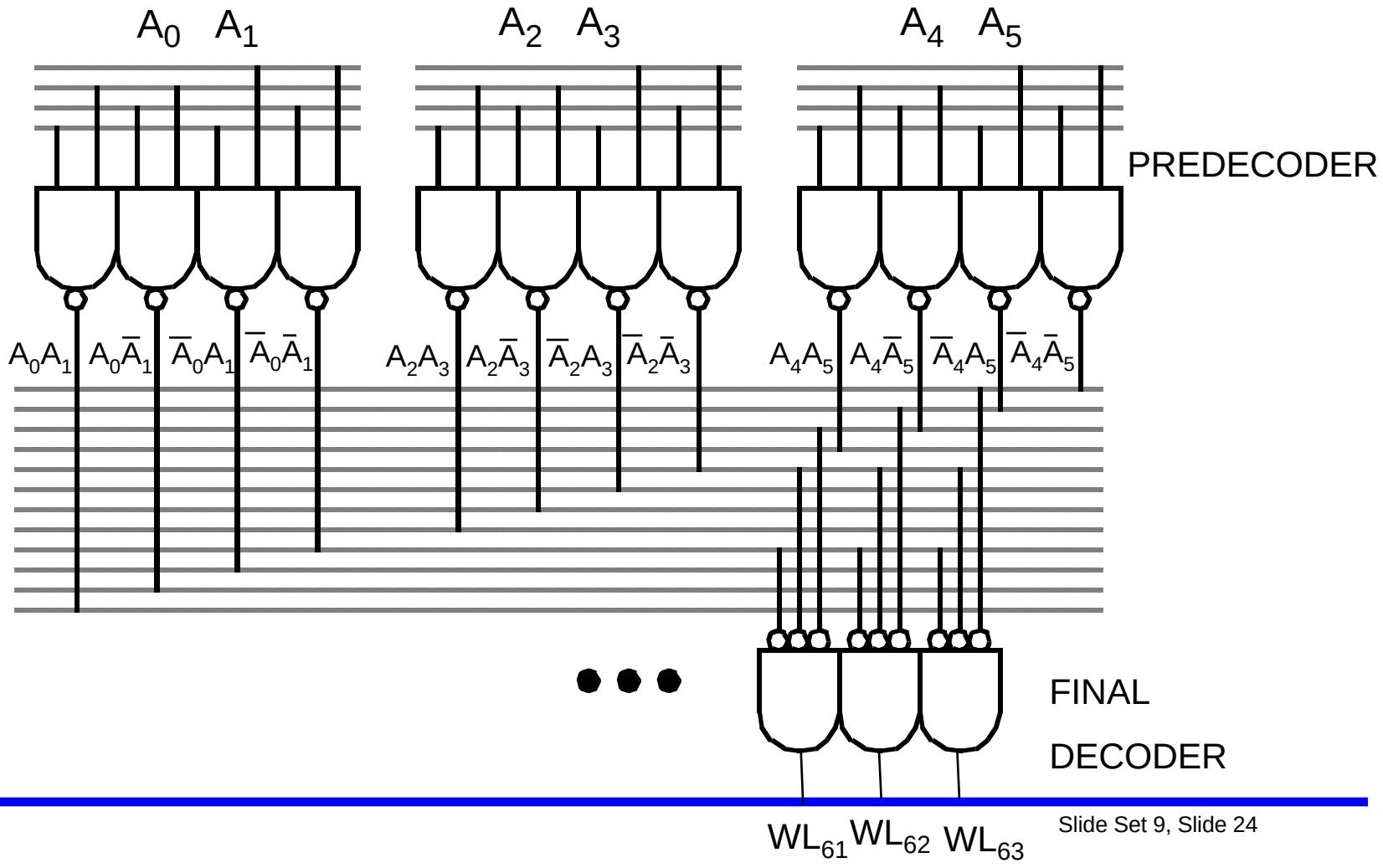
One can do a 2 input predecode, or a 3 input predecode

- A 2 input predecoder generates 4 outputs
- A 3 input predecoder generates 8 outputs

The difference with standard logic is that we need to decode all possible inputs. This means that each predecode gate can be reused by many 'final' decode gates. A little planning can yield a regular layout.

Predecode

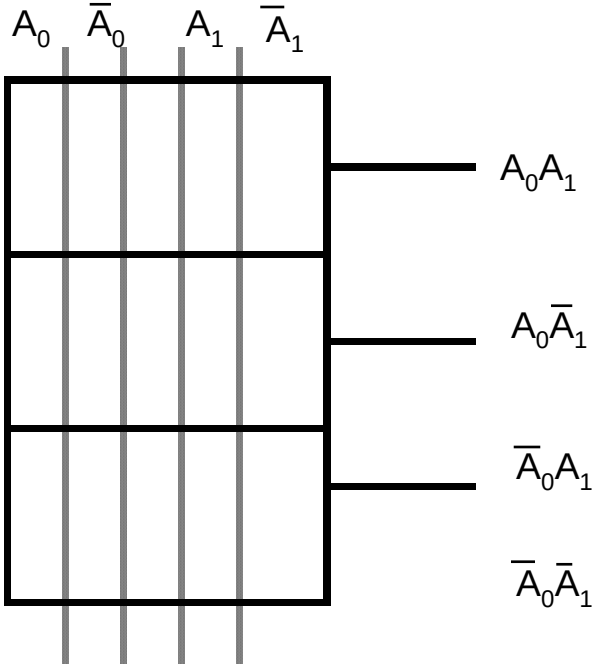
A pre-decoded Decoder:



Decoder Layout Issues

Often we need to build large array structures (for example we need a large RAM), so we want to layout the decoder in as little space as possible. We need to find a good way to layout this structure.

Clearly we need to run the address lines through each decoder cell, and stack the decoder cells on top of each other.



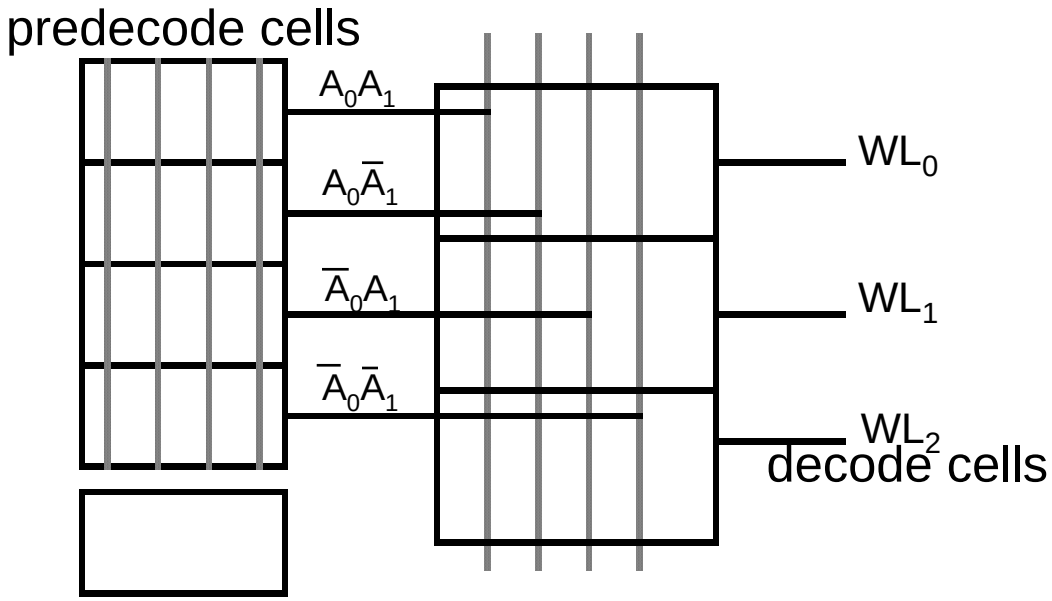
Predecode Layout

The output of the predecode gate need to drive the address lines.

These address lines are usually high capacitance

So usually it is better to use a NAND with an inverter buffer as the predecode cells.

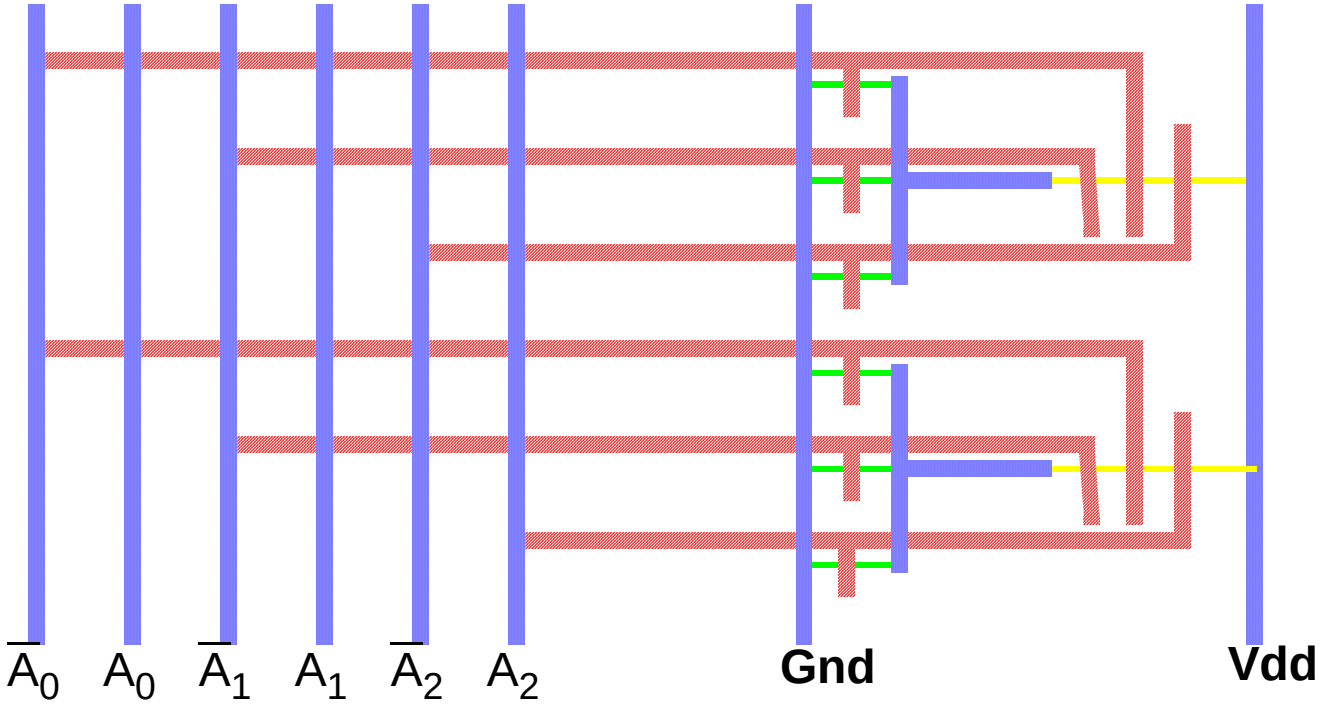
Cells can be placed underneath the address lines, or to the left of the address lines.



Decoder Layout

Cell Area is proportional to n^2. Decoder area is n^3.

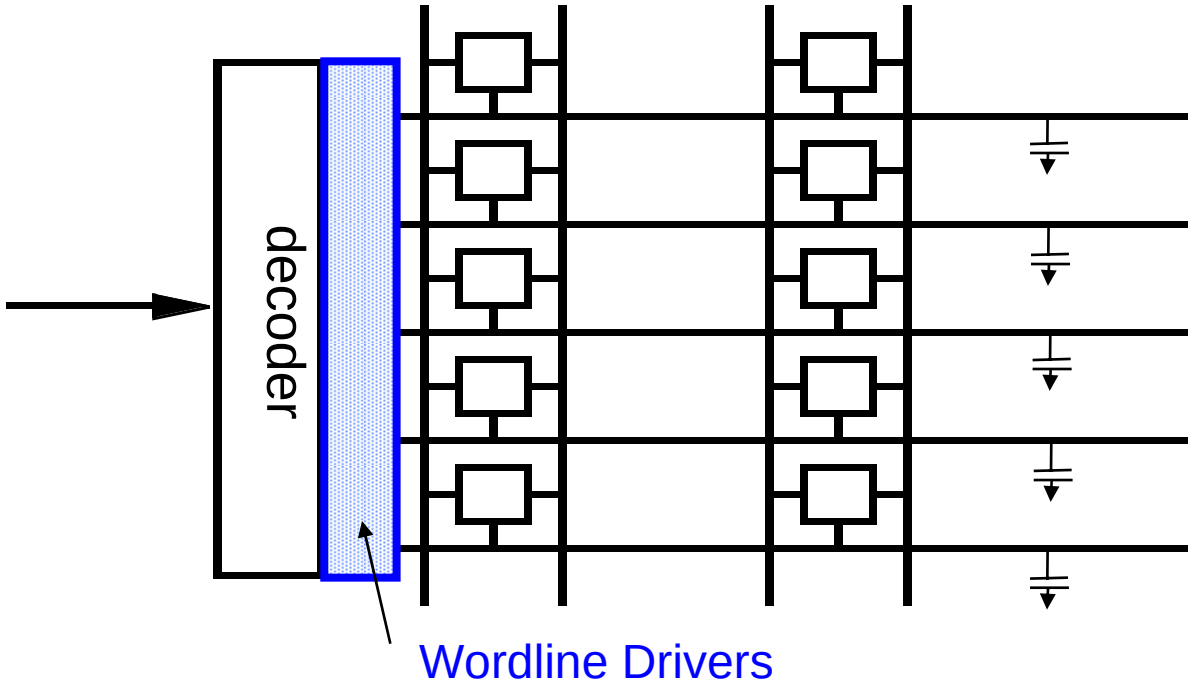
Layout for 3 inputs at a time (pre-decoder)



The problem with this layout is that most of the space is wasted. All of the area under the wires is wasted. We should rotate the gate to fit under the wires.

Wordline Drivers

I lied to you: the outputs of the decoders are not the wordlines. The problem is that the wordlines are high-capacitance, so we need a large driver between the decoder outputs and the wordlines. These drivers are called “wordline drivers”.



Wordline Drivers

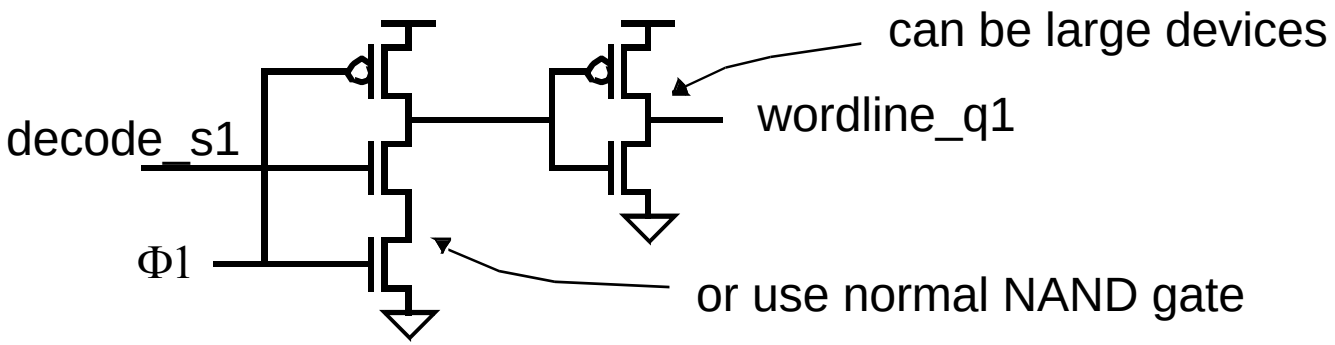
Also need to AND the wordlines with the clock (because we only want the wordlines going high once the bitlines have been precharged).

Clock qualification can be done in the decoder

A0 … An Phi1 - just another input to the decoder

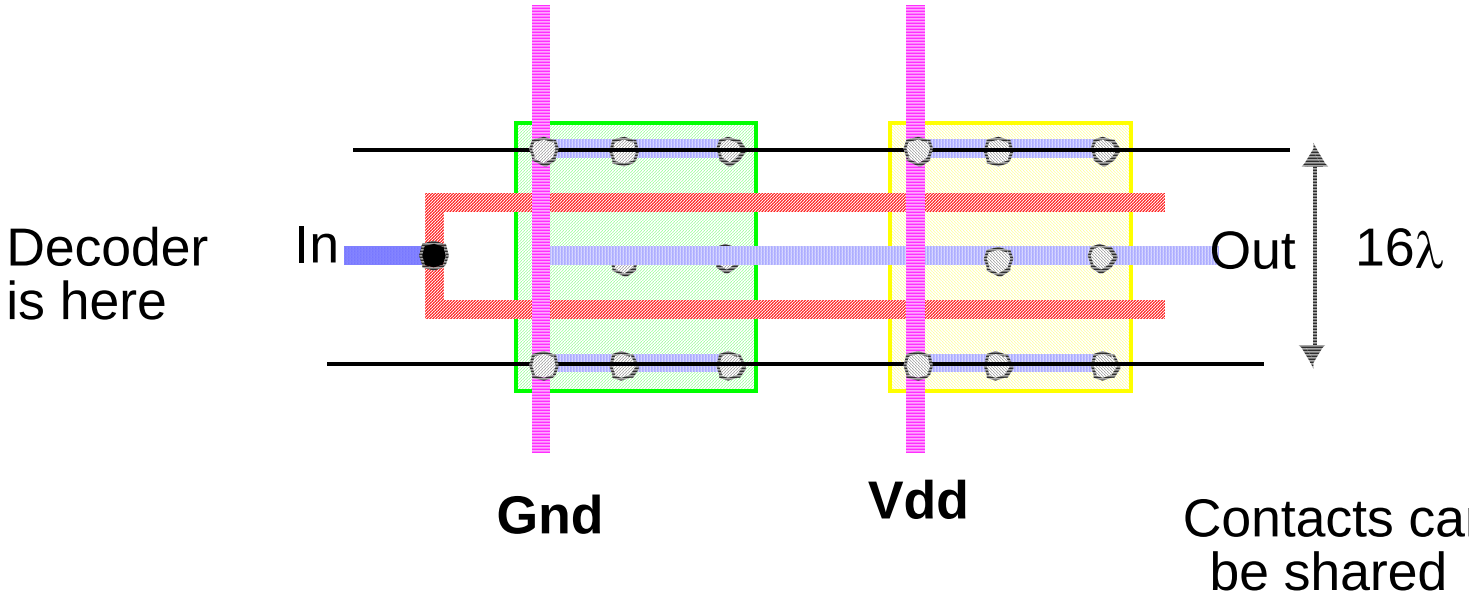
Usually not a great idea, since this can lead to large skew

Clock AND is usually done in last stage before driver



Thin Drivers

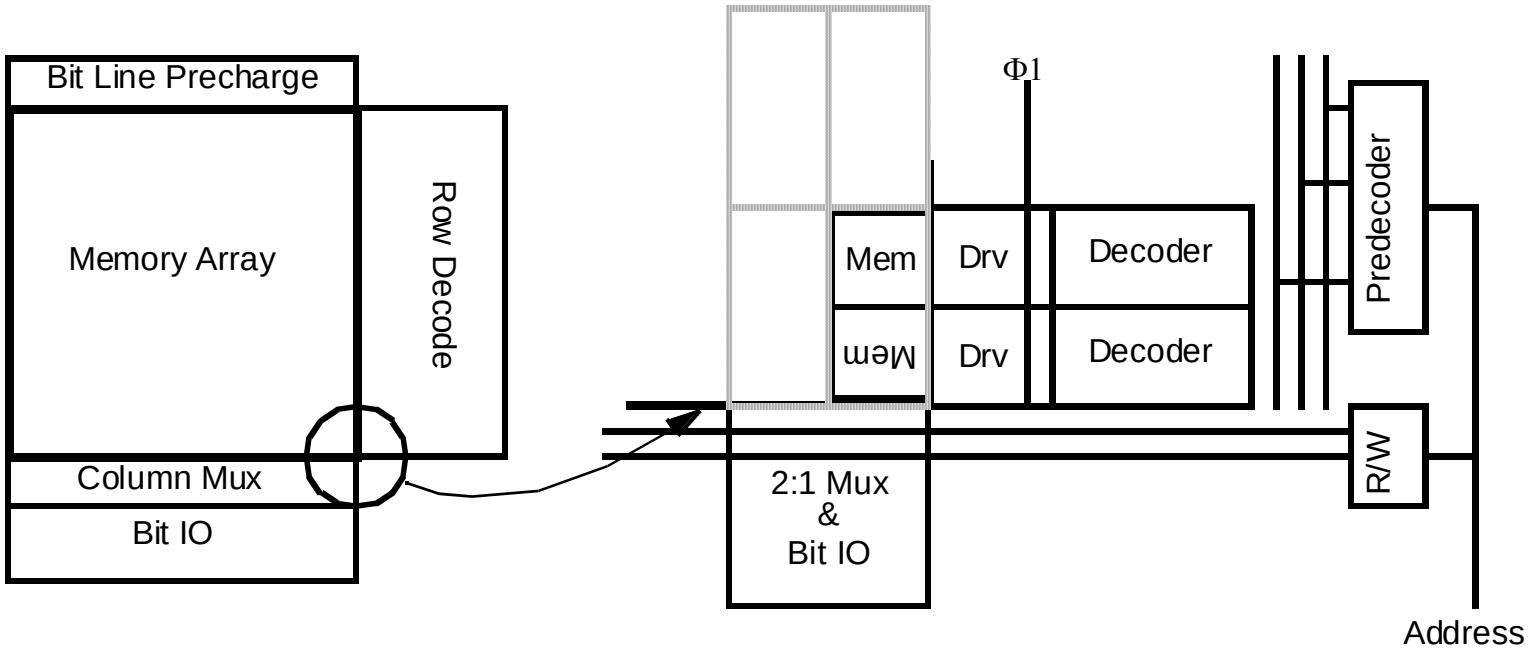
Wordline pitch of memory cell is not that tight (about 40λ), but not that large either. There are some memories (ROMs, dRAMs) with much tighter pitch. For many of these applications you need thin gates and drivers. The minimum useful space is 16λ



For the wordline driver, I might use two of these drivers in parallel, to reduce the horizontal length (effectively fold the transistors again)

Putting it Together

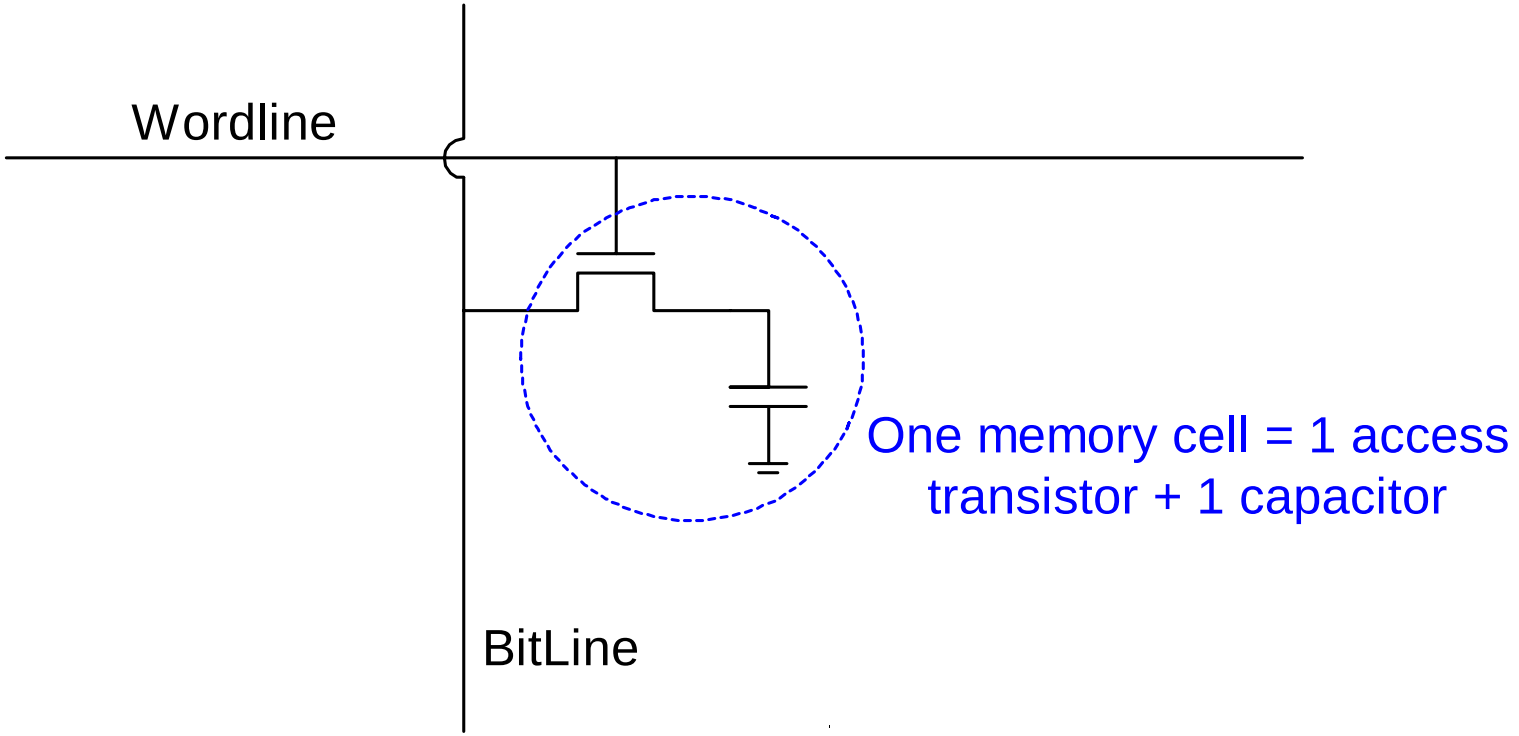
Floorplan for a memory



Often, the memory is generated using a “memory generator”

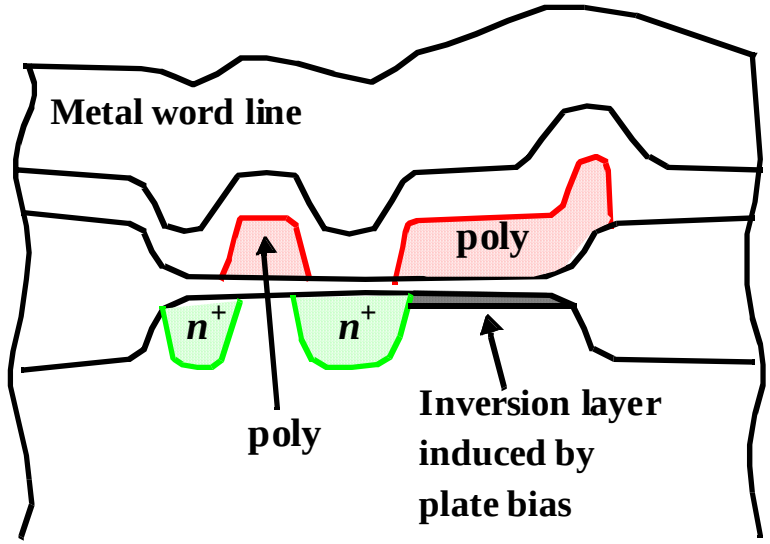
DRAM

DRAM relies on charge stored on a capacitor in each cell:



Simple way to design capacitor:

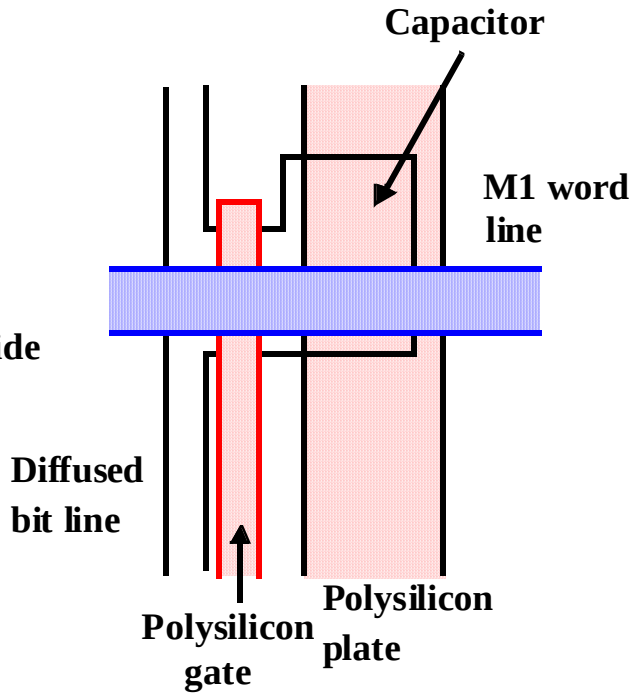
Use an extra “poly-plate” layer



SiO_2

Field Oxide

(a) Cross-section

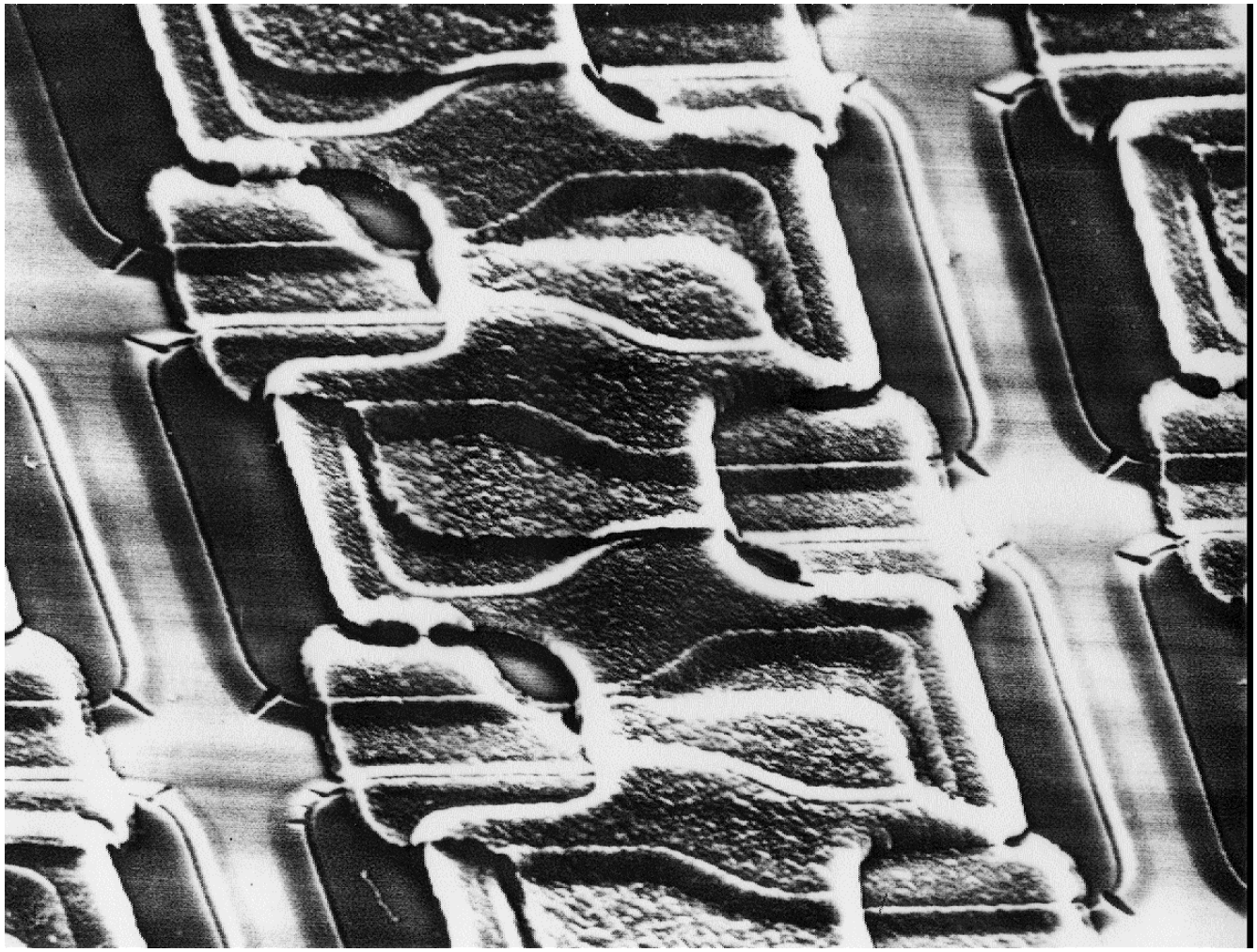


(b) Layout

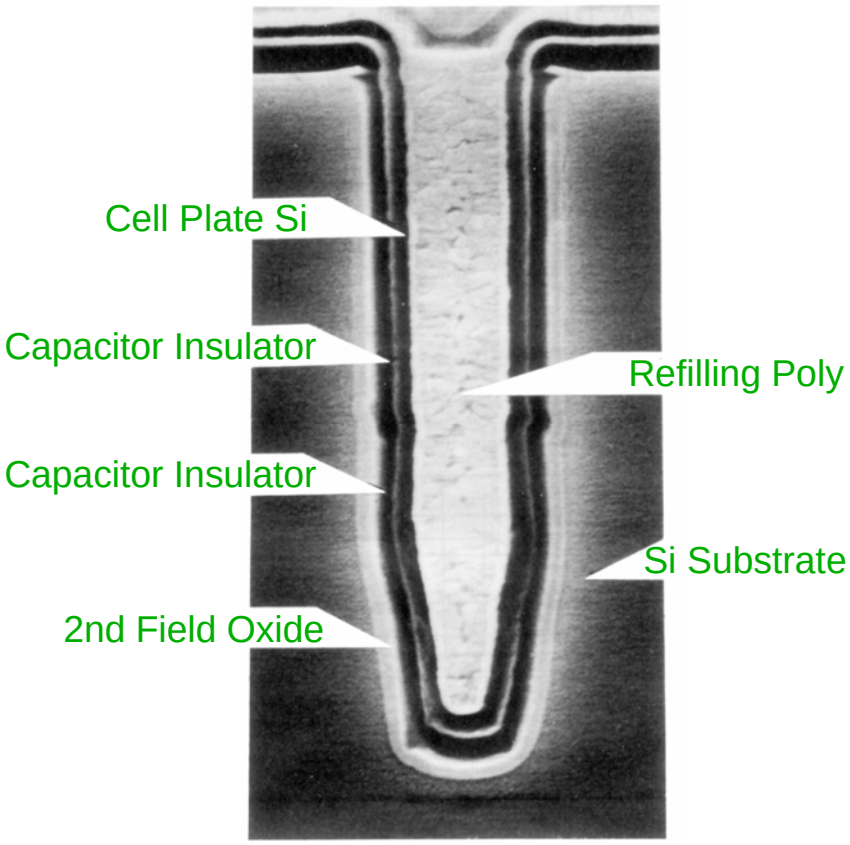
Used Polysilicon-Diffusion Capacitance

Expensive in Area

Microphotograph of DRAM:

- 

Trench Capacitor Cells:



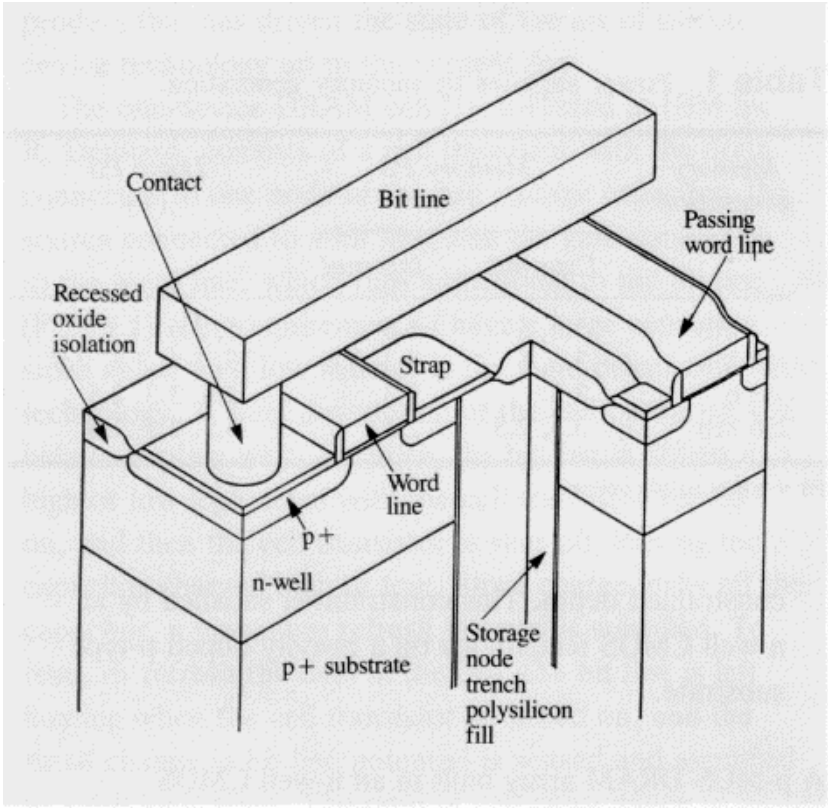
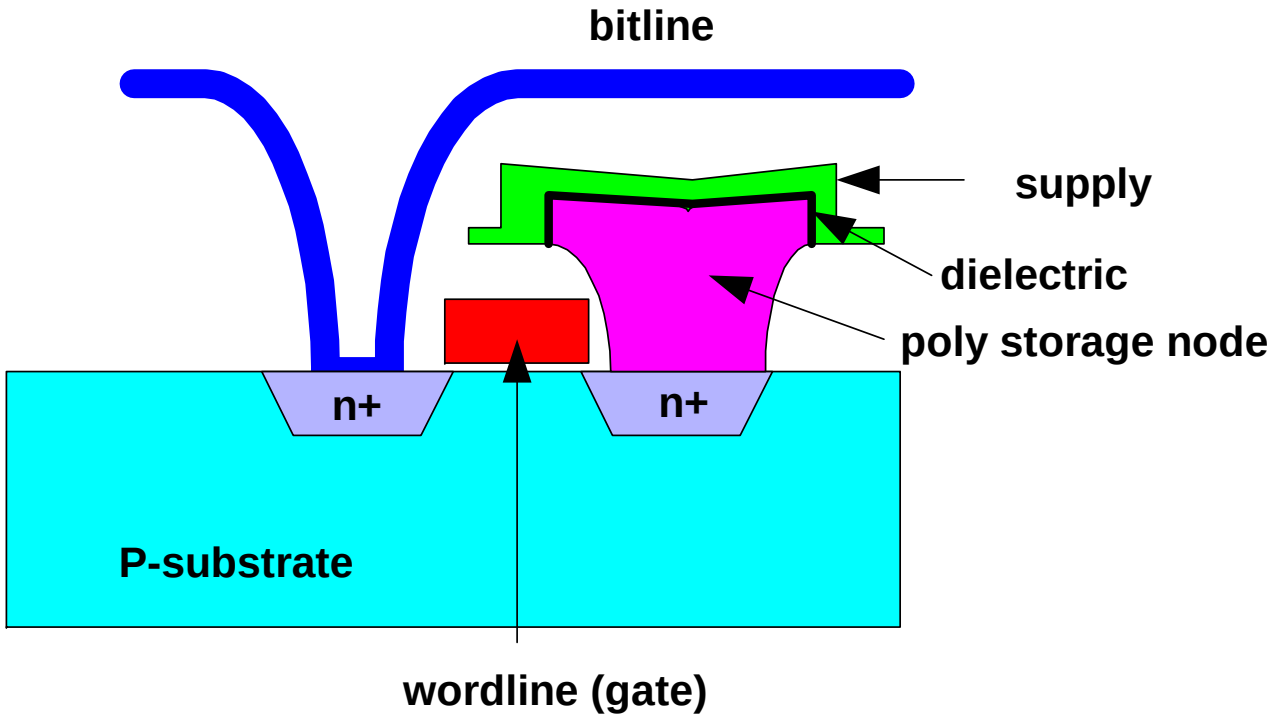


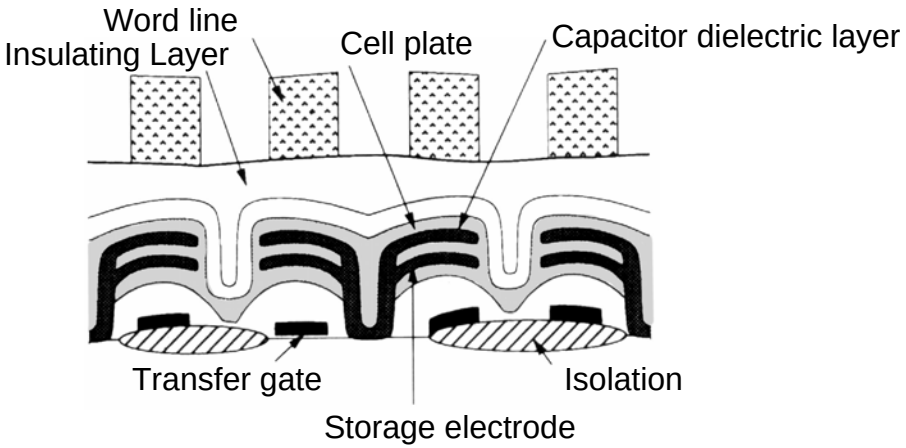
Figure 4

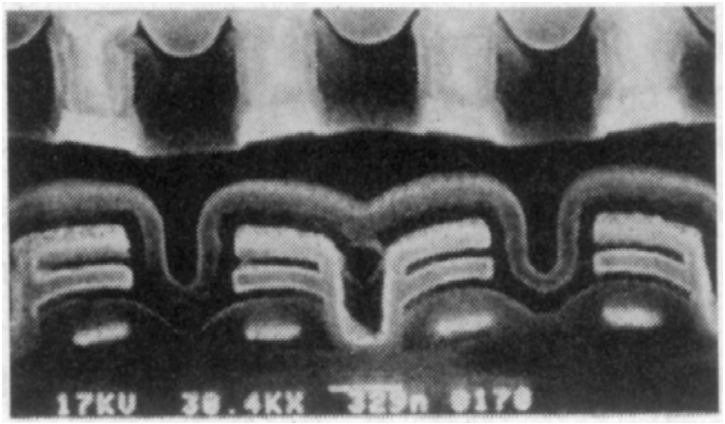
Cross section of 4Mb substrate plate trench (SPT) DRAM cell.

Stacked-Capacitor Cells



Stacked-Capacitor Cells:





And Finally…

- For more on digital integrated circuits and memory design: take EECE481 next term