EENG 4250/ENAS 8750
Introduction to VLSI System Design

Rajit Manohar

Computer Systems Lab

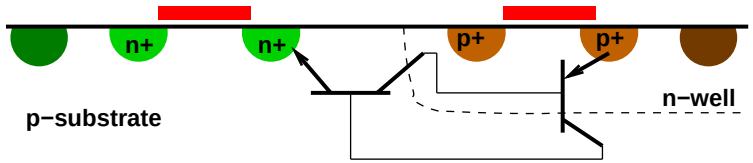
http://csl.yale.edu/~rajit

Fall 2025

Yale

AVLSI

Aside: well contacts/plugs



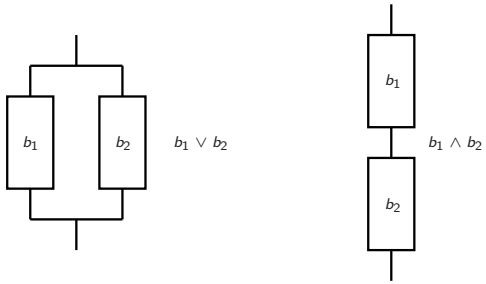
Well contacts prevent this pair from turning on.

Yale

AVLSI

Static CMOS

Gates from Switches:



Dual network

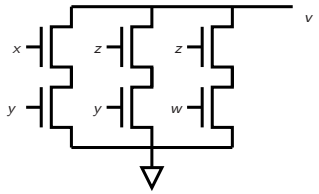
Yale

AVLSI

Static CMOS

Examples:

$(x \wedge y) \vee (y \wedge z) \vee (z \wedge w) \mapsto v\downarrow$



What does the pull-up network look like?

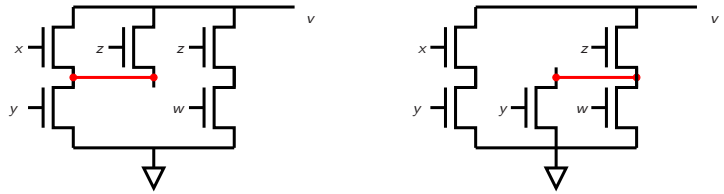
Yale

AVLSI

Gate sharing

Can we share transistors?

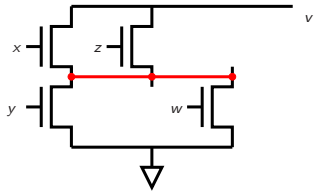
$(x \wedge y) \vee (y \wedge z) \vee (z \wedge w) \mapsto v\downarrow$



Gate sharing

Still more sharing?

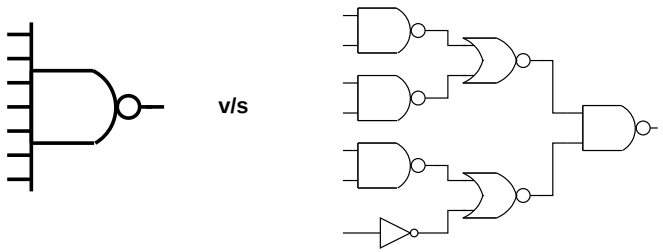
$(x \wedge y) \vee (y \wedge z) \vee (z \wedge w) \mapsto v\downarrow$



How about unrelated outputs?

Multi-stage logic

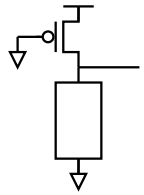
What constitutes a single gate?



When do we go to multiple stages?

Pseudo-NMOS logic

A way to avoid series pFETs:



Tradeoffs?

Better options for this basic idea?

Large scale designs

We know how to build gates.

- Combine switches
  - restoring logic
  - pass transistor logic
- Compute arbitrary Boolean functions
- Represent values with Booleans

How do we organize large designs?

Yale

AVLSI

Concurrency

Nothing "stops" transistors from operating.

$\Rightarrow$ we have a highly concurrent system!

How do we organize the concurrency?

What are some of the features of this concurrent system?

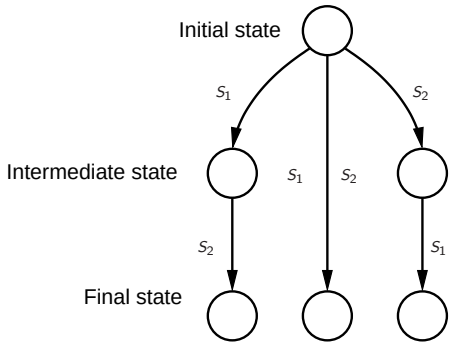
Abstraction: A gate sets its output to *true* or *false* or doesn't modify it. (*recall production rules*)

Yale

AVLSI

Concurrency

If we execute two actions in parallel: $S_1 \parallel S_2$



Yale

AVLSI

Concurrency

What is the net effect of:

1. $x\uparrow \parallel y\uparrow$
2. $x\uparrow \parallel x\downarrow$
3. $x\uparrow \parallel y := x$

(assume x and y are initially *false*)

Yale

AVLSI

Concurrency

1. x becomes *true* and y becomes *true*

Postulate: Concurrent activities of programs that do not share variables do not interfere with each another.

This is an assumption!

2. error!

No write atomicity!

Barrier synchronization

Clocked design: we avoid the problem encountered in 3 by using *barriers*.

Idea: the computation proceeds as follows:

wait;
compute;
wait;
compute;
...

Constraints on reading/writing variables?

Barrier synchronization

Conservative technique:

wait0;
compute0;
wait1;
compute1;
wait0;
compute0;
wait1;
compute1;

Constraints on reading/writing variables?

Implementing barriers

Global signals (*clocks*) used to implement global barrier synchronization.

- First case: one-phase clock
- Second case: two-phase clock

Barrier implementation:

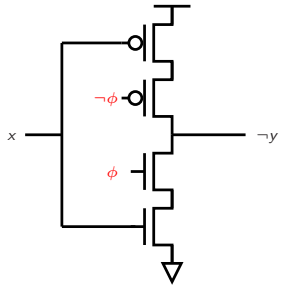
- Wait for clock to be high
- Wait for clock to be low

Implementing barriers

Barrier implementation and Boolean variables:

- Wait for ϕ to be high
- Perform assignment $y := x$

Circuit:



Yale

AVLSI