EENG 426/CPSC 459/ENAS 876
Silicon Compilation

Transistors and Switching Networks

Computer Systems Lab

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

Fall 2018

Yale

AVLSI

Feature size

We are going to be using abstract geometry.

- Dimensions will be in integer units of λ
- Manufacturing rules specified in λ
- Examples:
  - Minimum width of a wire $\geq 3\lambda$
  - Minimum spacing between two wires $\geq 3\lambda$
- $F = .6\mu m$, the *feature size*, determined by the smallest transistor that can be manufactured
- $\lambda = 0.3\mu m$, or $F/2$
  $\Rightarrow$ minimum transistor width is usually 2λ

Yale

AVLSI

Capacitors

- Charge $\propto$ Voltage
- $Q = CV$

Normally,

$$I = C \frac{dV}{dT}$$

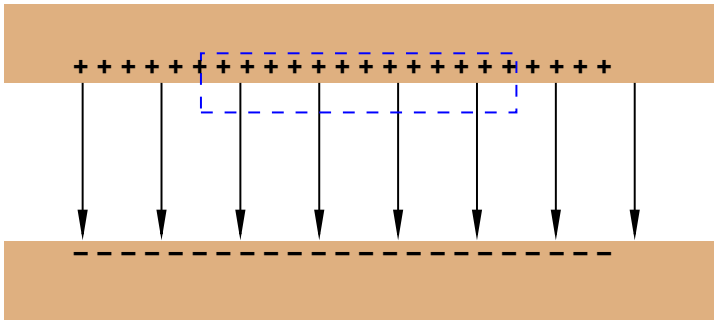
Units: Farads (F)

Order of magnitude in $0.6\mu m$ CMOS: $1fF/\mu m^2$

Yale

AVLSI

Capacitors



Electrostatics: $\nabla \cdot \mathbb{E} = \frac{\rho}{\epsilon_0}$

$$Q = \sigma A \qquad \epsilon_0 = 8.85 \times 10^{-12} F/m$$

So: $E = \frac{\sigma}{\epsilon}$ $\qquad \epsilon = k\epsilon_0$

So: $V = \frac{\sigma d}{\epsilon}$

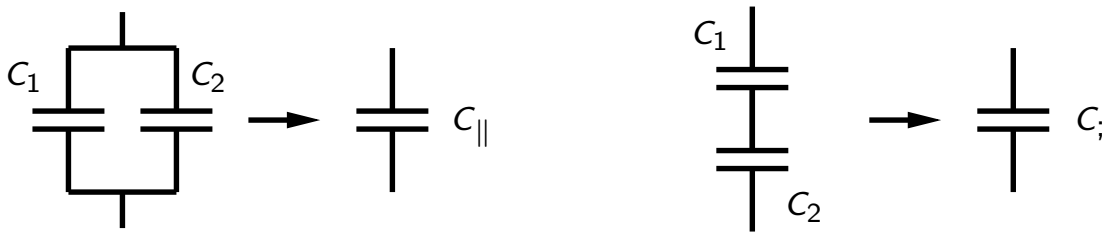
So: $Q = \frac{\epsilon_0 A}{d} V$

Capacitors

Capacitance $C = \epsilon A/d$

When we draw geometry, we control A

So for us $C = (\epsilon/d)A \approx 0.1fF/\square$



Series/parallel combination of capacitors:

- $C_{\|} = C_1 + C_2$
- $\frac{1}{C_;} = \frac{1}{C_1} + \frac{1}{C_2}$

Yale

AVLSI

Resistors

- $I = V/R$, or $I = GV$, or $V = IR$

Units: Ohms (Ω)

Order of magnitude in $0.5\mu m$ CMOS: $0.1\Omega/\square$

Resistance $R = \rho \frac{l}{A}$

$A = w \times t$, t: thickness of wire

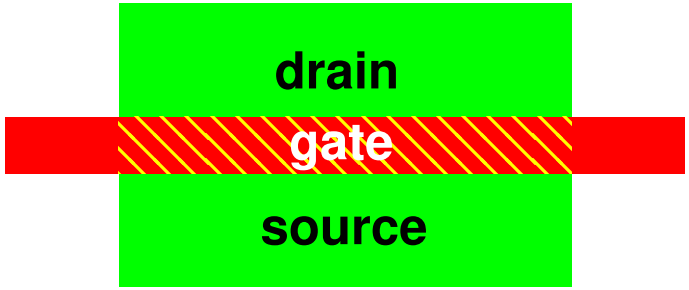
So for us $R = (\rho/t)\frac{l}{w}$

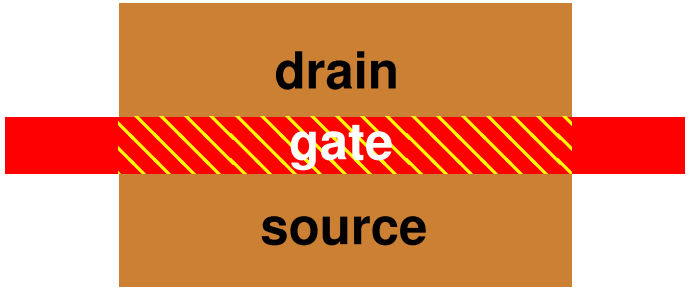
$$\frac{1}{R_{\|}} = \frac{1}{R_1} + \frac{1}{R_2} \qquad R_{;} = R_1 + R_2$$

Yale

AVLSI

Transistors





- Green: "ndiffusion" ("ndiff")
- Brown: "pdiffusion" ("pdiff")
- Red: "polysilicon" ("poly")

The intersection defines the gate region.

- width W, length L

Yale

AVLSI

Simplified transistor equations

$V_{GS} - V_T \leq 0 < V_{DS}$ (subthreshold):

$$I_{DS} \approx 0$$

$0 < V_{GS} - V_T < V_{DS}$ (saturation):

$$I_{DS} = \mu C_{ox} \frac{W}{L} \frac{(V_{GS} - V_T)^2}{2}$$

$0 < V_{DS} < V_{GS} - V_T$ (linear):

$$I_{DS} = \mu C_{ox} \frac{W}{L} [(V_{GS} - V_T) V_{DS} - V_{DS}^2/2]$$

Yale

AVLSI

Digital abstraction

We will simplify all this to:

- If $V(g) > \min(V(s), V(d)) + v_{tn}$, current flows between s and d until $V(s) = V(d)$; otherwise no current flows.
- If $V(g) < \max(V(s), V(d)) + v_{tp}$, current flows between s and d until $V(s) = V(d)$; otherwise no current flows.

Note: $v_{tp} < 0, v_{tn} > 0$ for standard MOSFETs.

Yale

AVLSI

Digital abstraction

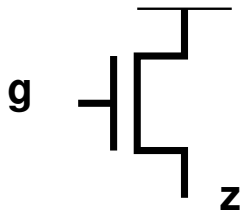
Drawing transistors:

nFET: if $V(g)$ is "high" long enough, then $V(s) = V(d)$

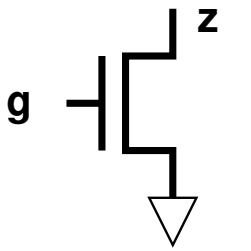
pFET: if $V(g)$ is "low" long enough, then $V(s) = V(d)$

Digital abstraction

Using an nFET:



$g \mapsto z \uparrow$



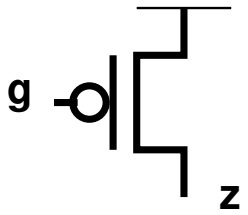
$g \mapsto z \downarrow$

Remember: current flows when

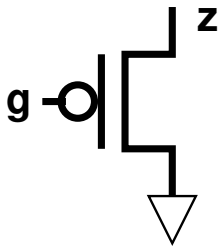
$$V(g) > \min(V(s), V(d)) + v_{tn}$$

Digital abstraction

Using a pFET:



$\neg g \mapsto z \uparrow$



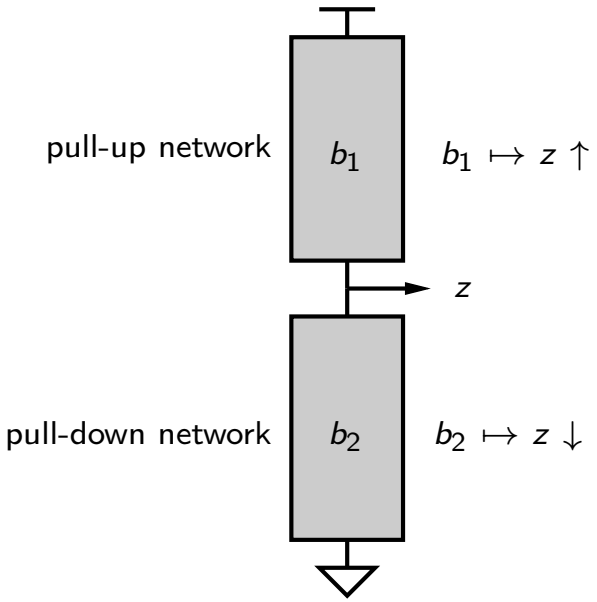
$\neg g \mapsto z \downarrow$

Remember: current flows when

$$V(g) < \max(V(s), V(d)) + v_{tp}$$

Switching networks

In general, we have something like:



We only use pFETs in the pull-up and nFETs in the pull-down.

Yale

AVLSI

Production rules

$G \mapsto S$ is called a *production rule*

- S is $x\uparrow$ or $x\downarrow$
- G is a Boolean expression, called the *guard*

$$x \wedge y \mapsto z\uparrow$$
$$\neg x \mapsto u\downarrow$$

$$\neg x \mapsto u\downarrow$$
$$\neg x \mapsto v\uparrow$$

Yale

AVLSI

Inverting logic

CMOS is said to be "inverting."

Given a one-stage circuit $\mathcal{C}$ computing

$$y = \mathcal{C}(\mathbf{x})$$

where $\mathbf{x}$ is a Boolean vector of inputs, we know that if any component of $\mathbf{x}$ changes from *false* to *true*, the output changes from *true* to *false* or remains unchanged.

Why?

Production rules

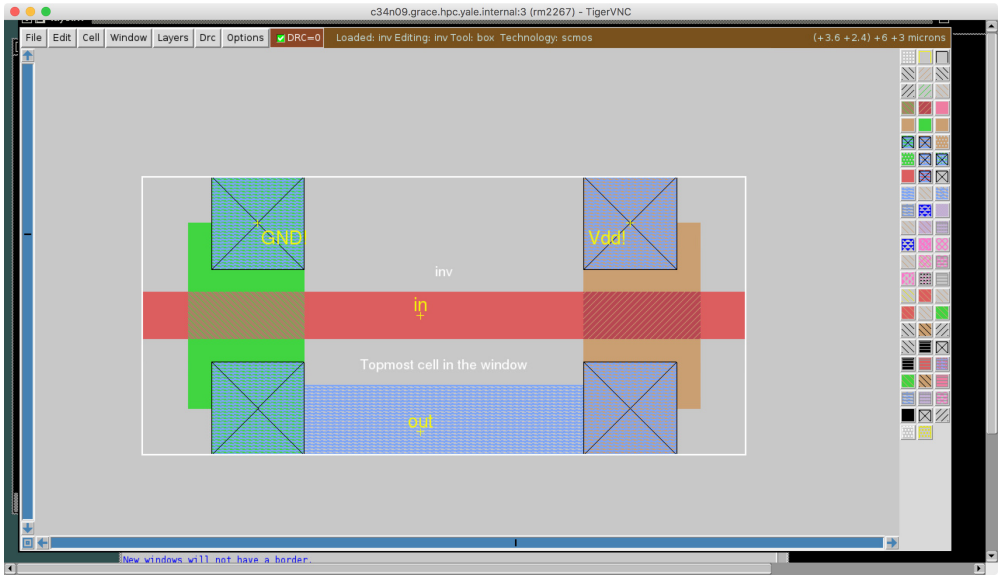
What can we say about a CMOS gate represented by a production rule?

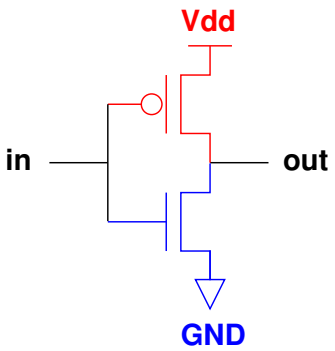
- $x \wedge y \mapsto u\uparrow$
- $x \wedge y \mapsto u\downarrow$
- $\neg(a \vee b) \mapsto c\uparrow$

Yale

AVLSI

Ratioless logic





in	out
0	1
1	0

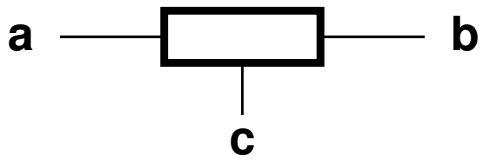
$R_{on} \approx 10^4\,\Omega$

$R_{off} \approx 10^9\,\Omega$

Yale

AVLSI

Switches



Switch is controlled by $V(c)$.

$$c \Rightarrow (a = b)$$

Computing with switches

Use the switches to set nodes to specific voltages.

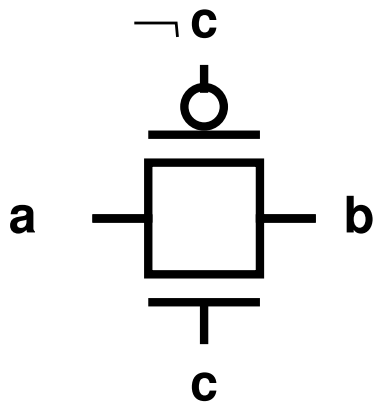
The voltage could come from:

- Power supply
- Another node

Restoring logic switches power supplies.

Pass gate logic switches other nodes.

Switches in CMOS



A restoring inverter using switches?

NAND? NOR?

Pass gates

Pros and Cons:

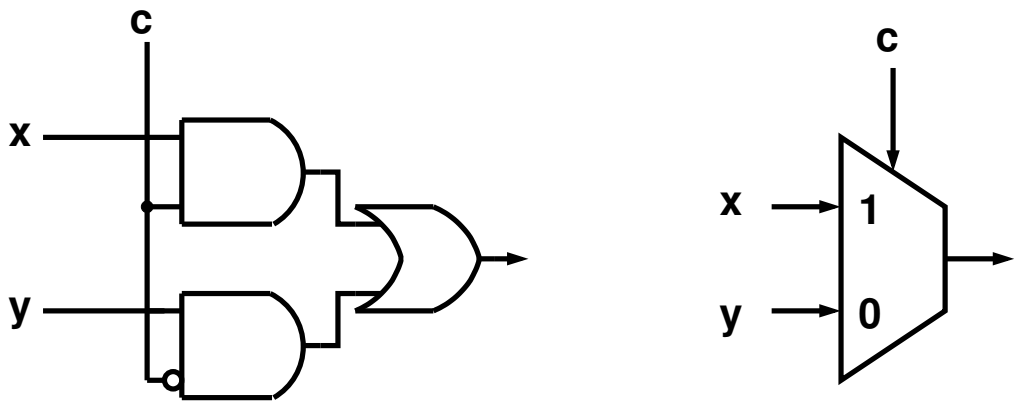
- Can reduce transistor count
- Can reduce logic stages (non-inverting logic)
- Does not restore signals!
- Unintended "sneak" paths

Yale

AVLSI

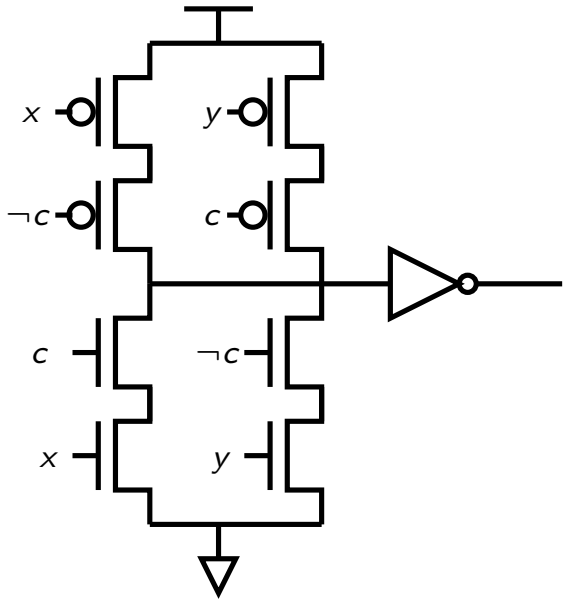
Multiplexer

A 2-input MUX:



Terrible!

Restoring multiplexer

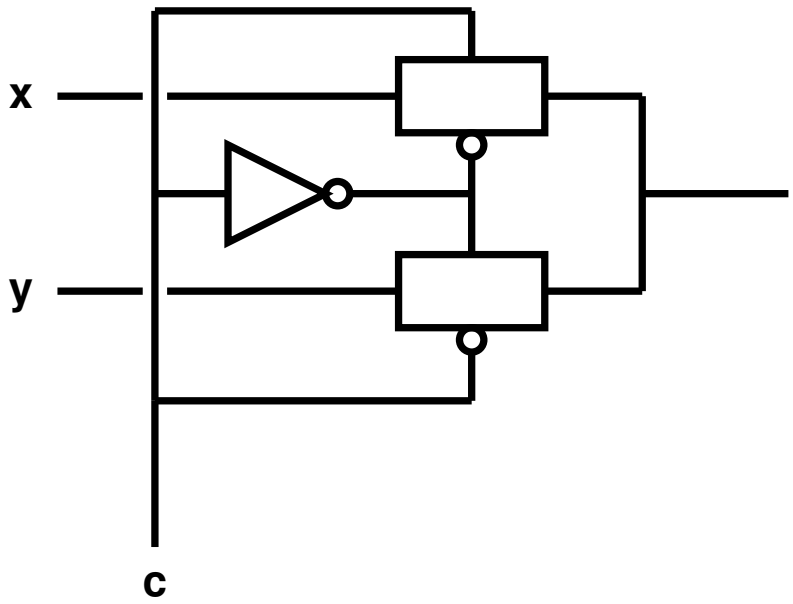


$(\neg x \wedge c) \vee (\neg y \wedge \neg c) \mapsto \_z\uparrow$
$(x \wedge c) \vee (y \wedge \neg c) \mapsto \_z\downarrow$

Yale

AVLSI

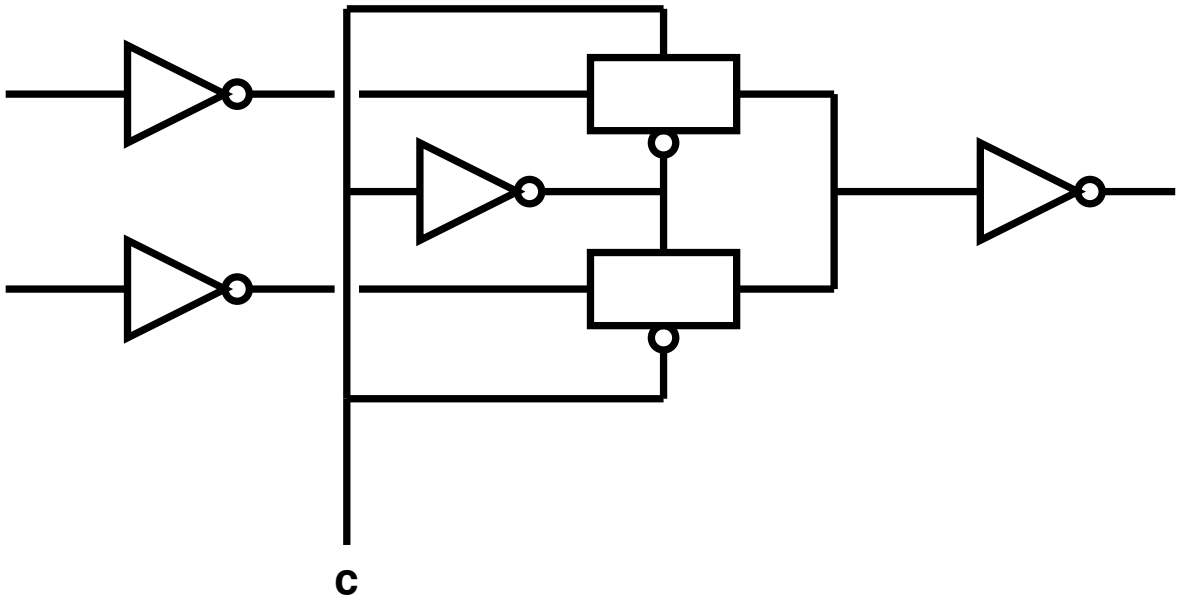
Pass gate MUX



Simple and fast, but it doesn't restore the quality of the logic signals.

Pass gate MUX

To restore the pass gate MUX signals:



Can we simplify this?

Simple functions: static CMOS

If we want to compute the Boolean function $f(\mathbf{x})$:

- Pull-up switching network: $f(\mathbf{x})$
- Pull-down switching network: $\neg f(\mathbf{x})$

The net result: the output is *always* connected to a power supply.

$f(\mathbf{x}) \mapsto z\uparrow$
$\neg f(\mathbf{x}) \mapsto z\downarrow$

Yale

AVLSI

Production rules

A CMOS gate is of the form:

$$B^+ \mapsto z\uparrow$$
$$B^- \mapsto z\downarrow$$

We require: $\neg(B^+ \wedge B^-)$ non-interference

Gates come in two flavors:

- $B^+ \vee B^- \Rightarrow$ the gate is combinational or static
  (This is the same as $B^+ = \neg B^-$)
- $B^+ \neq \neg B^- \Rightarrow$ the gate is state-holding or dynamic

Yale

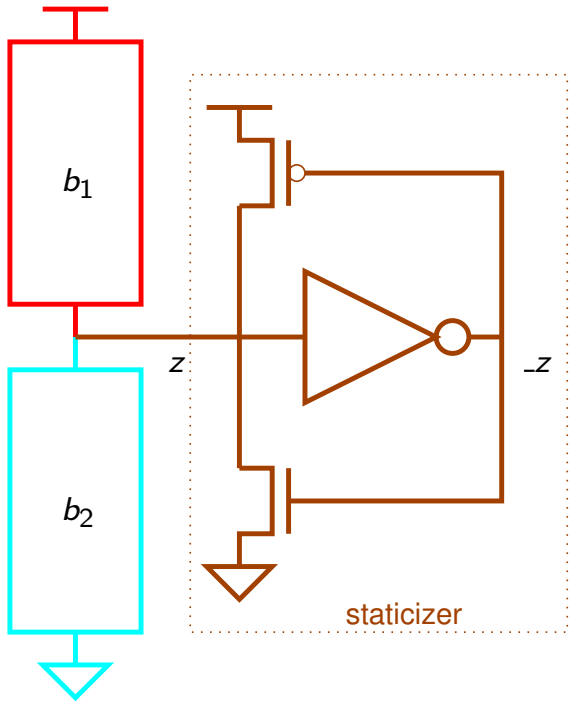
AVLSI

Production rules

$$b_1 \vee z \mapsto z\uparrow$$
$$b_2 \vee \neg z \mapsto z\downarrow$$

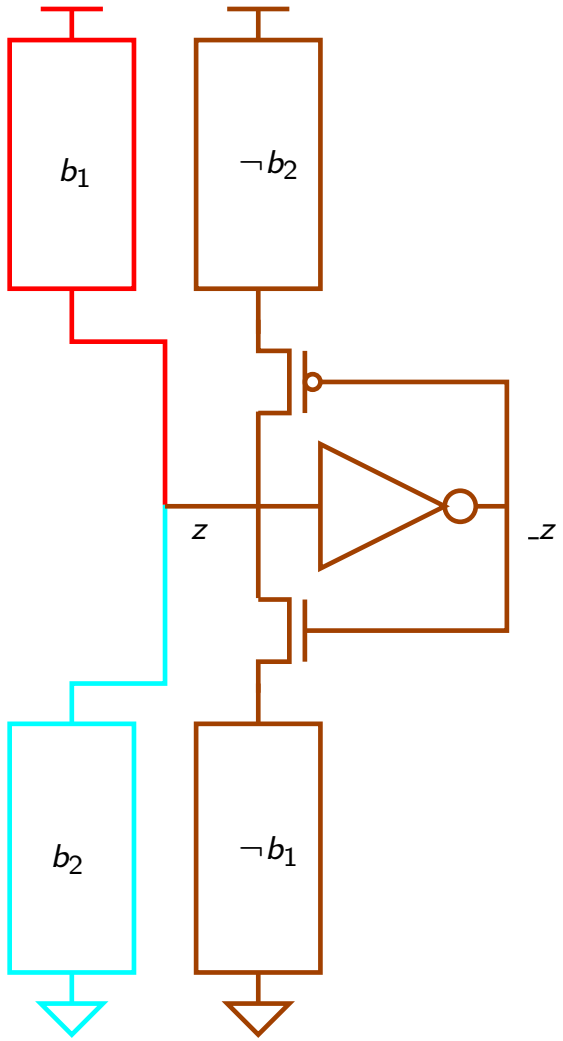
$$b_1 \vee \neg\_z \mapsto z\uparrow$$
$$b_2 \vee \_z \mapsto z\downarrow$$

$$z \mapsto \_z\downarrow$$
$$\neg z \mapsto \_z\uparrow$$



Production rules

$$b_1 \vee \neg b_2 \wedge \neg \_z \mapsto z\uparrow$$
$$b_2 \vee \neg b_1 \wedge \_z \mapsto z\downarrow$$



Yale

AVLSI

Production rules

An *execution* of $G \mapsto S$ is an unbounded sequence of *firings*.

- If G is **true**, the firing amounts to executing S
- If G is **false**, the firing amounts to a **skip**
- If a firing changes the state, it is *effective*; otherwise it is *vacuous*

The execution of a production rule *set*:

- the parallel composition of the individual production rules in the set
- assumption is "weak fairness": a specific production rule will get a chance to fire *eventually*

Yale

AVLSI

Production rules

Stability: an important property of a production rule.

A production rule $G \mapsto x\uparrow$ is *stable* in a computation iff

- G can change from **true** to **false** only if x is **true**
- Alternatively: G can change from **true** to **false** only in states where a firing would be vacuous

(Similar for $G \mapsto x\downarrow$)

What happens in the circuit if a production rule is *unstable?*

Yale

AVLSI

Production rules

If all production rules in a set are stable, then their execution is equivalent to the following:

- Repeat the following steps forever:
  1. Pick one production rule (in a weakly fair fashion) from the production rule set;
  2. Fire the selected rule

Yale

AVLSI