CS/EE 181a 2010/11 Lecture 6

Administrative:

- Projects.

Topics of today's lecture:

- More general timed circuits—precharge logic.
- Charge sharing.
- Application of precharge logic: PLAs
- Application of PLAs: FSMs

Questions about last lecture.

Questions about Lab 3.

Some examples

Copyright ©2010, California Institute of Technology. All Rights Reserved.

Generalized Precharge Logic

Registers are examples of *state-holding* elements.

How are they different from combinational logic?
Future outputs (e.g., register read) depends on *history* of element.
Leakage current (for CMOS) $\Rightarrow$ need to staticize $\Rightarrow$ node is always weakly driven

Basic idea:

Combinational circuit:

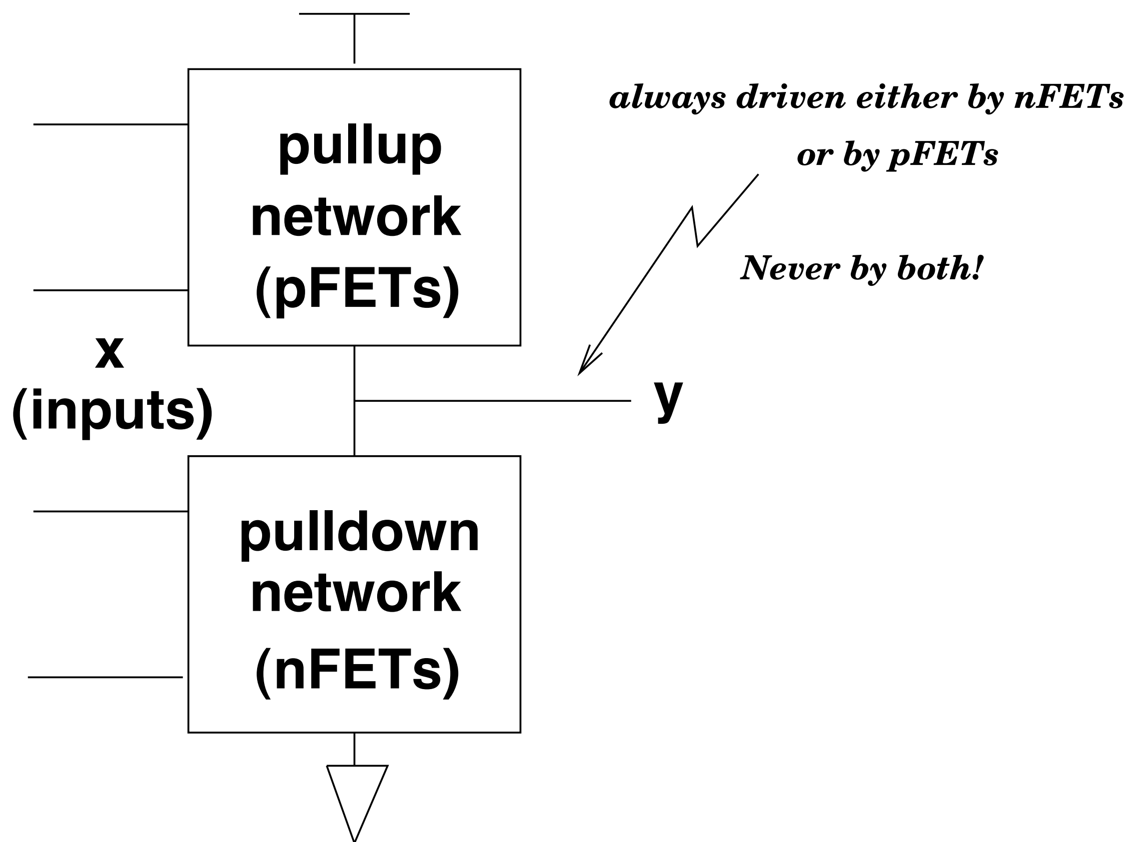
- Pulldown network pulls down when the output should be low.
- Pullup network pulls up *in all other cases.*

(Or the reverse, if you prefer.)

Problem: What if one of the two cases is a lot more difficult than the other?

Copyright ©2010, California Institute of Technology. All Rights Reserved.

Static logic



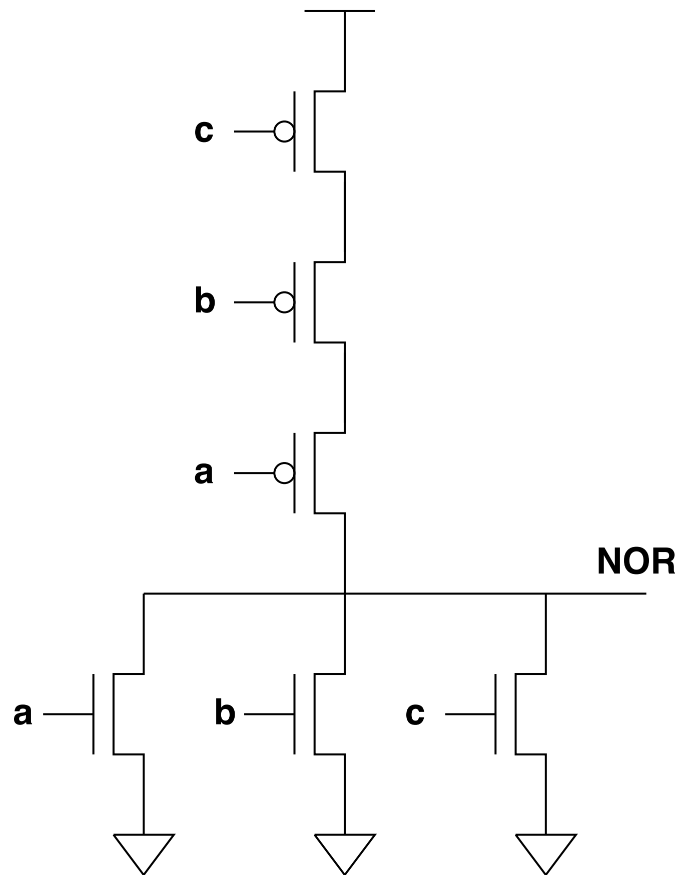
We often have circuits that have

- Large parallel networks in one direction (good!)
- Large series network in the other (bad!)

Copyright ©2010, California Institute of Technology. All Rights Reserved.

Slow combinational logic

Example. Three-input NOR gate:



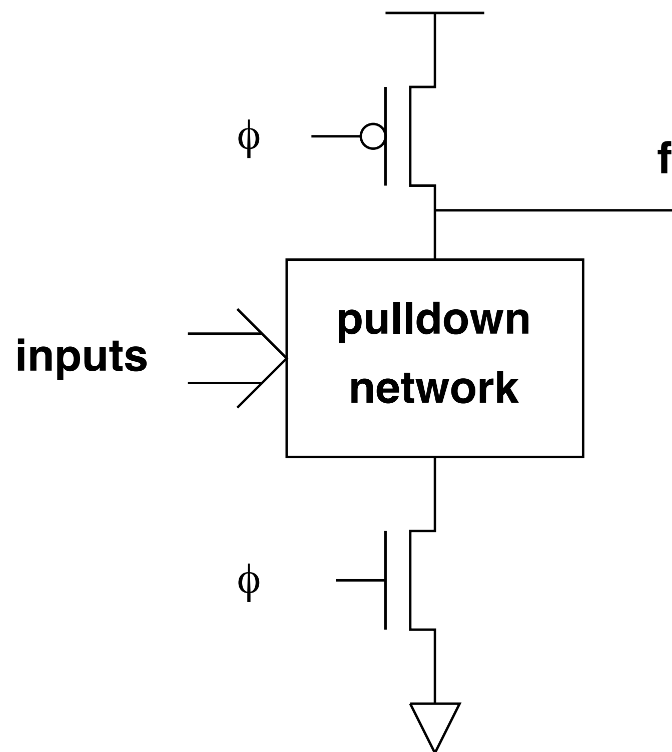
Only one nFET required to pull down output, but three slow pFETs in series!

Can we get rid of the pFETs?
How?

Copyright ©2010, California Institute of Technology. All Rights Reserved.

Approach:

- Pull up output on every clock cycle.
- Only pull it down if necessary.



Output driven high during $\neg\phi$; driven low *or* not at all during ϕ (some clock phase)...

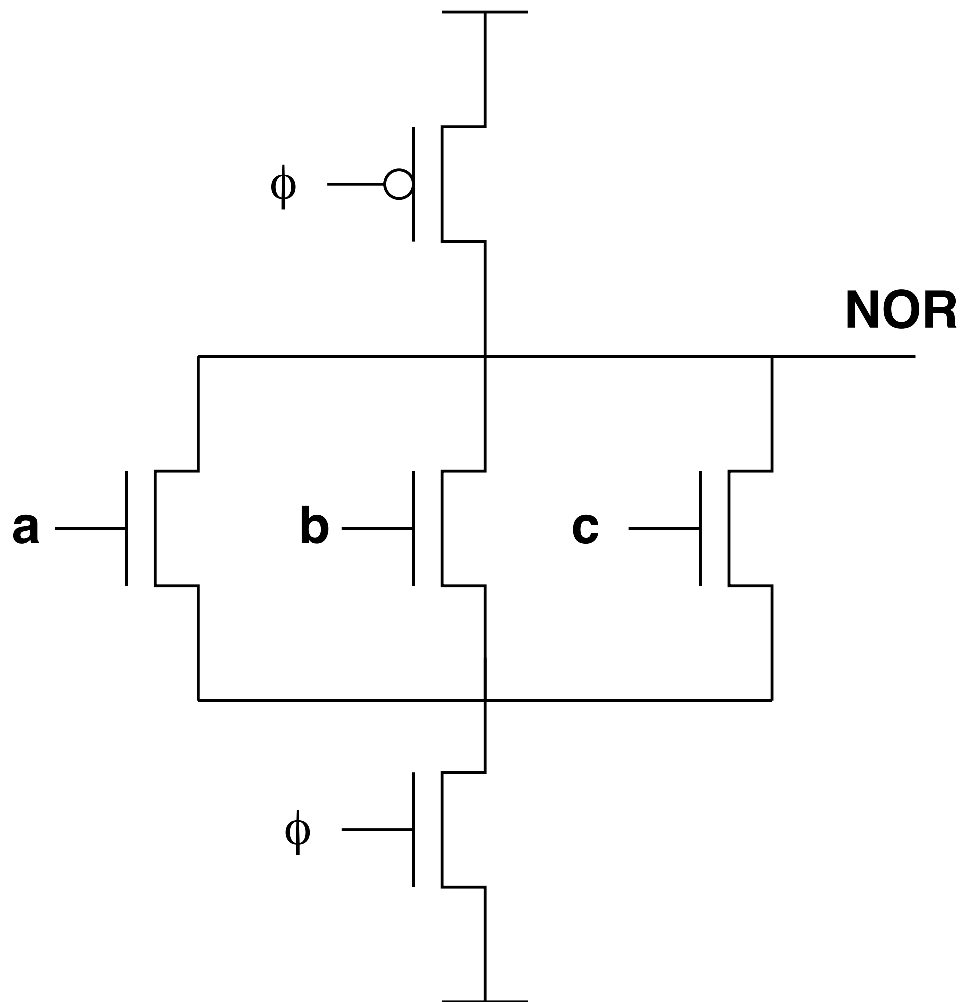
Conditions:

- Output can be state-holding.
- Inputs stable at start of ϕ or...
- Inputs transition upward during ϕ (with time to spare for output to switch)
- Check for charge sharing!

Copyright ©2010, California Institute of Technology. All Rights Reserved.

Precharge example

The NOR gate from before.
If we know that the inputs are stable or transition from low to high during ϕ then we can design the NOR as follows:

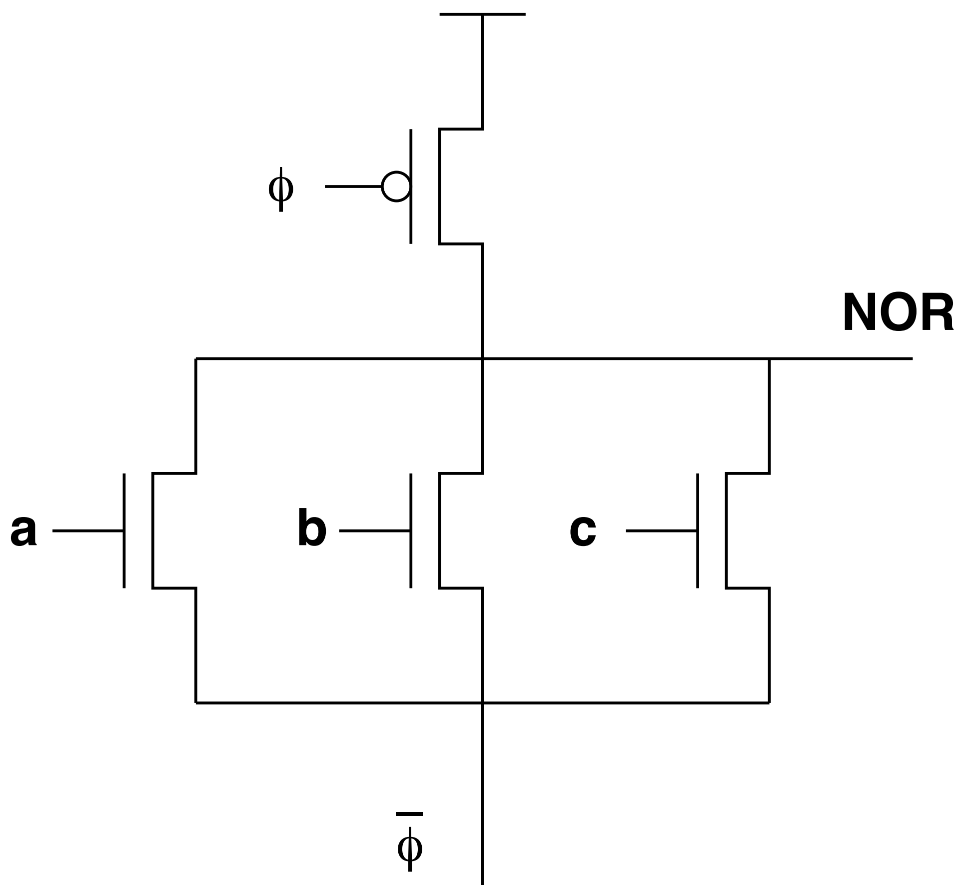


We got rid of the nasty series pFETs!

Copyright ©2010, California Institute of Technology. All Rights Reserved.

The Pass-Gate Transformation

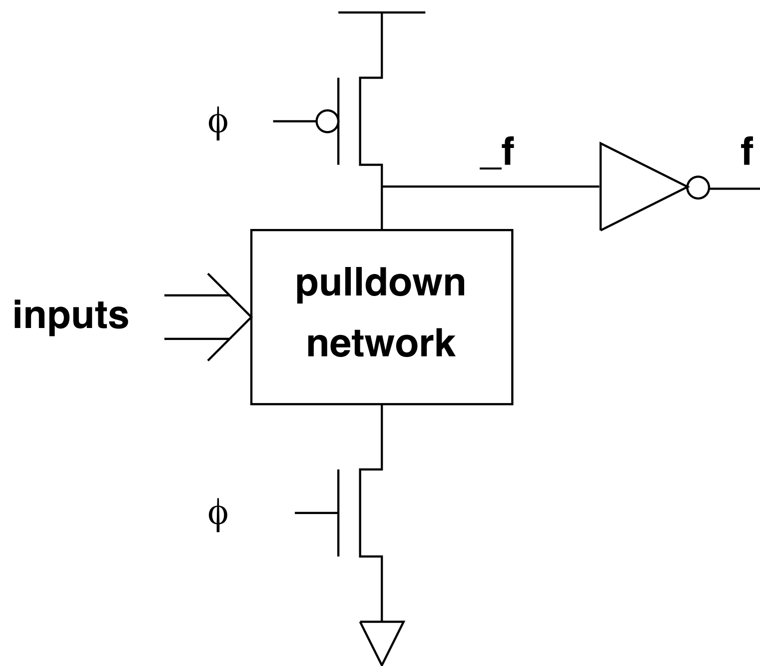
Recall homework 2—the Tricky XOR:
We can do the same pass-gate transformation here since the clock may be treated as a power supply.



Copyright ©2010, California Institute of Technology. All Rights Reserved.

CMOS Domino Logic

Note that given that the inputs are monotonically increasing (e.g., in terms of weight), the outputs are monotonically decreasing.



In a cascaded set of logic blocks each stage evaluates and makes the next one to evaluate ⇒ same way as a line of dominos fall

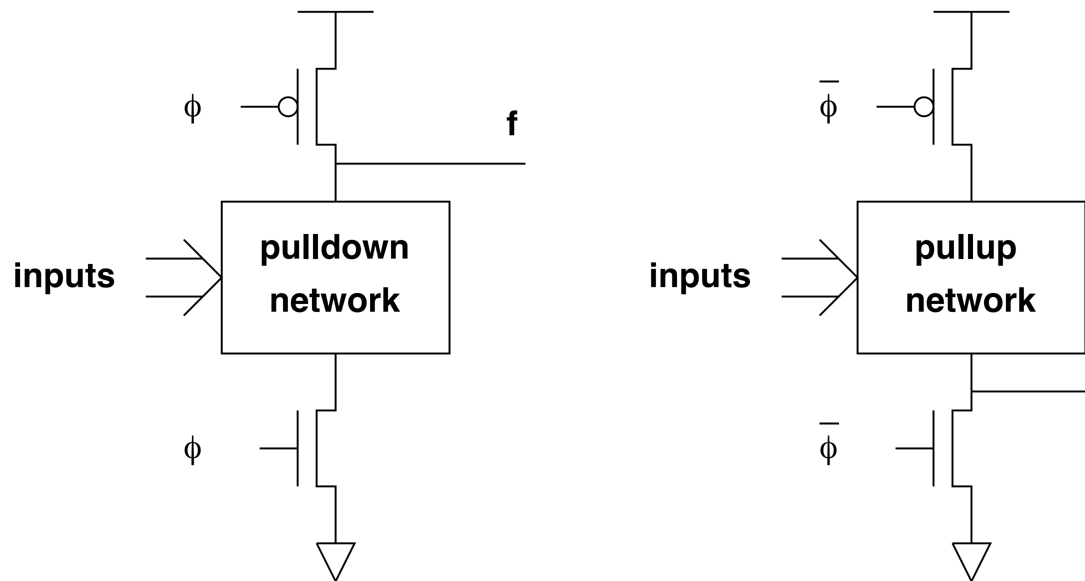
Output is *clean* and amplified.

How many stages should we connect to the same clock phase ?

Copyright ©2010, California Institute of Technology. All Rights Reserved.

NP Domino Logic

What happens when we use pFET precharge logic?



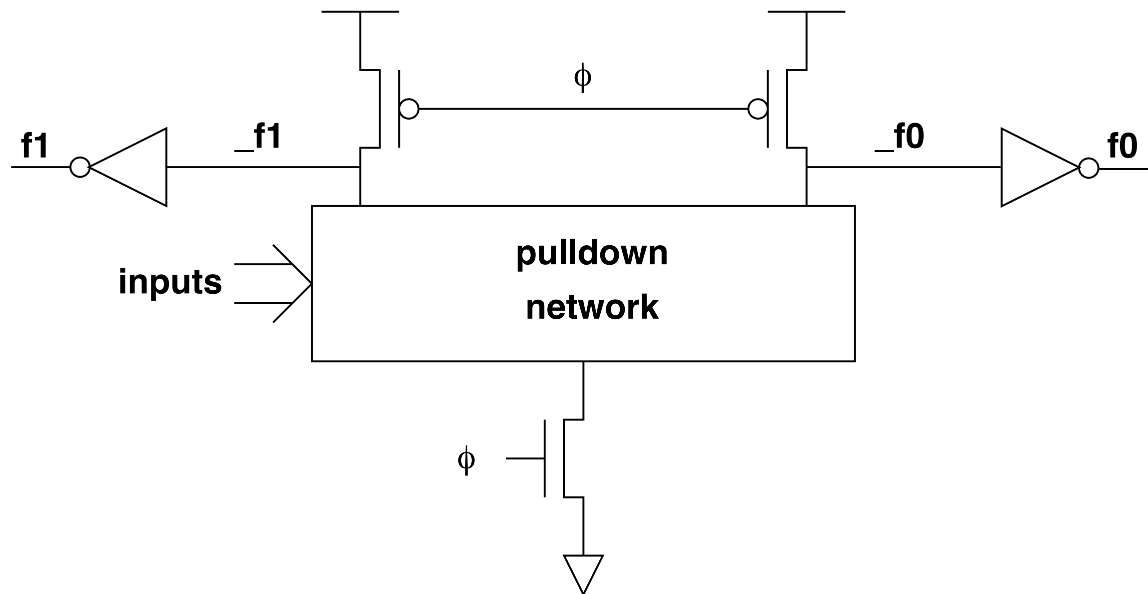
- n-precharge expects inputs going up—produces outputs going down when it computes.
- p-precharge expects inputs going down—produces outputs going up when it computes.

$\Rightarrow$ We can cascade n- and p-precharge blocks!

Copyright ©2010, California Institute of Technology. All Rights Reserved.

Dual-rail Logic (Cascade Voltage Switch Logic)

Same idea as CMOS domino logic but it computes both *true* and *false* values.

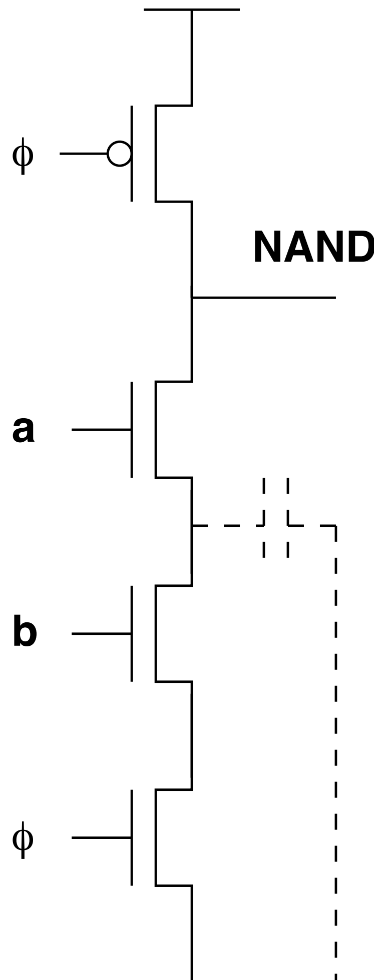


We can share some gates in the pulldown ⇒ less input load ⇒ faster circuit ?

Copyright ©2010, California Institute of Technology. All Rights Reserved.

Charge sharing

Example: Precharge NAND:

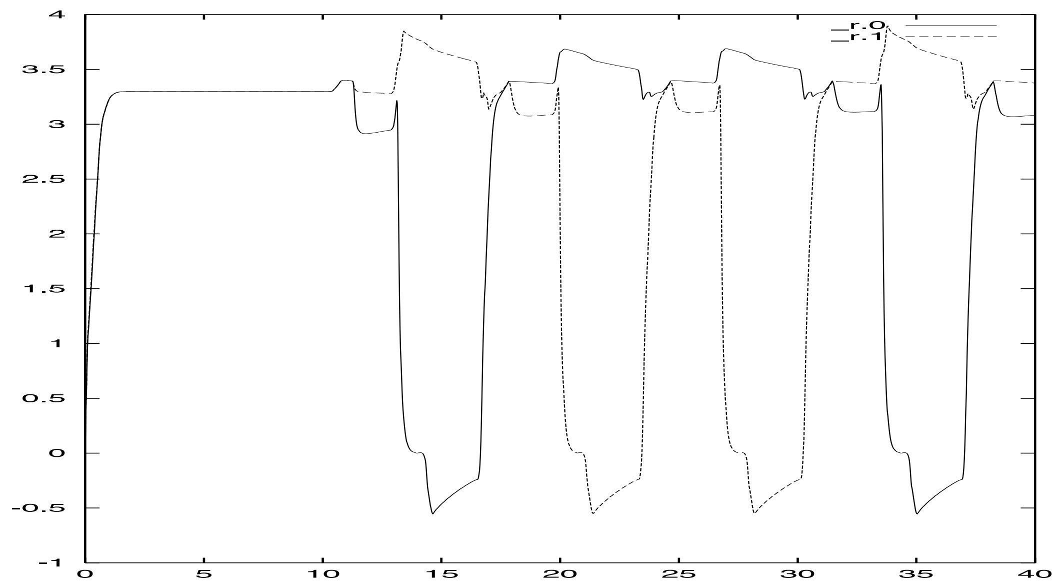


This circuit may exhibit "static charge sharing." (Especially if the output is a small capacitor.)
Same problem for pass gates.

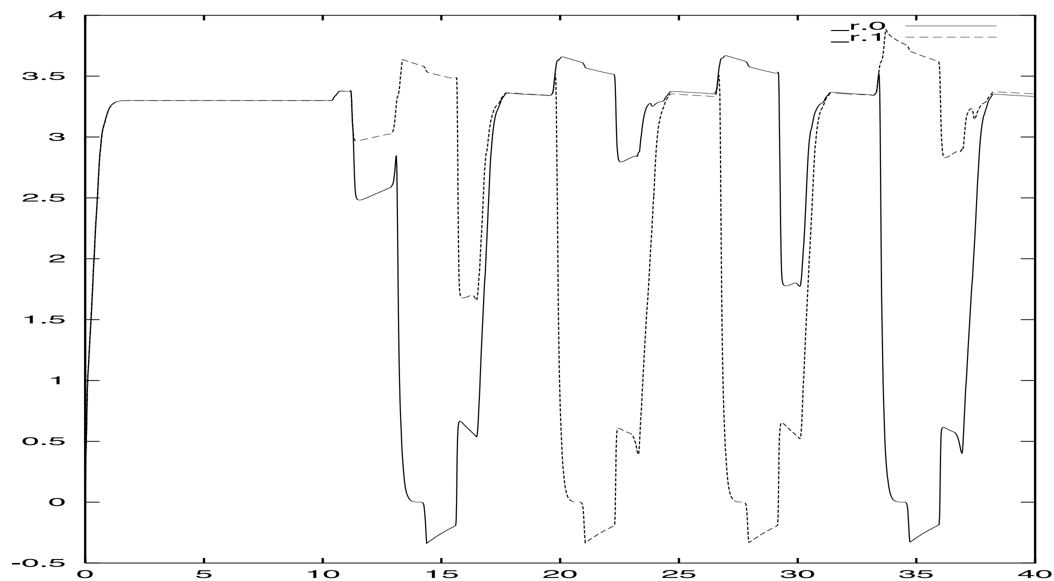
Copyright ©2010, California Institute of Technology. All Rights Reserved.

Charge sharing in SPICE

Normal signal behavior ...



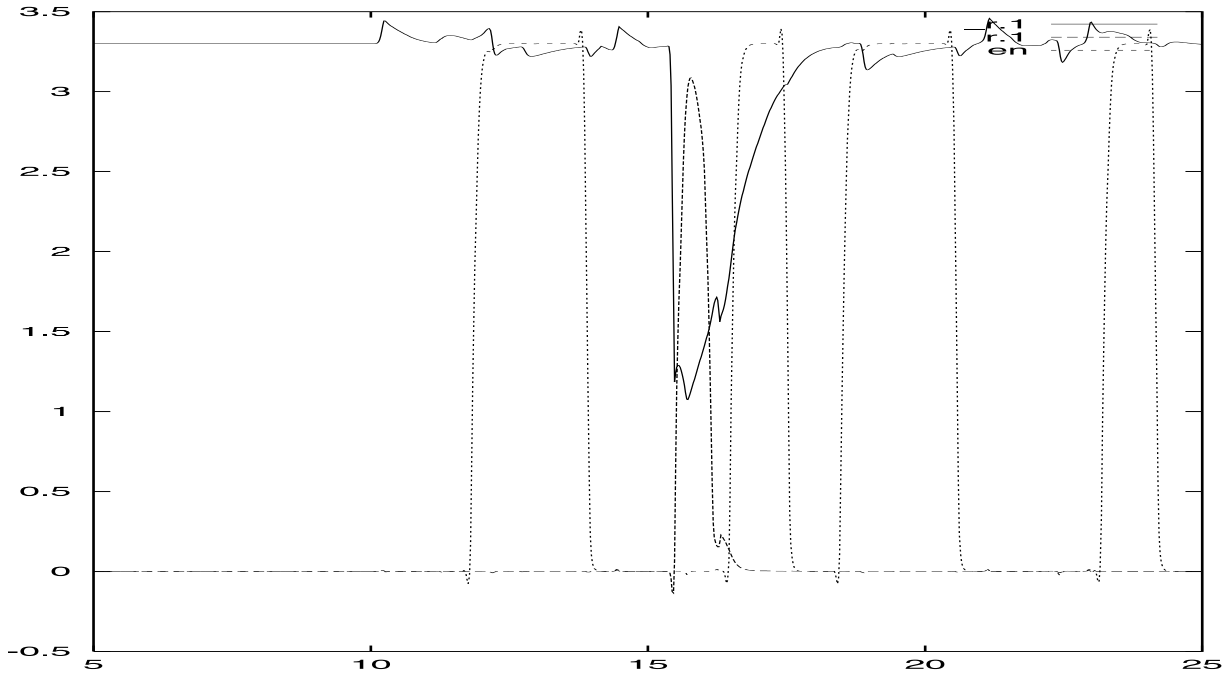
Charge sharing ...



Copyright ©2010, California Institute of Technology. All Rights Reserved.

How bad does it get ?

Fatal charge sharing ...



Copyright ©2010, California Institute of Technology. All Rights Reserved.

Solutions to Charge Sharing

Output node: $V0$, $C0$ Internal node: $V1$, $C1$.
Assume $V1$ small.
Final output node voltage: $V = \frac{C0 \times V0}{(C0+C1)}$.
We want to minimize $V - V0$, i.e., $\frac{C1 \times (V1 - V0)}{(C0+C1)}$

- Reduce the capacitance ratio between internal nodes and output:
  - increase output load
  - reduce internal nodes (reduce sizes & sharing)
- Reduce the voltage difference between internal node and output:
  - precharge the internal node ...
  - switch gate ordering

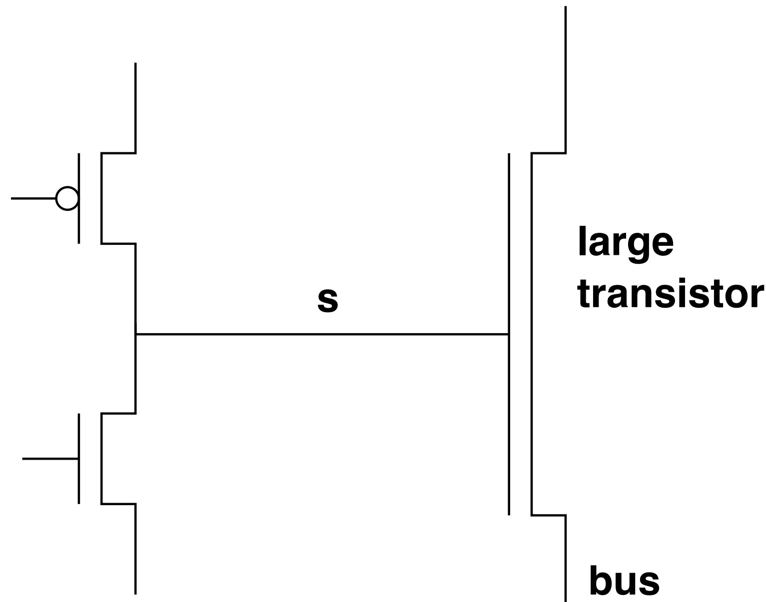
How about combinational logic vs state-holding logic in terms of charge sharing ? Staticizers ?

We will deal more with this when we do analog simulations.

Copyright ©2010, California Institute of Technology. All Rights Reserved.

Dynamic Charge Sharing

Another kind of charge sharing: Coupling from transistor source/drain to gate.



- Can cause problems for state-holding node s if bus switches.

  (Usually only happens with very big transistors, e.g., bus drivers.)

- On modern chips with dynamic logic: happens with very long wires.

Copyright ©2010, California Institute of Technology. All Rights Reserved.

An Application of Precharged Logic

In Lecture 4:

$$f(a, b, c) = ab + bc + ac \ldots$$

How do we implement this efficiently?

- Use a regular structure that can be generated by machine.

If we have large terms, we want to avoid the long series chains of transistors.

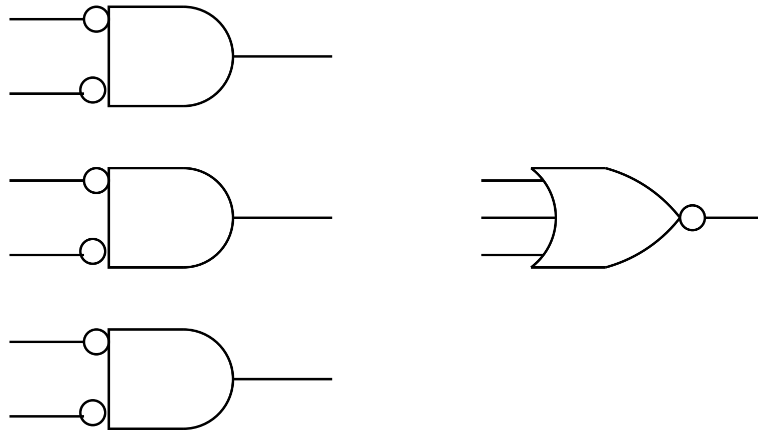
- Use precharge NORs!

$\Rightarrow$ PLAs.

Copyright ©2010, California Institute of Technology. All Rights Reserved.

PLAs

To implement $f(a,b,c) = ab + bc + ac$:
To start with:



And then we get a bit more tricky.

$$\neg f(a,b,c) = (\neg a \vee \neg b) \vee (\neg b \vee \neg c) \vee (\neg c \vee \neg a)$$

$$f(a,b,c) = \neg(\neg a \vee \neg b) \vee \neg(\neg b \vee \neg c) \vee \neg(\neg c \vee \neg a)$$

Only nor's...Precharge nor's most efficient: at most 2 NFETs inseries.

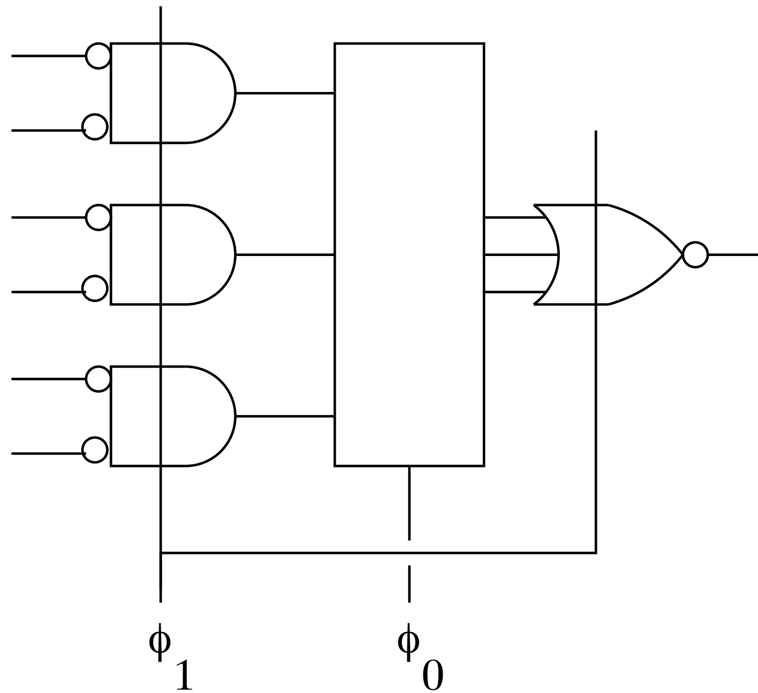
Copyright ©2010, California Institute of Technology. All Rights Reserved.

PLAs compute several functions at once! Very dense!
The IBM Cell processor contains 27 dynamic PLAs in each core for control signals.
Only way to meet timing requirements!

Copyright ©2010, California Institute of Technology. All Rights Reserved.

The Implementation

The PLAs we'll be using are implemented as follows:

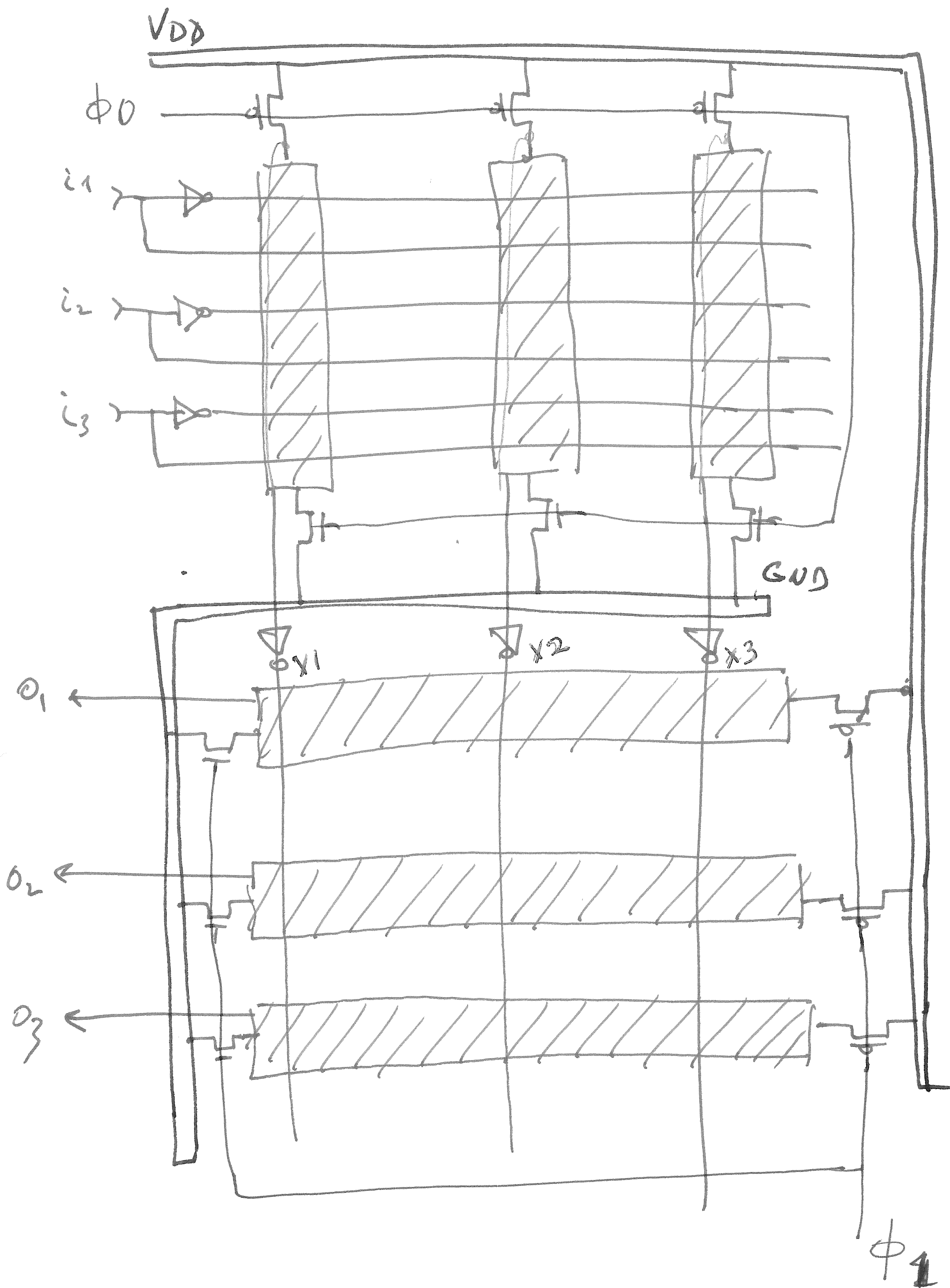


(Precharge on ϕ_1, compute first half on $\neg\phi_1$, second half on ϕ_0.) Can compute rather large sum-of-product expressions! (Several dozen terms with ten or so literals.)

Useful for:

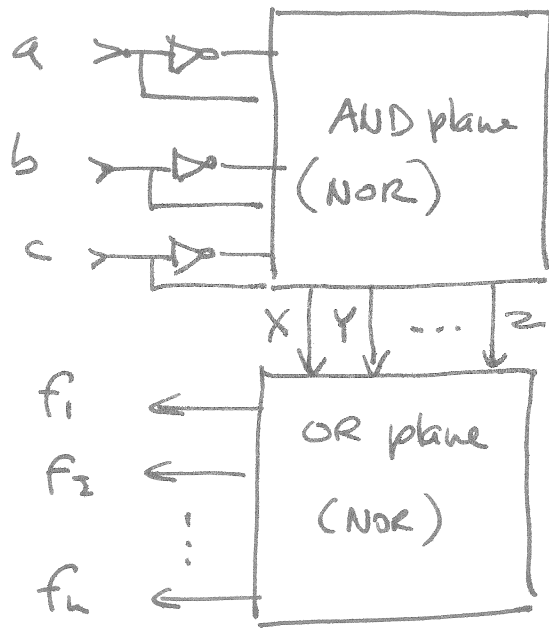
- Computation of complicated Boolean expressions.
- Implementing FSMs.

Copyright ©2010, California Institute of Technology. All Rights Reserved.

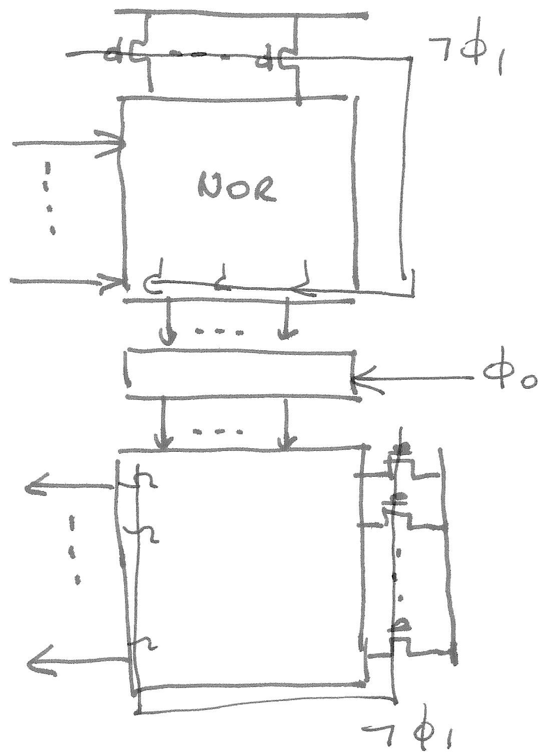


straight two-phase implementation ...

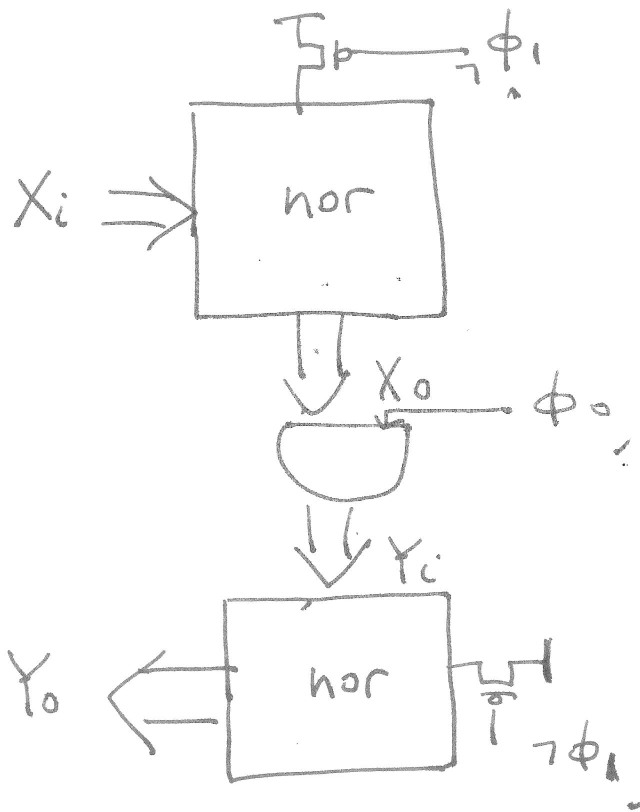
WRONG!



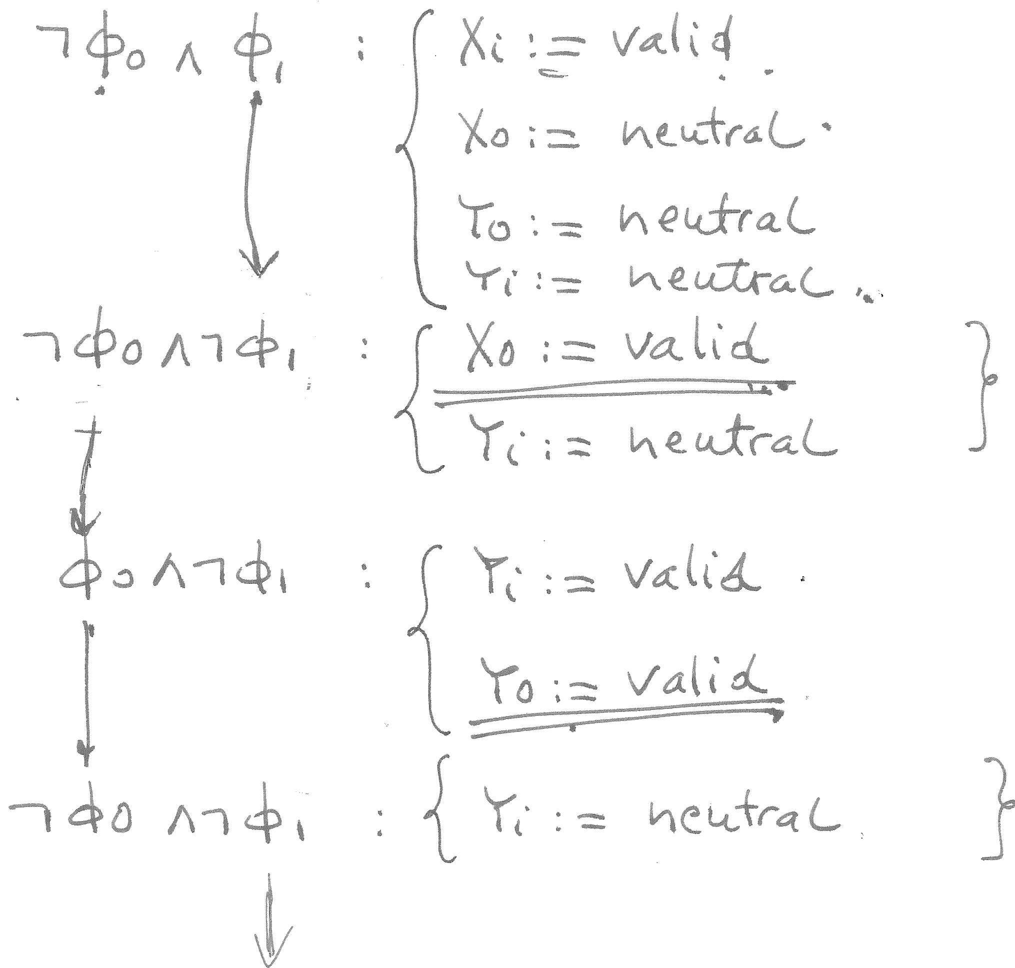
combinational



"tricky" clocking scheme



φ1
Xi
nor
Xo
φ0
Yi
nor
Yo
¬φ1



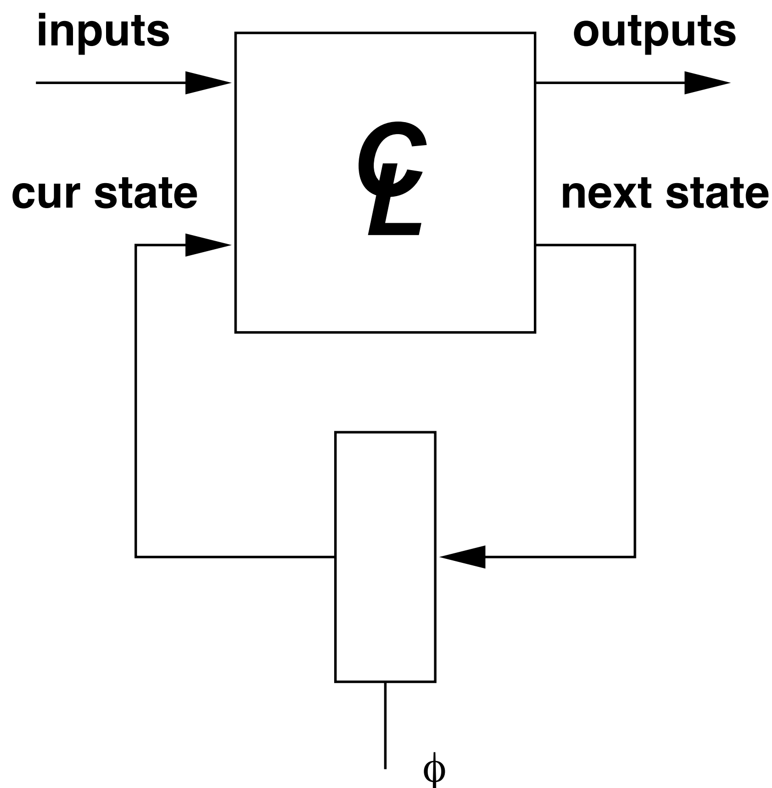
¬φ0 ∧ φ1 : Xi := valid
Xo := neutral
Yo := neutral
Yi := neutral
¬φ0 ∧ ¬φ1 : Xo := valid
Yi := neutral
φ0 ∧ ¬φ1 : Yi := valid
Yo := valid
¬φ0 ∧ ¬φ1 : Yi := neutral

Finite State Machines

Quick review of finite state machines:

$$I(t) \times S(t) \rightarrow S(t+1)$$

$$I(t) \times S(t) \rightarrow O(t+1)$$



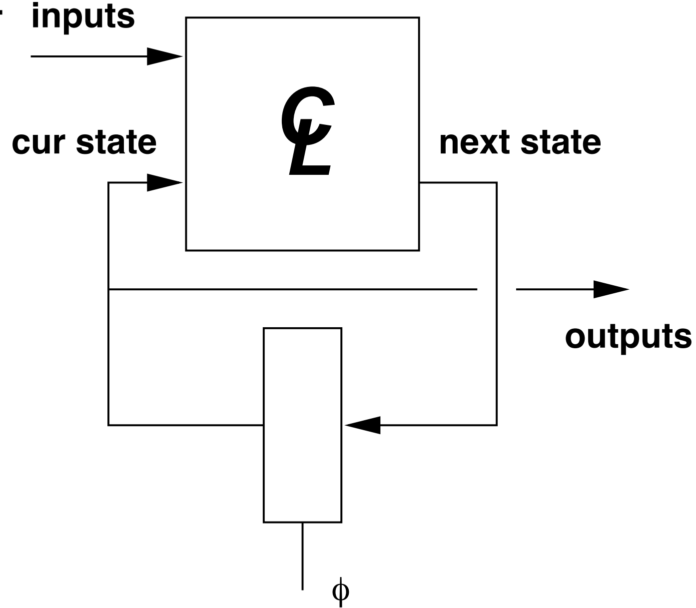
Powerful (and simple) model of computation.

We will be using the model mainly to discuss control circuitry.

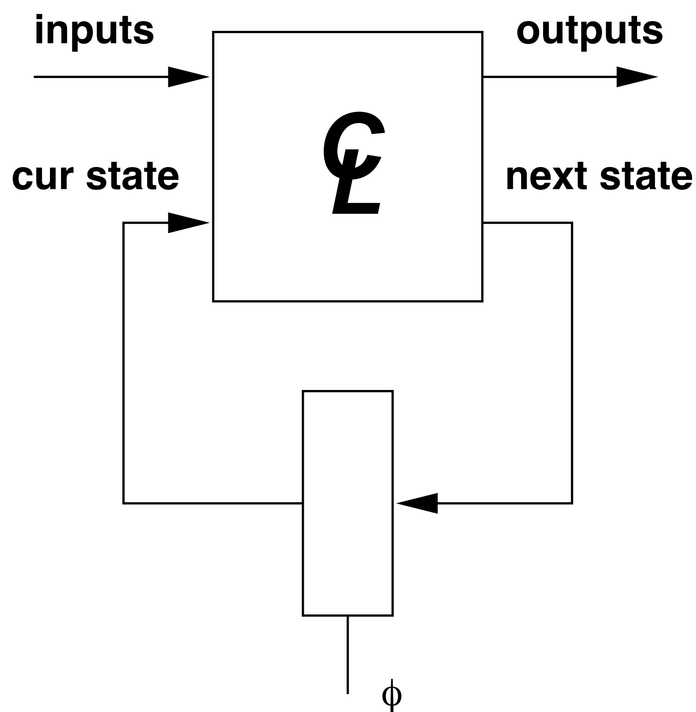
Copyright ©2010, California Institute of Technology. All Rights Reserved.

Technical details: *Moore* and *Mealy* FSMs

Moore FSM:



Mealy FSM:



Copyright ©2010, California Institute of Technology. All Rights Reserved.

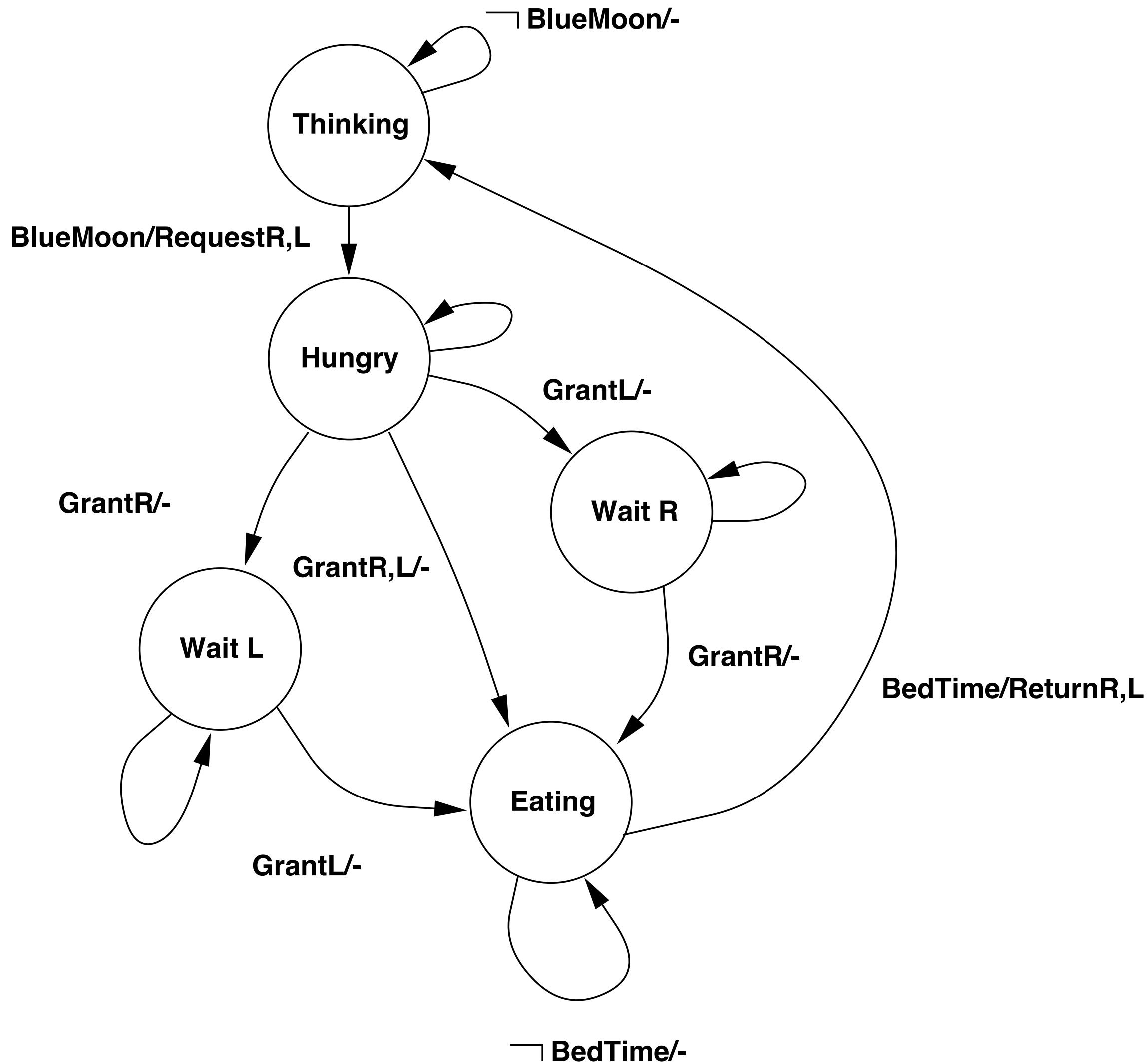
- In the Moore model, outputs are generated by inspecting the current state of the machine.
- In the Mealy model, outputs are a function of the current state and the current inputs.

—The two models are formally equivalent, although one or the other may in practice be more appropriate to any given problem.

Copyright ©2010, California Institute of Technology. All Rights Reserved.

An example

Let's consider a hypothetical FSM called a "philosopher." (Didn't draw every condition.)



We want to express this in the Moore model (outputs functions of current state alone).

Copyright ©2010, California Institute of Technology. All Rights Reserved.

Implementing the Philosopher

We decide to implement the philosopher FSM as a PLA.

Several ways to do this:

- Number the states in binary and generate a truth table that takes the value of the current state and current inputs and generates a new state on the next clock cycle.

Random example:

Current Inputs	Current State	Next State	Next Outputs
——	0000	0001	0000
0000	0001	0000	0000
0001	0001	0001	0000
0010	0001	0010	0001
11—	0001	0000	0000

Ensure:

- Always a proper next state. (Cover *all* possibilities.)
- Start in the right place (A *reset* rule—let FSM reset to state 0.)

Copyright ©2010, California Institute of Technology. All Rights Reserved.

An Easier Way

Doing the state assignment and figuring out all the outputs and next states is rather tedious. . .

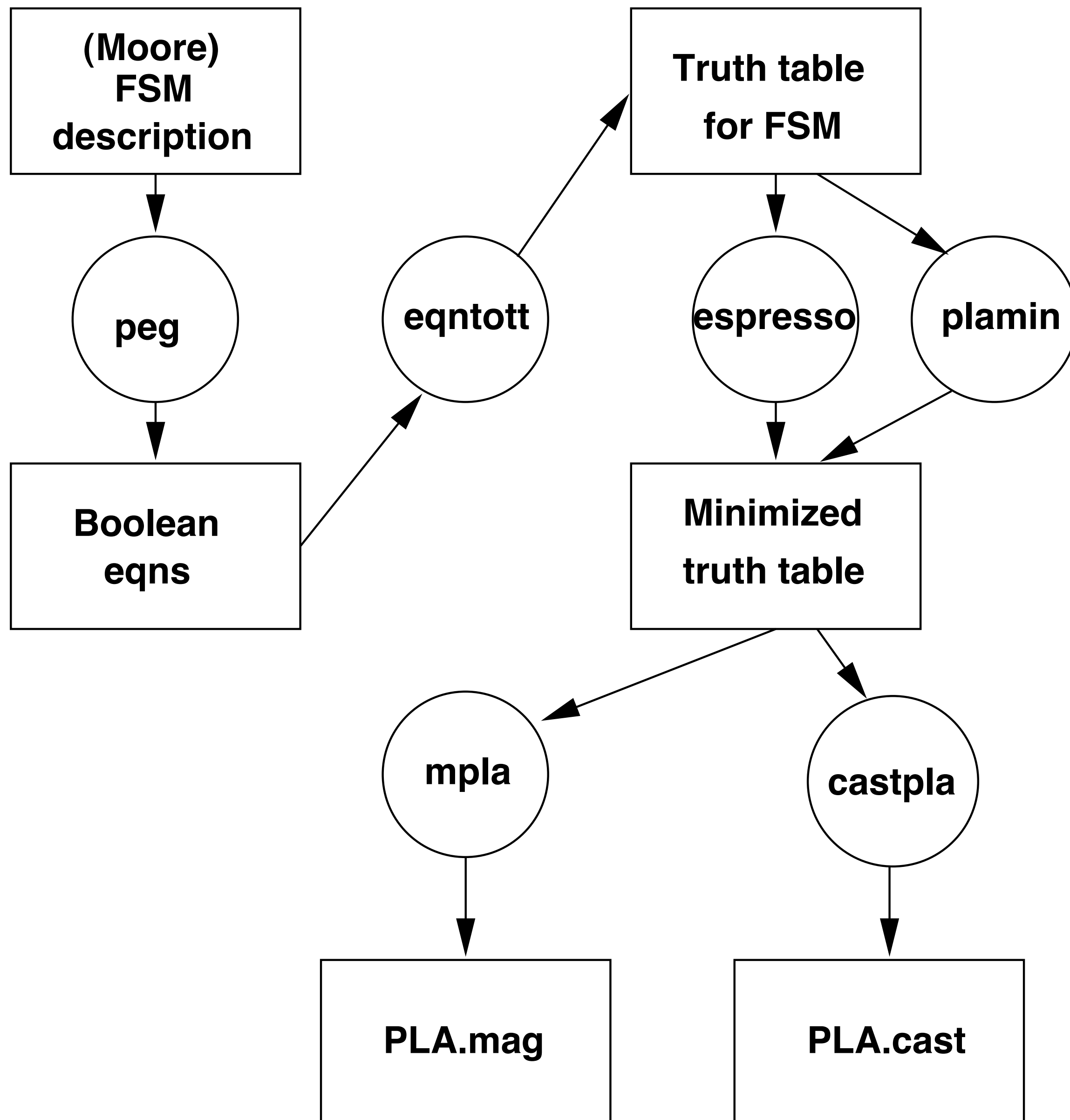
To help, we use peg:

The philosopher becomes:

```
INPUTS     :  RESET GrantR GrantL BlueMoon BedTime;
OUTPUTS    :  RequestR RequestL ReturnR ReturnL;
Start      :
Thinking   :  IF NOT BlueMoon THEN LOOP;
Hungry     :  ASSERT RequestL RequestR;
           :  CASE (GrantR GrantL)
                      0 0 => LOOP;
                      0 1 => WaitR;
                      1 0 => WaitL;
                      1 1 => Eating;
           ENDCASE;
WaitR      :  IF NOT GrantR THEN LOOP ELSE Eating;
WaitL      :  IF NOT GrantL THEN LOOP ELSE Eating;
Eating     :  IF NOT BedTime THEN LOOP;
           :  ASSERT ReturnR ReturnL; GOTO Thinking;
```


Copyright ©2010, California Institute of Technology. All Rights Reserved.

The tools: Use Manpage to learn them!



It's easy to make PLAs!

Copyright ©2010, California Institute of Technology. All Rights Reserved.

Putting it together ...

```
> peg phlsphr.peg | eqntott | espresso | mpla█
```


Copyright ©2010, California Institute of Technology. All Rights Reserved.

Where we are now

We have covered:

- Basic CMOS principles—n- and p-transistors.
- Basic restoring CMOS logic.
- `magic` layout—simple cells, datapath hierarchy.
- Sum-of-products Boolean minimization and more general switching networks.
- Clocking strategies, registers.
- Dynamic logic.
- PLAs and implementing FSMs.

Remains:

- Computer arithmetic (next class).
- Combine FSM control with datapath techniques (Lab 4).
- Make smaller, faster, better circuits (more or less rest of this term).

Copyright ©2010, California Institute of Technology. All Rights Reserved.