Harris

Introduction to CMOS VLSI Design (E158)

Lecture 10: Circuit Families

David Harris

Harvey Mudd College

David_Harris@hmc.edu

Based on EE271 developed by Mark Horowitz, Stanford University

Overview

Reading

W&E 5.4

Introduction

So far we have talked about the two most common forms of logic, static CMOS gates and switch-logic. But there are other forms of gates that people have invented to improve on some of the characteristics of logic gates. These improved gates have limitations of their own, and which logic family is best strongly depends on the application. As was mentioned in the previous lectures, static CMOS gates are extremely robust gates, and in many design styles they are the only gates that you are allowed to use. But in certain places, especially in custom layout, some alternate circuit configuration can come in handy to increase performance or reduce layout area.

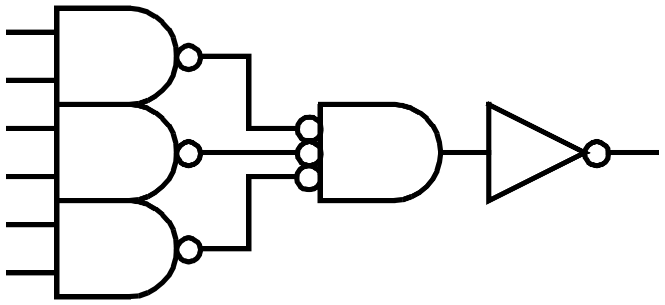
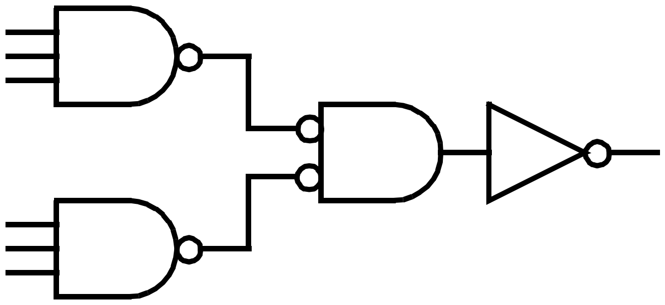
We will briefly look at a few of these alternative circuit styles.

Large Fanin

Notice that all the CMOS logic gates need a series stack, where the number of transistors in the stack is usually equal to the number of inputs. For NAND gates, it is a series pull down network, while for NOR gates it is a series pull up network. Since series stacks are slow, to limit the height of the stack we need to limit the fanin of the gate to 3 or 4.

If you need a large fanin gate, what can you do?

- Use a fanin tree

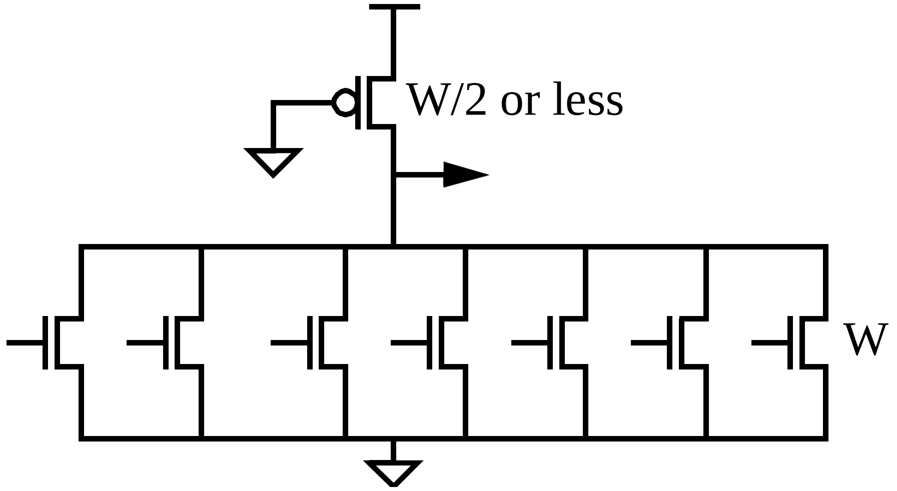



- Use pseudo nMOS (with various choices of pull up)

Pseudo nMOS: Circuits with Static-load Pullups

Using nMOS was great for high fanin gates. Since a *depletion* transistor (conducts even when V_{gs}=0) was the default pullup, you only needed to build the pulldown network. And for NOR gates, the pulldown network has only parallel transistors. But, CMOS does not have depletion pullups.

- Instead of using a depletion load, use a pMOS with its gate grounded



- But static load circuits have the nMOS problems: – DC power and ratio rules

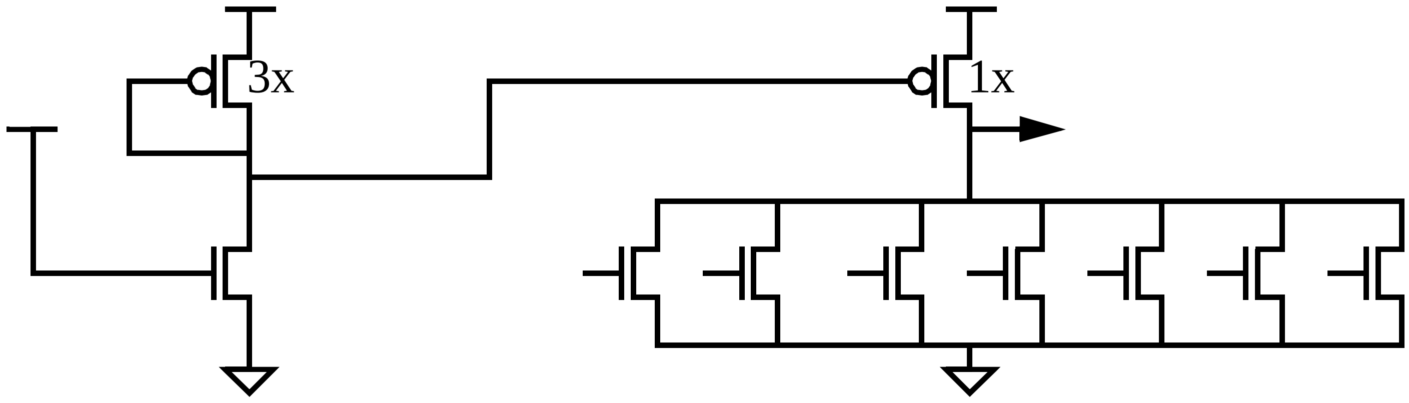
For a 4:1 current ratio, the nMOS width (w_n) must be twice w_p

+ Pseudo nMOS Load Choices

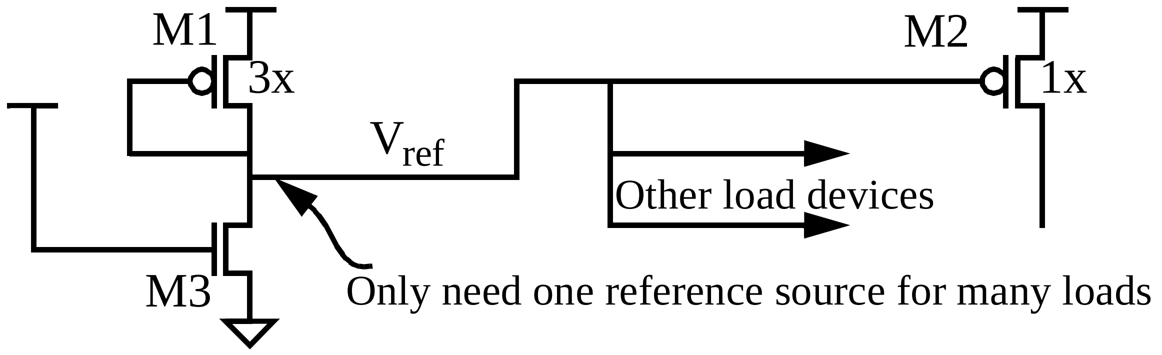
Better than just grounding the pMOS load, we can:

Make the pMOS current track the nMOS device (to reduce the variations in the ratio of the currents as the fab process changes) by using a circuit trick – a current mirror. The basic idea is pretty simple, rather than grounding the gate of the pMOS device, you control its voltage. Using feedback, you can set the voltage to make the current of the pMOS load to be some percentage of the nMOS current.

The feedback is easy to achieve; just connect the gate to the drain of the nMOS transistor with the current you want to track.



+ Current Mirror



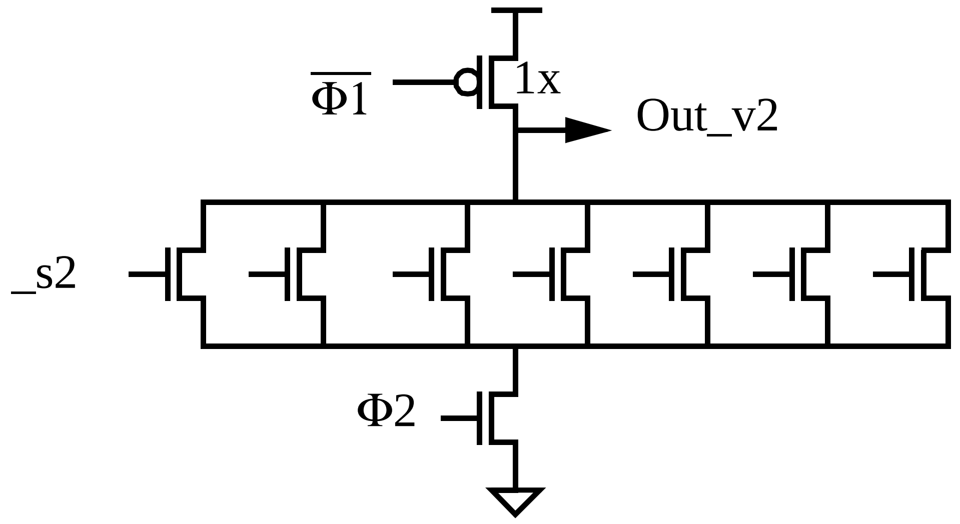
Since M3 is on, it pulls current out of the node ‘V_{ref}’. This will cause the voltage of the node to decrease, increasing the magnitude of V_{gs} on M1 which increases its current (decreases its resistance). If M1 is much larger than M3 (probably at least 4X) then it will prevent V_{ref} from being pulled to ground, and V_{ref} will settle on a level where the current through M1 is equal to the current through M3. If this gate voltage is applied to another transistor M2 that is 1/3 the size of M1, this transistor will have 1/3 the current in M1, which is 1/3 the current of M3. Thus, the current of M2 will track the current of the nMOS transistors, with the ratio set by the size ratio of the transistors in the current mirror.[1]

1. This better tracking allows you to run tighter current ratio between pullup and pull down, which makes the rise time of the gate faster. Using this technique, a current ratio of 3:1 or 2:1 is possible, instead of 4:1

Precharging For A Large Fanin Gate

Another approach is to “pre-pullup” the output. That is, the output is ‘precharged’ to a ‘1’, and is discharged only if the output should be a ‘0’

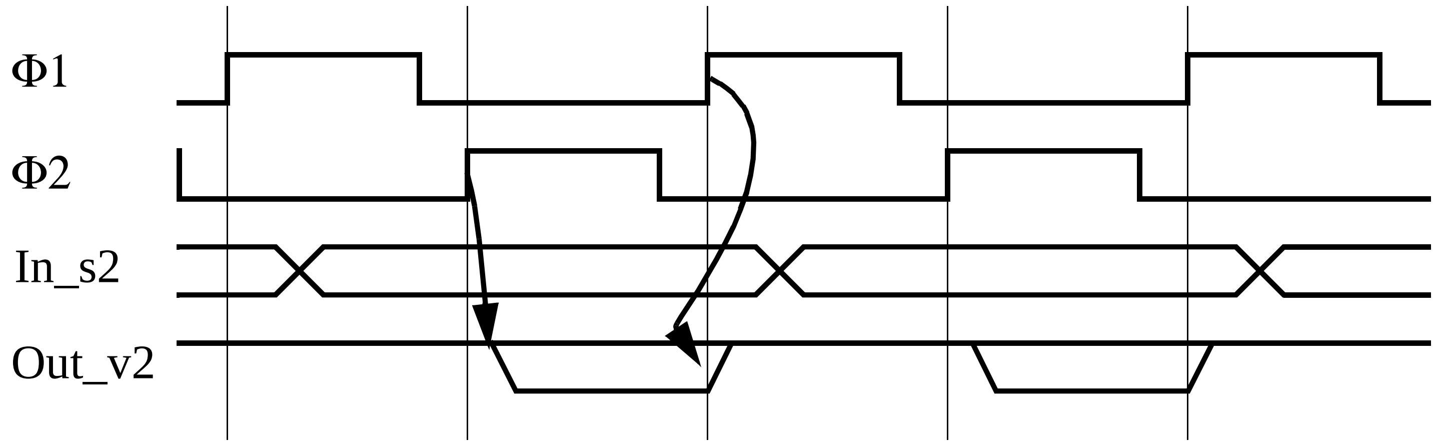
This allows us to build a gate like pseudo-nMOS with large fanin, but without the slow rise time (the output starts high) and without static power because there is no fighting when the n-tree pulls down.



The inputs must be stable when the pulldown network is activated, since if any inputs goes low again, the output can’t change back high. Once it is discharged, the output will stay low until the next time the precharge is asserted (usually the next cycle).

Precharge Gates

The timing for precharge gates in 2-phase clocking is easy

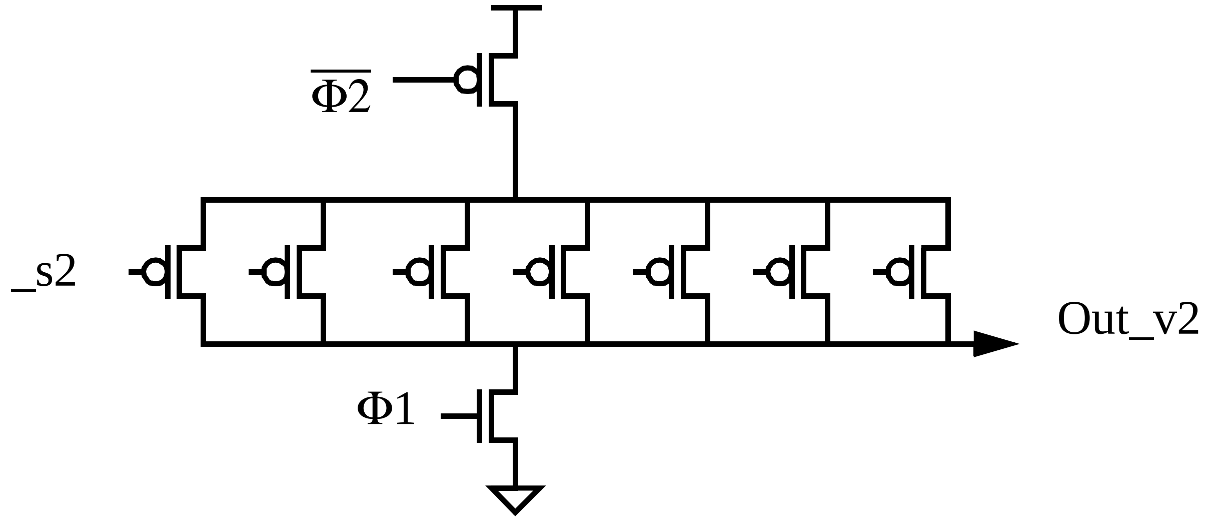


Could also choose to flip Φ1 and Φ2, and have the gate evaluate on Φ1

Key point is that the output settles some time after the evaluation clock goes high and does not change again until the precharge is asserted when the other clock goes high. This makes the output of precharge logic either _v1 or _v2 type.

Pre(dis)charge Logic

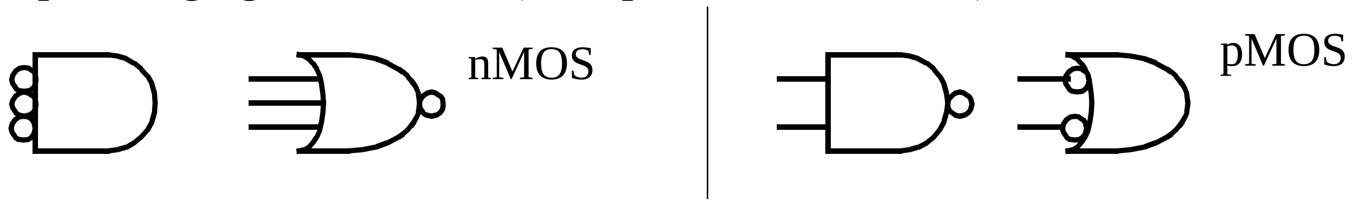
Could make the logic pre-discharge the output to zero, and then selectively pull the output up too. In this case, the logic transistors would be pMOS devices:



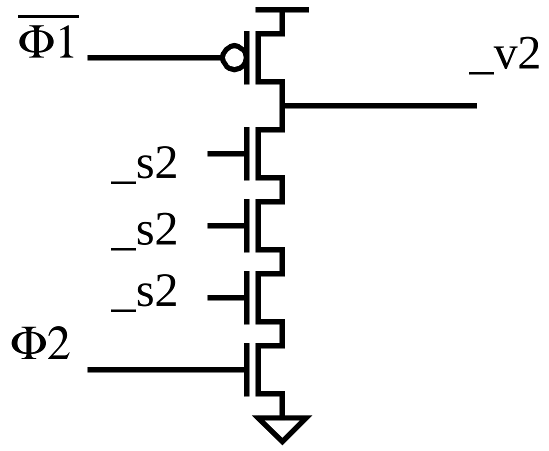
In this case, we evaluate the pMOS stack on Φ2, and predischarge the output on Φ1. Since pMOS devices are slower than nMOS, the nMOS style precharged logic block is usually preferred instead of this style, but both will work.

Precharge Logic

The good precharge gates to build (with parallel transistors) are:



These functions use parallel transistors instead of series stacks. For NAND gates using nMOS evaluation transistors you must unfortunately still use series evaluation transistors:



Advantages and Disadvantages of Precharge Logic

Precharged logic is fast:

- Only worry about speed of one transition. Can make eval trans. bigger
- Less gate loading, because only need half the transistors per input.

Precharged logic is dense:

- Only need one to build one tree, not both pull-up and pull-down.

But you can’t cascade these gates together directly in strict timing model.

- Inputs to the gate must be _s1 or _s2
- Output of the gate are only _v1 or _v2
- For full generality, only static gates can follow precharge gates directly.

We can make precharge outputs legal inputs to another precharged gate

- Need to invert output and take advantage of monotonic signals.

Cascaded Precharged Logic

Take advantage of the fact that an nMOS precharge logic gate’s output can only fall when it evaluates. This means that the inverted output only rises.

- The inverted output of a precharge gate starts at zero.
- It either stays at zero, or it rises.
- If it is an input to another nMOS precharge gate:
  - When it is at zero, it will not cause the output to change.
  - If it transitions to 1, it might cause the output to change.
  - It can’t change from 1 to 0

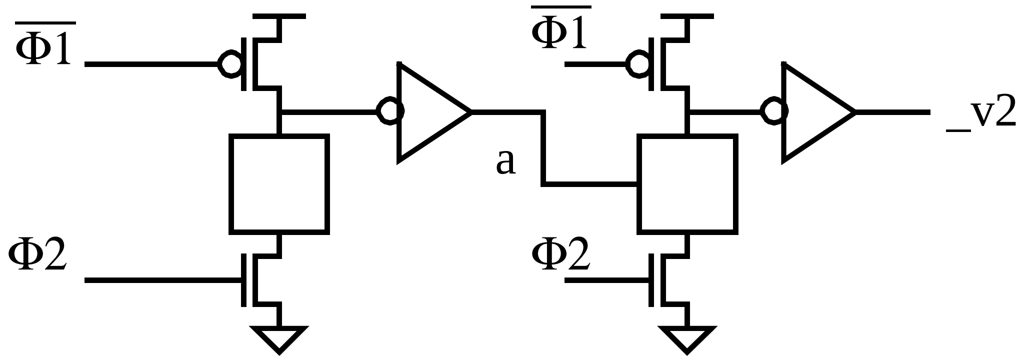
    Which was the reason we wanted _s inputs in the first case

    These inputs can’t cause the gate to fail
- So, inverting each output makes it a valid input to another precharged block.
- The inverters also serve as good buffers for driving the long output wires

The output of cascaded dynamic logic falls like dominos, it is called domino logic.

Domino Logic

Allows limited cascading of precharge gates



During precharge ‘a’ is low. Then during Φ2, it will either stay low or rise. If it stays low, it will not affect the next gate, and if it rises it causes the next gate to discharge, and that is the desired output.

Can cascade gates, but there is one more limitation to overcome...

Domino Logic

All domino gates must be monotonic in their inputs. That means when an input rises, this change can either not affect the output or cause the output to rise. A rising input can’t cause the output to fall.

Monotonic logic is not complete. You can’t build an inverter!

- In fact, can’t build any inverting logic directly.

  Makes sense, since all MOS stages invert, and each precharged gate always has exactly two inversions --> non-inverting.

- But, this too can be easily overcome.

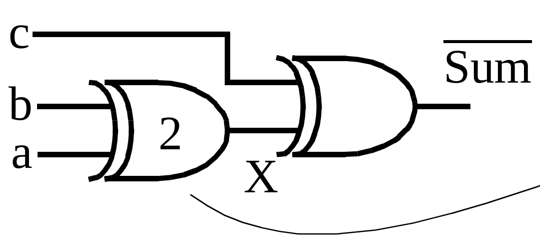
  We can build any logic function as long as all inputs and their complements are available (use DeMorgan’s law to push the inversions around). We can generate both true and complemented outputs with two precharged gates, so subsequent precharged stages will have both polarities of the inputs it needs available too.

  In the worse case takes 2x number of static gates. Usually not all gates need to be duplicated, so the total is less than 2.

Dual-Rail Logic

The reason the maximum overhead is 2x for monotonic gates is, in the worst case, you need to compute the true and complement for each logic function, since some future function might need both the true and complement outputs of this function.

Look at adder example:



Generate $a \oplus b$ and $\overline{a \oplus b}$

$$a \oplus b = a\bar{b} + \bar{a}b = X$$

$$\overline{a \oplus b} = \bar{a}\bar{b} + ab = \bar{X}$$

Generate $c \oplus X$

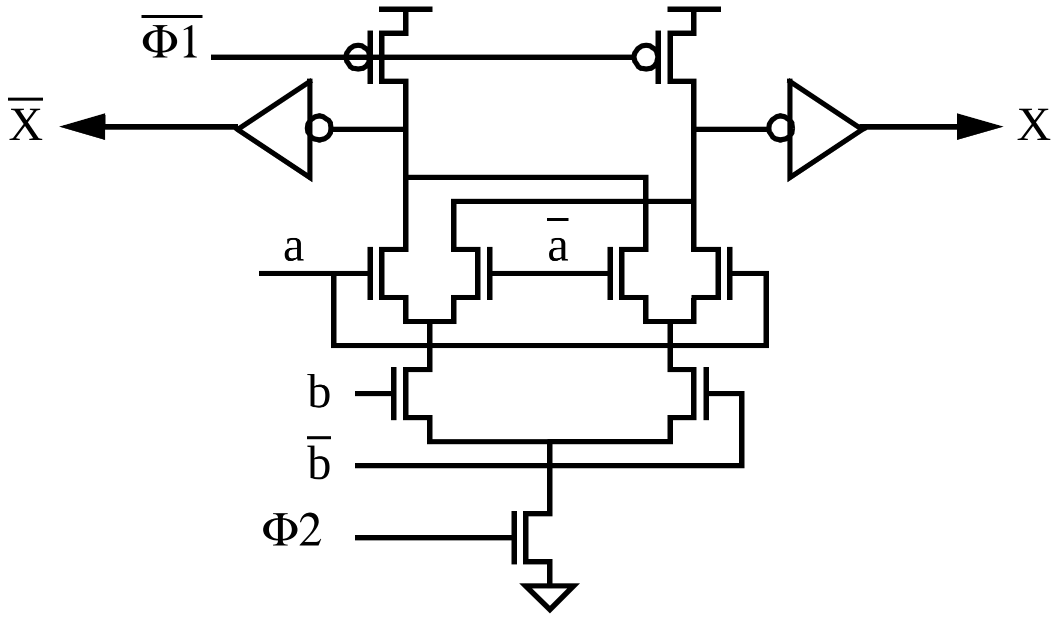
$$c \oplus X = c\bar{X} + \bar{c}X = \text{Sum}$$

Can be built with three precharge gates this way. But wait.

Dual-Rail Gate Example

Rather than building two separate gates for generating x and $\overline{x}$ you can use one merged gate. This gate is similar to a switch network that routes gnd to either out or $\overline{out}$.

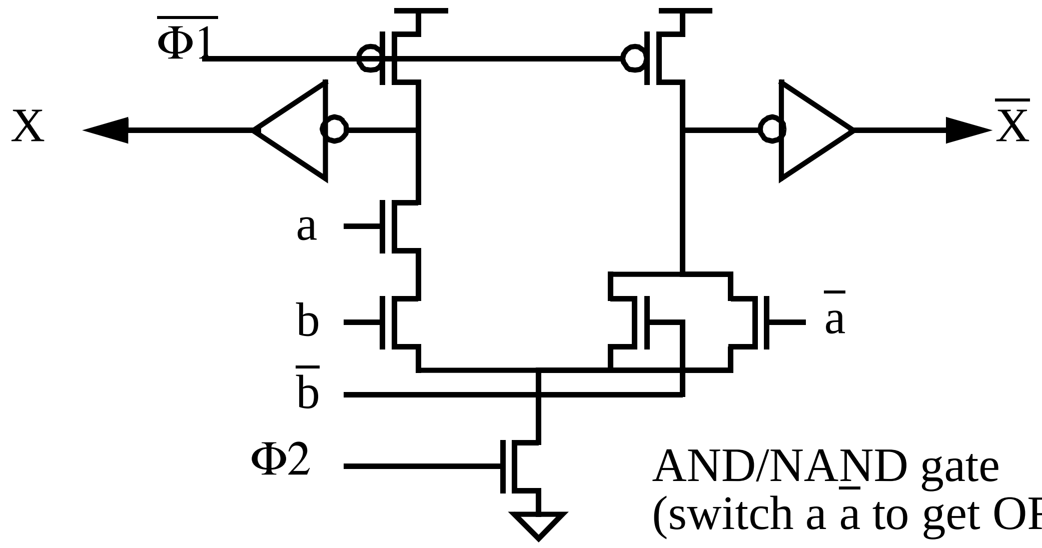
A Dual-Rail EXOR gate:



Dual Rail Constraint

In a dual-rail gate you need to implement the function F and $\overline{F}$

- You have as inputs a_i and $\overline{a}_i$
- So you end up building $\overline{F}$, using $\overline{a}_i$, and it is the dual of F
  - Dual implies parallel devices becomes series
  - Implies all dual rail gates have series structures
  - Can’t build large fanin dual rail gate!



AND/NAND gate
(switch a $\overline{a}$ to get OR/NOR)

+ Dual-Rail Signalling

- Most gates don’t share as many transistors as the EXOR gate

  AND / OR gates don’t really share any transistors.

- But, in general, dual-rail gates **are** complete in logic function

  Generate monotonically rising true and complement versions of all outputs. We can take advantage of one more thing:

- Each signal can indicate both value and completion. For a Signal ‘A’:

Table 1:

A^H	A^L	Meaning
0	0	Reset = Not yet evaluated
0	1	Ready with value FALSE
1	0	Ready with value TRUE
1	1	Not used = Never Occurs = Error

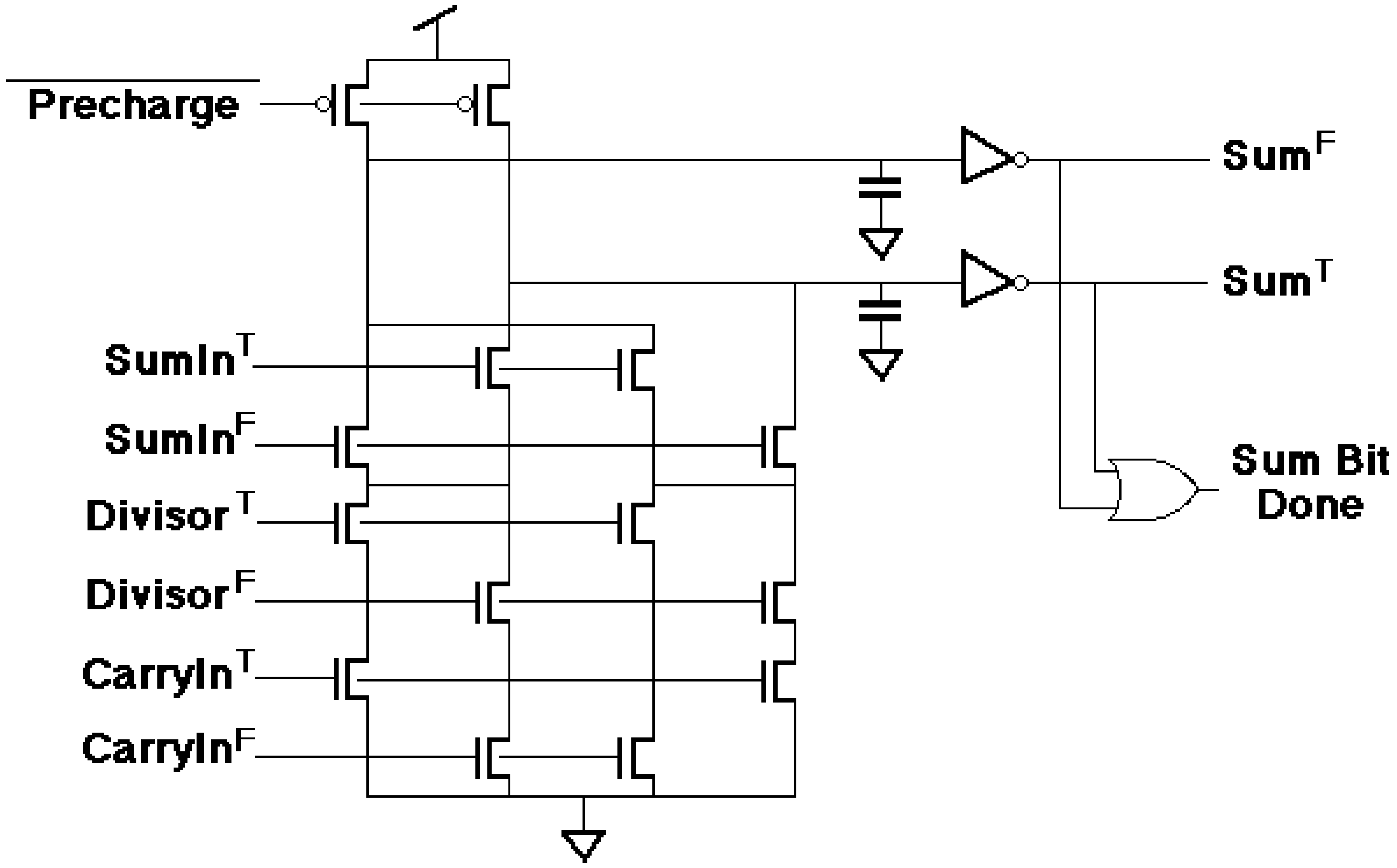
- Signal (on two wires) is not yet valid, if both wires are low.

  When one wire is asserted, the signal has become stable.

  Great for a type of sequencing discipline called self-timed design.

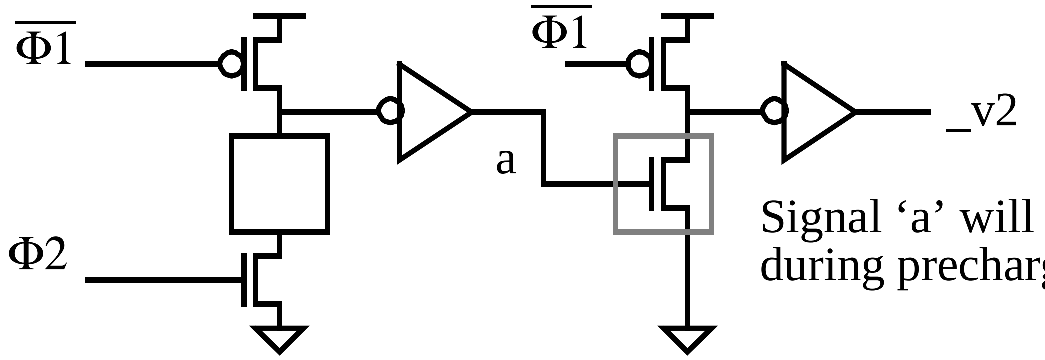
+ A Dual-Rail Gate w/ Completion

A Full-Adder bitslice has two outputs: Sum and Carry. The Sum logic is:



+ Removing the Evaluate Transistor

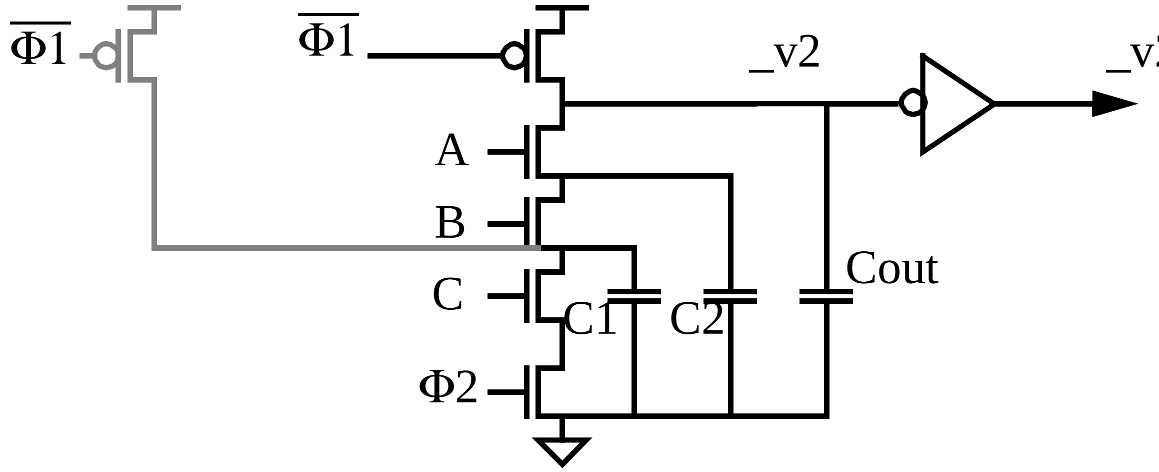
If all the inputs come from other domino gates, then all the inputs will be low during the precharge. You don't need the explicit evaluate transistor.



- Reduces the height of the series stack of nMOS devices which is good
- Need to be a little careful. When precharge begins, the first gate's output must precharge before the next gate can precharge. Both evaluate and precharge ripple in this scheme. But, if there is already a tall stack, transistor ratioing will let precharge win anyway. (but you waste power until the precharge ripples)
- Don't do this in this class, but you can do this on real designs (remove every other evaluate transistor)

Charge-Sharing in Precharged Logic

Since domino logic gates don't invert, can't use NOR gates to generate AND functions. This means to get ANDs you will need to build gates with series devices:

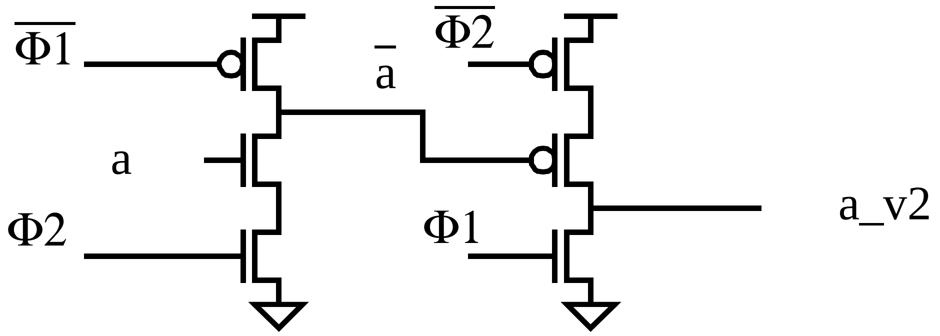


You need to watch for **charge-sharing** in the series devices. If A and B rise, but C remains low, then C1 and C2 will share charge with Cout. Since A and B could have been low during precharge, C1 and C2 might be low, and Cout better be much larger than C1+C2. If that is not the case, you can add more precharge transistors, as shown in grey.

+ Precharged nMOS and pMOS Together

Just say no!

Precharged nMOS gate can follow a precharged pMOS gate (NORA):



Don’t do it!

- The output of the nMOS gate is normally high, keeping the pMOS off, without requiring an inverter between the two stages.

But there are two LARGE issues:

- AND gates still require series stacks (does not buy anything)
- Noise margin is very small. Power supply variation counts as noise too
  - If the output of the nMOS drops just V_{th} then the pMOS transistor can turn on, and you will get the wrong output!

+ Optimizing for Single Edge Circuits

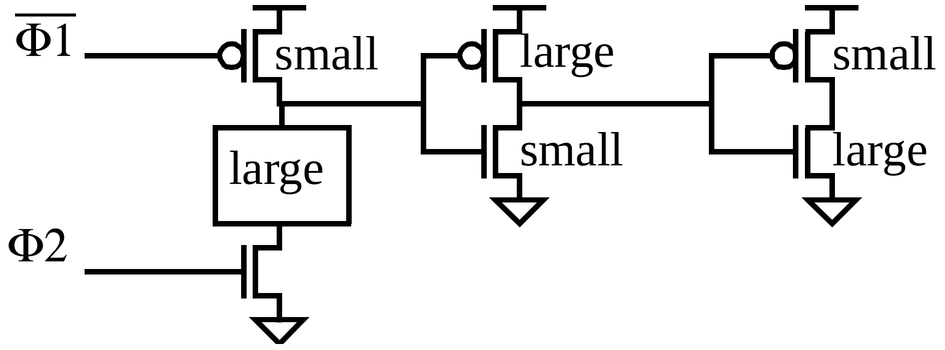
In domino logic circuits, only one edge is significant – the rising edge

Can optimize the speed of this edge

- Make nMOS transistor in the inverter much smaller than the pMOS

    Lowers the loading on the precharge gate

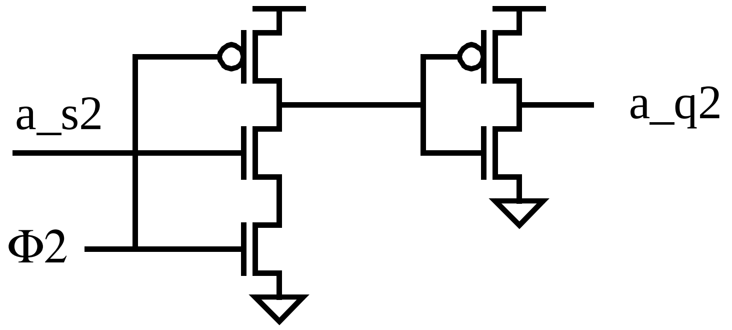
- If the following gate is not a precharge gate, make the pMOS transistors in this gate much smaller than the nMOS



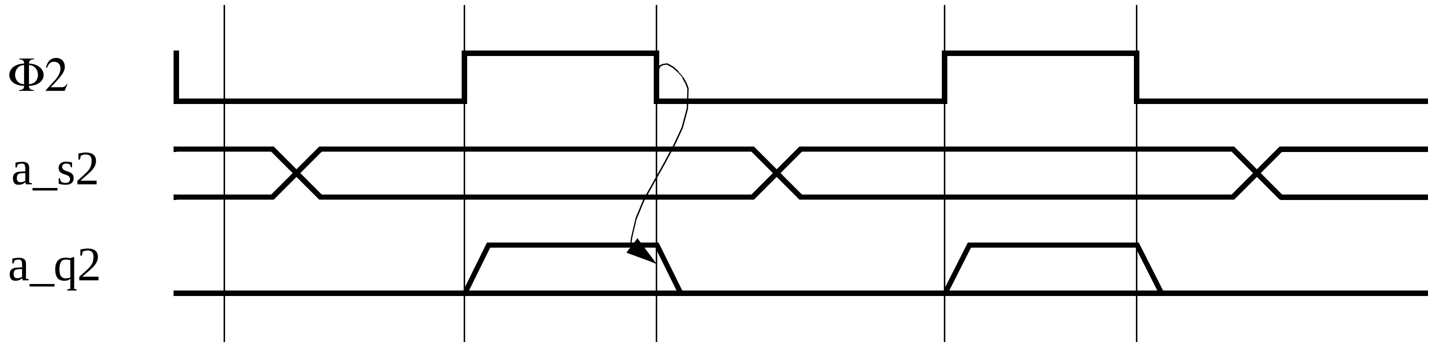
- Gate delay can be cut almost in half! (If large >> small)
- But the other edge becomes very slow, so must allow time somewhere

Clocked AND

If we use the same clock for precharge and evaluate, we create a qualified clock generator:



- This is a special precharged NAND gate
- Notice when the precharge occurs

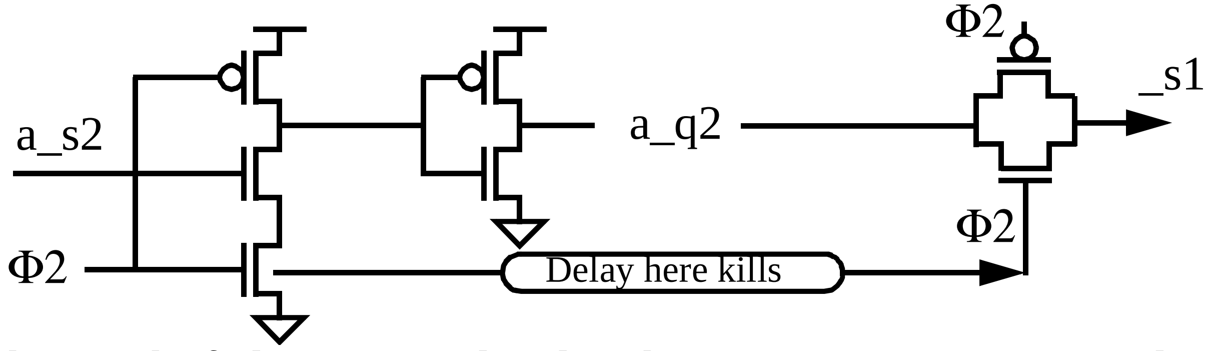


Clocking Issues

Just say no!

- If we used $\overline{\Phi 1}$ or $\overline{\Phi 2}$ for the precharge control signal, then we generate a true _v1 or _v2 output, which doesn't change again (precharge) after evaluation, until rising edge of the next clock Φ1 or Φ2.
- Using one clock doesn't generate a _v signal, so we can't use them as data for a latch. They are really of timing type _q1 or _q2.
- A two-phase clocking will always work, independent of clock skew, but if we use _q1 and _q2 as data, then must make sure there is no race between the output of the precharge gate and the following latch. Don't want the reset to get through latch too soon:

Don't do it!



- Since this might fail, it is not a legal 2 phase circuit. So DO NOT do this!

Another Circuit Idea

In this lecture, we have talked about three primary logic styles:

- Fully-Complementary Static CMOS
- Ratioed-Logic Pseudo-nMOS
- Precharged Logic

But, remember, in Lecture 3 we talked about one more primary style:

- Switch Logic

Switch logic just uses transmission gates to steer around data, for example, in n-way multiplexors.

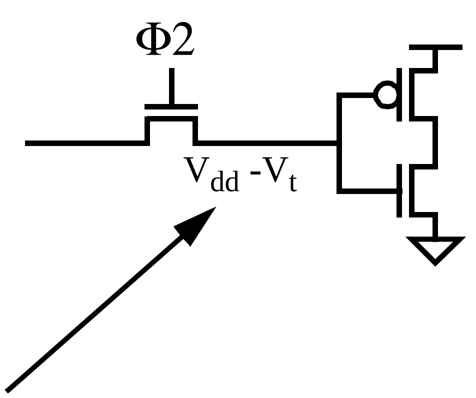
When transmission gates are controlled by clocks, we call them latches.

But for both switch logic and latches, we have the nasty problem of needing both polarities of control for a full transmission gate. If we instead used just the nMOS transistor, then could use just one polarity of data (for switch logic) or one polarity of clock (for latches).

Using nMOS Pass Gates for Switch Logic or Latches

It can be nice not to have to generate the true and complement controls for transmission gates. Why not just use nMOS pass transistors, and leave off the pMOS device?

Dynamic Latch



- Advantages:
  - One clock or polarity of data
  - Lower capacitance on both gate and source/drain diffusion

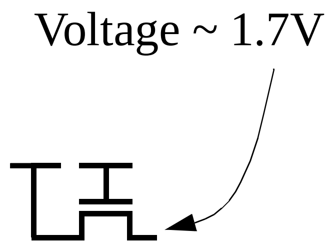
- Disadvantages:

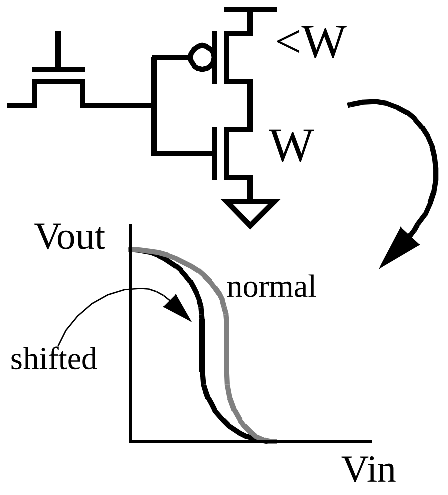
  Degraded levels because the switched node doesn't get fully pulled up
  - Lower noise margins
  - Static power dissipation in the following gate/inverter
  - Limited power supply voltage scaling

Dealing with Degraded Levels: Adjust Inverter Ratio

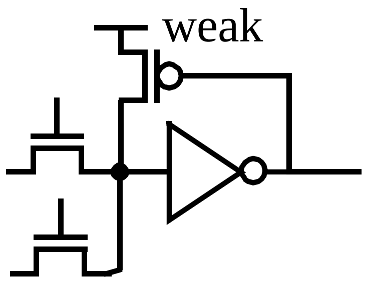
The high output of an nMOS transistor is V_{th} down from power supply.

- The threshold voltage (V_{th}) depends on the source voltage, and at high source voltages, it is larger than when the source is at ground (body effect, back-body bias). So the output only really gets up to about 1.7V for VDD = 2.5V
- Want the following inverter to have a low switching point since the input isn't very high. This means that the inverter following an nMOS-only pass gate should have a nMOS ≥ pMOS device
- Even when the input is high, 1.7V will leave the pMOS slightly on, so the following inverter or gate might dissipate DC power





Weak pMOS Restore



- When the output of the inverter goes low, the weak pMOS will turn on and will pull the input of the inverter to a good level. This will make sure that there is no wasted current nor DC power draw in the following inverter.
- The pMOS device must be weak, since the nMOS device must fight it to pull the input of the inverter low. The pMOS does not turn off until after the gate switches.
- The weak pMOS does **not** help the noise margin issue, since it does not do anything until after the gate has switched.
- This circuit is also used on the output of precharge gates to make them static. The weak pMOS devices hold the outputs high even with leakage currents or charge-sharing. But remember they are weak, and so the recovery after charge-sharing may be slow.
- Risk: If there is too much (wire) resistance leading up to the inputs, the voltage divider with the weak pMOS may mean it never switches.

nMOS Pass Gate Summary

- With adjusted inverter threshold ratio, and weak pMOS restore, it makes nMOS-only transmission gates good enough to be used for switch logic and latches.

But, why go to all the trouble?

- For switch logic, it is most relevant when there is not just one or two, but many inputs to a node. For example, an 8-way multiplexor. Save 8 pMOS transistors, by replacing with one weak pMOS restore.
- A multiplexor can be built into a latch by using qualified clocks controlling just nMOS pass gates, all followed by just the single inverter and weak pMOS level-restoring transistor.

Why you should avoid doing this? (Don’t use nMOS only for this class)

- Limits the supply range of your design (fails at low voltage)
- Need to make sure that your tools handle structure correctly

  Need to find the conflicts / leakage currents