End of lecture 9:

More about wires and wire models

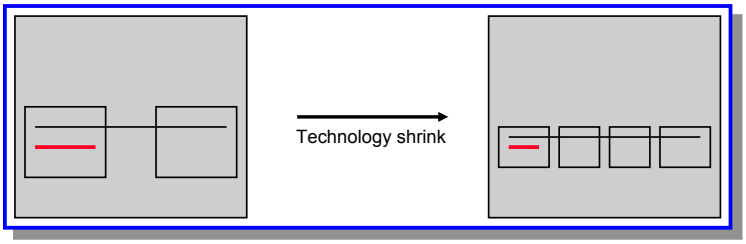
Computer Systems Laboratory
Stanford University
ronho@vlsi.stanford.edu

Wire scaling

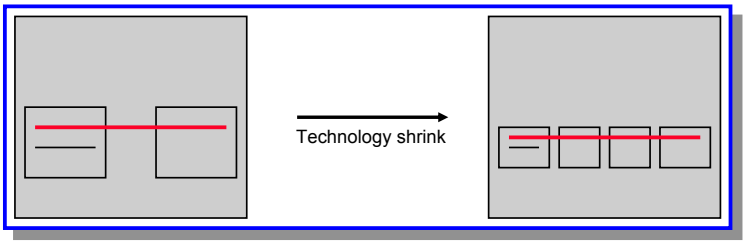
- Key is to remember there are two kinds of wires
  - Wires with a constant logical reach
    - All the wires when you shrink a chip to a new technology
    - All module level wires
    - L scales, so the delay of the wire decreases as T/W grows
    - Ratio of wire to gate delay grows, but slowly
  - Wires of constant length
    - Global wires spanning a chip
    - Delay increases, while the gate delays decrease
- Duality should not be surprising
  - Communication has some cost
  - Starting to have communication delays on-chip
  - Today's chips are like yesterday's boards

Scaling: two kinds of wires

Local wires, inside modules or blocks, get shorter under scaling



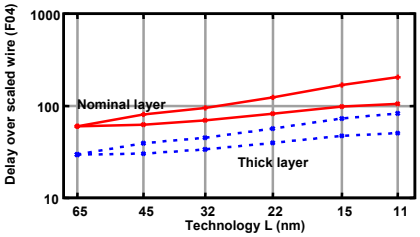
Global wires, connecting modules together, do not get shorter



D. Sylvester *et al.*, "Getting to the bottom of deep submicron design," *IEEE/ACM ICCAD* 1998.

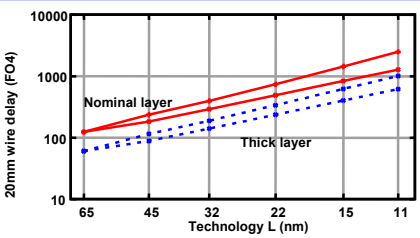
Scaling: two kinds of trends

Local, scaled length wires (with repeaters) are okay



Repeated delays stay basically constant over technology Scaling (2.5x over 6 generations)

Global, fixed length wires (with repeaters) are not



Repeated delays get fairly large compared to gate delays (30x over 6 generations)

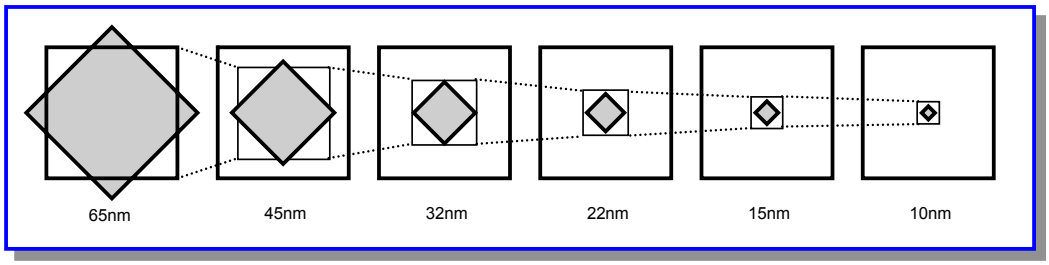
Scaling: another perspective

How far can a wire “reach” in a clock cycle?

- Assume a 35FO4 clock cycle and a small 85mm^2 die

Each technology scaling step reduces everything

- Except die size: assume die complexity grows
- Wire reach falls a little faster than the original block size



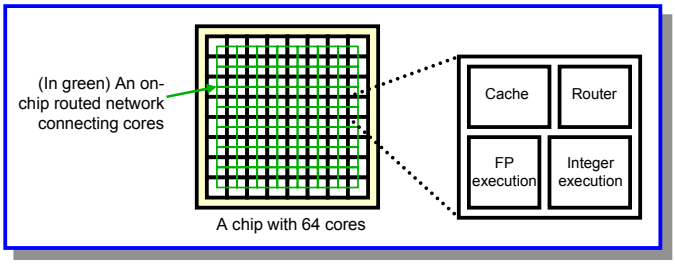
Scaling implications for architecture

Look towards architectures that can handle the duality of wires?

- Slow global communication does not throttle performance
- Performance is set by fast local (scaled-length) wires

Answer: modular architectures using multiple tiled cores

- Cores are built with local wires that are fast and that scale well
- Global core-to-core communication is explicitly slow
- Can even make global communication visible to the programmer

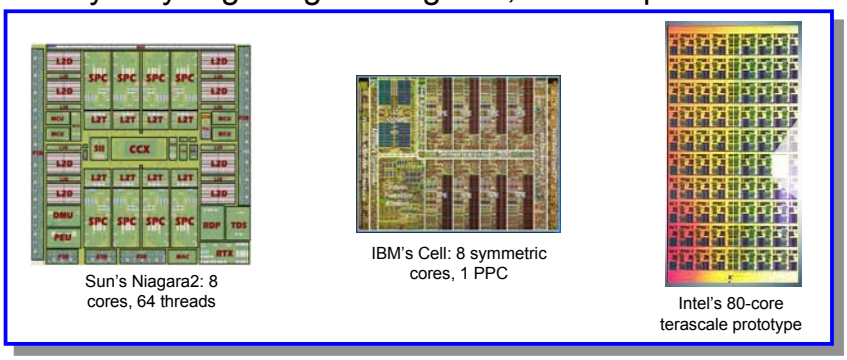


Scaling: modular machines

Not a new idea: lots of early work in mid- to late-1990s

- Academic: MIT RAW, Stanford Smart Memories, UT GRID/TRIPS
- Industry: Dual-core Sparc, Pentium, and Opteron processors

Now everybody is getting in the game, and in spades



G. Grohoski, "Niagara2: A highly-threaded server-on-a-chip," *IEEE Hot Chips*, 2006.
J. Warnock, *et al.*, "Circuit design techniques for a first-generation Cell broadband engine processor," *IEEE JSSC* Vol.41, No.8, August 2006.
Intel Developer Forum, 2006.

Modularity: good for other reasons

Moderate energy using selective block power-down

- The popular question in 1997: "What do you do with *x* billion xstrs?"
- The practical answer in 2007: "Don't turn them all on at once!"
- Deep sleep for blocks you aren't currently using

Manage design complexity with divide-and-conquer

- CPU teams regularly exceed hundreds of designers
- Yet Intel's Pentium4 design team had only a handful of "wizards" [1]

Modularity can lead to reconfigurability and IP reuse

- Various estimates of ASIC design costs in 65nm: $100M
- So build "general-purpose ASICs" with reconfigurable cores/logic
- Exploit heterogeneity in the cores for repurposing logic

[1] B. Colwell, "CPU performance: complexity is a spent force," Industry talk, 2005.

Lecture 10

Why Circuits Fail: What to Evaluate

Computer Systems Laboratory
Stanford University
horowitz@stanford.edu

Overview

Reading

Lev	Signal and Power Network Integrity
Gronowski	Dynamic Logic and Latches (talk version of Chapter 8 in Chandrakasan)

Introduction

The basic components on a chip are quite simple: transistors and wires. The key point to remember is that wires are a critical part of your design, and they need to planned for, and simulated like, the transistors. Thus an effective schematic is part schematic and part floorplan. This allows designers to optimize a circuit's devices and placement to improve its speed, power, or area. Improving these parameters often makes the circuit more complex, which needs to be managed too.

Designers generally use simple circuits, because they are more robust than the 'fancy' designs you read about in journals. This lecture looks at some of the things that can (and have) gone wrong in ICs.

Cost of making mistakes

Design Stage	Approx. Cost	Effort Involved
Initial Design	$10	5 minute fix
Design Review	$100	1 hour re-work
Layout	$1000	10 hours –schematic, layout, simulation
Assembly / Tapeout	$10,000	50 hours rework, validation, re-stream
A-step Silicon	$100,000	200 hours debug/fix, equip costs, new stepping
Sampling	$1,000,000	Delay product launch
Volume Production	$10M to $500M	Product Recall

From J Stinson, Intel

Complexity

Why use complex circuits?

- Generally forced into needing them
  - Since the simple circuits don't make the spec
- Usually in the critical part of the design.
  - If it isn't critical, why not use the simple solution
- Fighting $\Delta t = C\Delta V/I$

Designers must choose their risks carefully

- Therefore, designers become fairly conservative
- Must validate designs over all corners and chip environments

Static CMOS is the "gold" standard (K.I.S.S.)

- Use it unless it won't meet your spec!

Things that go wrong

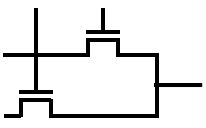
- There are lots of reasons why chips fail. Here are four:
  - Simulation vs reality mismatch
    - What you simulated was not what you built
    - You forgot to simulate an important case
  - What is your reference
    - The reference was more noisy than you expected
  - Signal Coupling / Writability issues
    - Signal voltages were affected by other nodes switching
  - Races / Pulses
    - With a two sided timing constraint it might be too long or too short

Simulation mismatch

Often stupid stuff but the errors still mean that the chip fails:

Case sensitivity: A chip had two nodes phi1 and Phi1

- There were not connected together on the chip
- Extractor did not complain (different nodes)
- Simulator connected them (case insensitive) (it worked fine)
  - We fixed this error in the tool chain



There was a mux which sometimes had both selects high

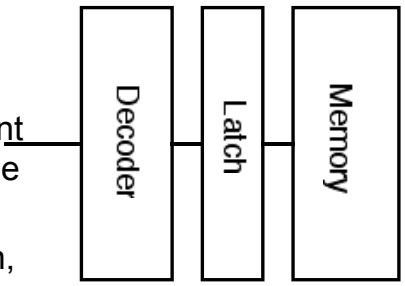
- During the tests the output of the mux did not matter
  - So the chip passed logical validation
- The mux was modeled as a unidirectional logic element
  - But of course pass gates are bidirectional
  - In operation, one input would corrupt the other…

Discounting false paths

In the real world, voltages are analog. This is almost never a problem

But there once was a chip that had a long false path to the memory

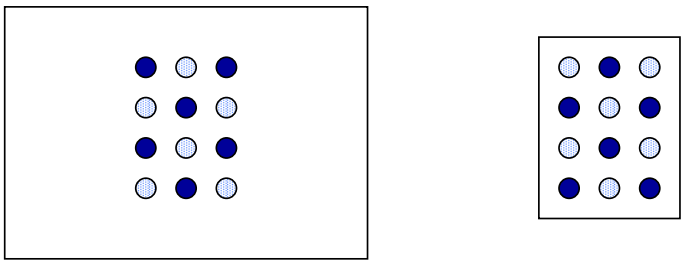
- In this case the decoder output did not settle in time (for the latch)
- The memory was not used in this case, so the address did not matter
- But the decoder outputs settled at different times, so for this long path, it was possible for multiple decoder outputs to be high
- This caused multiple wordlines to go high, which caused one of the locations being read to be written (corrupting real data)



Missing simulation test cases

3D integration: two chips were to be stacked face-to-face

- Face-up chip was big, face-down chip was small
- Solder balls would bond the two chips together for Vdd, Gnd
- Each chip had a field of Vdd and Gnd pads laid out in an array



- The problem is probably pretty obvious
  - But no tool flow exists to check “assembed 3D-stacked chips”!

References

- There are no absolute voltages. Voltage is the potential difference between two points. Any circuit that measure the value of a node (to decide whether it is a one or zero) is really measuring the value of that node relative to some reference. It is important to understand what the reference is, especially when you are using tricks to reduce the needed voltage swing. You need to ensure that the noise between the signal you want to measure and the node is small.

Look at a couple of different problems:

- Input pins
- Precharge gates
- Current source matching

Input pins

- For a normal gate the reference is set from its power supplies:
  - Switching point is relative to Vdd/Gnd
  - Really Vs = VGnd + X(VVdd - VGnd)
    - Where X is the switch point(as a percent of supply)
- If the gate is connected to a chip input
  - Input signal is references to board Gnd
  - But gate reference depends on chip Gnd
  - If the two are not the same, you can get into trouble
    - And board Gnd and chip Gnd are never the same

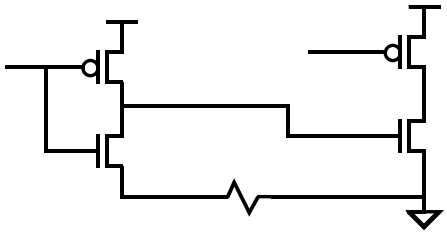
Vss reference (Gnd drops)

Margins are much smaller in a precharge gate

- Switch point is just Vth above Gnd

But Gnd on a chip is not the same

- With Amps of current running through supply can get Gnd drops



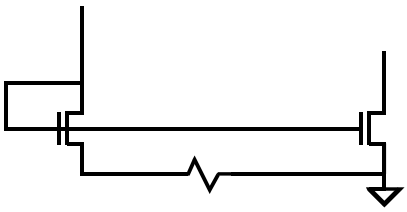
- This can make the margin on a gate even smaller

Gates with keepers have slightly higher margins

- Often require inputs to dynamic logic be locally generated

Current sources

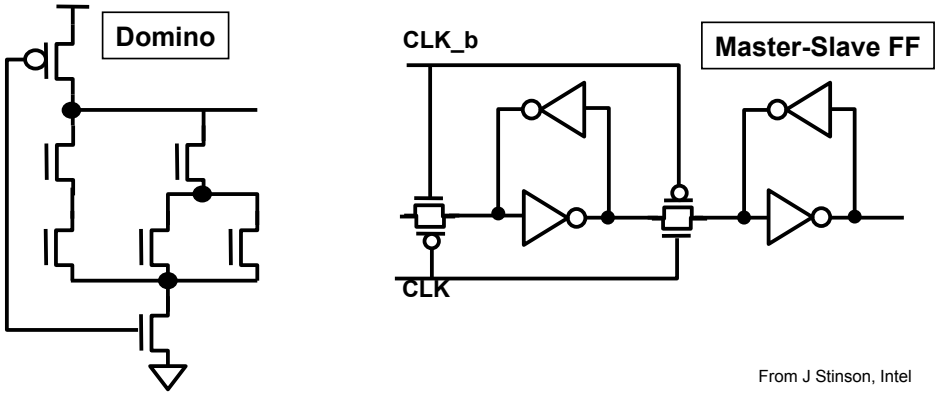
Ground drops area a real problem with making current sources:



- Current sources tend to be used in the analog section of the circuit
  These circuits use DC power
- Tends to want its own (special clean) supply so it does not use grid
  Its supply must be routed
- Gain of current sources tends to be high (30mV can be significant)
- Also watch for Vbb noise changing currents
  - If you need too distribute currents a far distance, pass a current and then go though a mirror

Charge-sharing

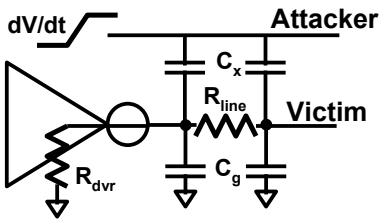
- Domino
  - Classic charge-sharing problem; fix with secondary precharge devices
- Pass-gate
  - Watch out for poorly driven nodes, memory cells, sequentials
- Static CMOS
  - Plain vanilla CMOS is still susceptible to charge-sharing!
  - Glitches can cause downstream failures



From J Stinson, Intel

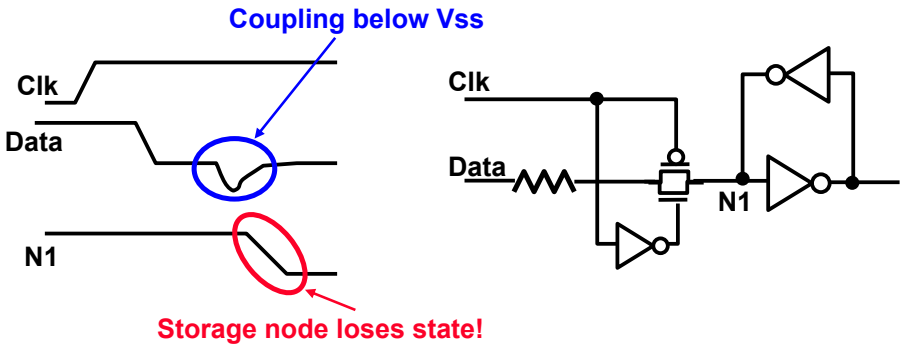
Capacitive wire coupling

- Can cause both functional and speed failures
- Signals can capacitively couple side-to-side and above/below
- Important elements
  - Victim driver strength (Rdrive and Rline)
  - Attacker slope (dV/dt)
  - Switching cap vs. Quiet cap
  - Attacker switching window relative to Victim



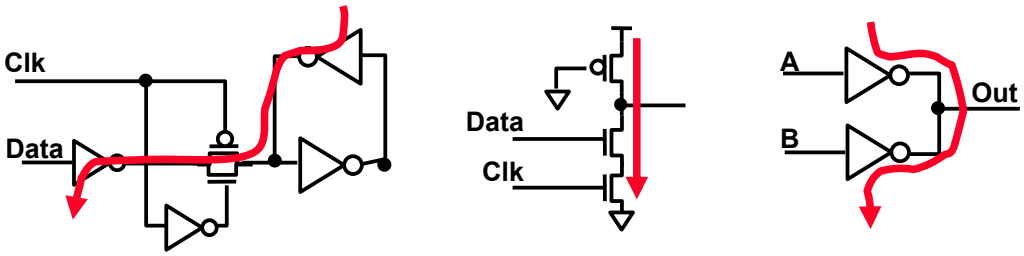
Wire coupling and passgates

- Typically only worry about coupling between Vdd and Vss
  - Noise induced causes downstream circuits to behave poorly
- Passgate inputs susceptible to coupling outside supply rails
  - Coupling above Vdd or below Vss can create Vgs > Vth



Contention circuits

- Rely on sizing and ratios to work correctly
  - Need account for process, voltage and temperature (PVT) variance
  - Correct path must always “win the fight” for correct operation
- Temporary contention circuits
  - Jam latches, memory writes
- Permanent contention circuits
  - Pseudo-NMOS, ratioed logic

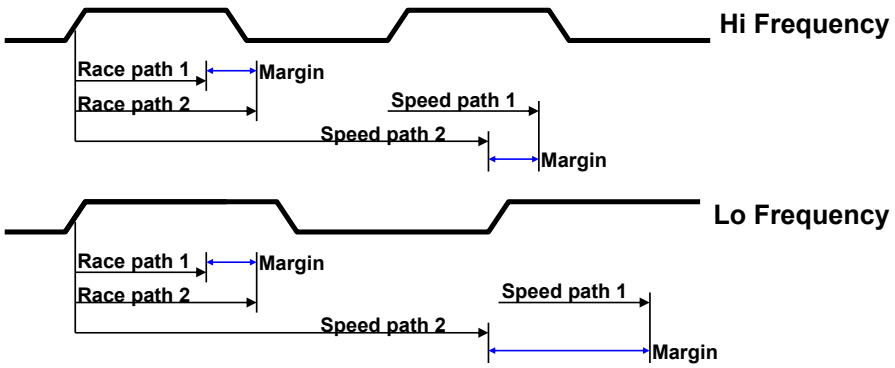


Leakage

- Becoming (has become) a major issue in design
  - Accounts for 10-20% of 130nm processor dissipated power
- Circuit operation needs to account for leakage
  - Storage nodes are especially susceptible to leakage
    - Domino outputs are treated as storage nodes
  - Sustainer sizes need to be large enough to fight leakage effects
    - Typically results in performance degradation due to contention fights
  - Increasing channel length reduces S-D leakage
    - Exponential reduction in leakage

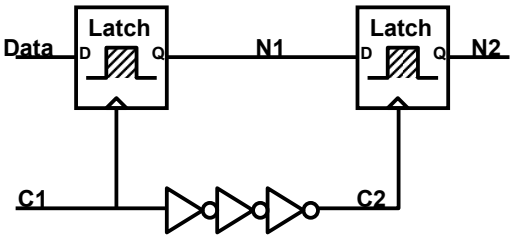
Races

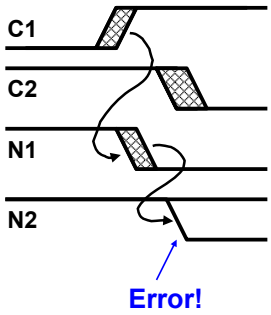
- Definition: signals starting from the same generation point, A, must arrive at a receiving point, B, in a specific temporal order
  - Generation point must be both geographically and temporally the same
    - Temporal component distinguishes a race from a speedpath
  - Need margin in each signal to ensure correct temporal arrival times



Functional race

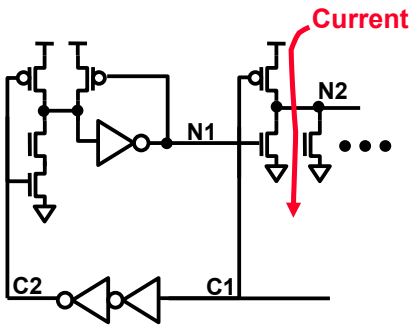
- Causes circuit to produce incorrect results
  - #1 concern with most race conditions
  - $T_{dmin} > T_{hold} + T_{skew} - T_{c\text{-}q}$
- Typically use very conservative estimate for T_{skew}
  - Assume both clock AND delay variance

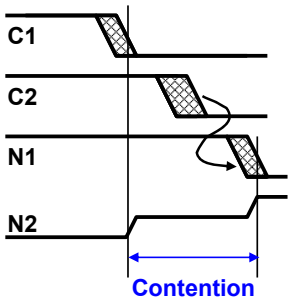




Power race

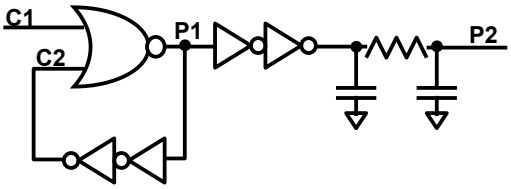
- Causes circuit to burn excessive power
  - Does not necessarily cause a functional or circuit failure
  - $T_{dmin} > T_{hold} + T_{skew} - T_{c\text{-}q}$
- Estimation of T_{skew} depends on design constraints
  - Power constrained designs need conservative estimates

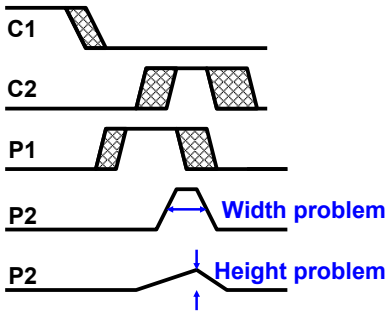




Pulse evaporation

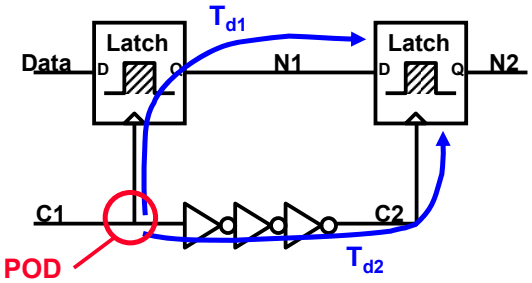
- Pulse generation is typically a race condition
  - Need to ensure that pulse is wide and tall enough to perform work
  - $T_{pw} > T_{work} + T_{skew}$
  - $T_{pw} > (T_{rise} / 2) + (T_{fall} / 2) + T_{skew}$, assuming T_{pw} is measured at Vdd/2
- Pulse evaporation usually results in a functional failure
  - Requires conservative pulse design
  - Usually need a pulse at least 3 FO4 to survive (4 is better)





Calculating race margins ($T_{skew\text{-}margin}$)

- Skew exists in both the clock path AND the data path
  - $T_{skew\text{-}margin}$ has to take both into account (be careful with T_{skew} terminology)
- Point-of-divergence (POD)
  - Calculate total loop length of racing signals
  - Take percentage of loop length as the $T_{skew\text{-}margin}$
- Statistical
  - More accurate, harder to calculate
  - Random variances add statistically ($T_{skew\text{-}margin}^2 = \sigma t_1^2 + \sigma t_2^2 + \sigma t_3^2 \ldots$)
  - Correlated variances add algebraically ($T_{skew\text{-}margin} = \sigma t_1 + \sigma t_2 + \sigma t_3 \ldots$)



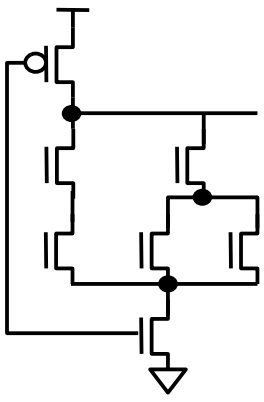
$T_{skew\text{-}margin} = (T_{d1} + T_{d2}) * \alpha$,
where α is usually 0.08-0.15

Why do chips fail?

- Overriding tool warnings or errors
- Poor accounting for parasitics
- Poor accounting for worst case stimulus
- Poor accounting for noise sources
- Poor accounting for variability in process, voltage, temperature
- Tool bugs
- Invalid assumptions

Simulating for success

- All noise sources accounted for
  - Include model for capacitive coupling
  - Accurate wire model for both attacker and receiver
  - Side-loads
  - Propagated noise from prior stage
  - Supply noise
- Correct simulation stimulus
  - Source worst case initial voltages
  - Active drivers for all switching signals
  - Active loads for circuit under test
- Simulate across worst corner(s)
  - Model changes in PVT



More on simulating for success

Good schematics go a long way to avoiding chip failures

Schematic “do”s

- Do make them “clear at a glance”
- Do err on the side of too much hierarchy
  - Rather than too complex schematics

Schematic don’ts

- Don’t connect nodes through labels. Draw the darn wire!
- Don’t use different names in layout vs Verilog vs schematic
- Don’t make icons overlap
- Don’t do this (next slide)

Bad example