Lecture 5

How to Optimize CMOS Circuit Delay and Power: Understanding Synthesis Output

Adapted from slides by Mark Horowitz
with significant contributions from Subhasish Mitra

Overview

- **Reading**

  W H 4.2, 4.3 skipping all optional sections

  + Harris Logic Effort

- **Introduction**

  Most of you will use synthesis to create the circuits you will use. The synthesis tool will optimize the logic and the gate sizes to met your power and performance goals. In order to understand the tool's output (and why it can't meet your constraints) you need to understand how to optimize the delay of a circuit. So the goal of this lecture is to provide this set of tools. To develop this set of tools we need to look at a very simple delay problem first, and that is the delay in a chain of inverters. Then we will take this result and look at a chain of logic gates. After looking at the pure delay optimization problem, we will look at power, and ask how the optimization changes when we consider power issues

We Want to Design at a Higher Level

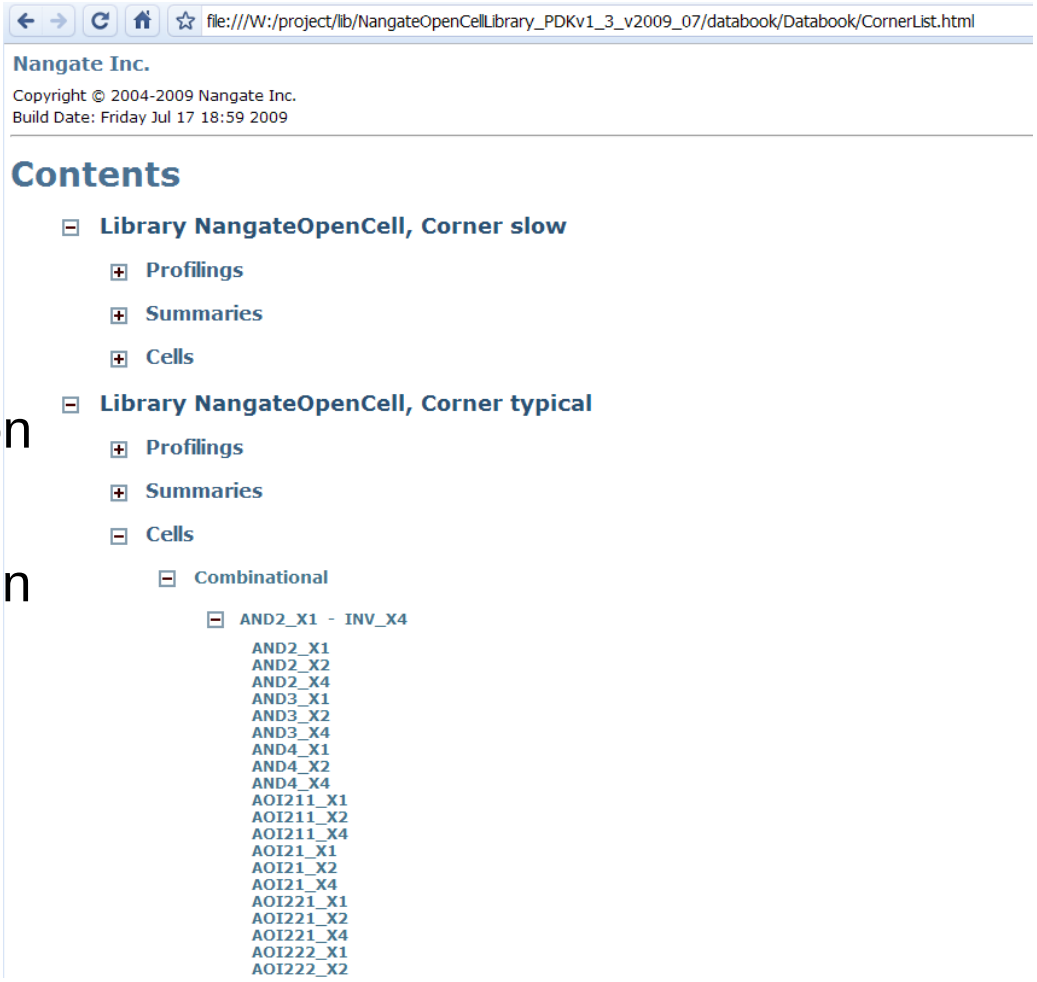
- Don’t want to spend our life creating layout
  - We will use a set of std (pre-made) cell
- Don’t want to spend our life creating schematics
  - Don’t want to construct each gate
- We want to use a higher-level language, Verilog
  - And System Verilog, and even C variants today
  - To more easily create our implementation
- And we will get to doing that next week

Checking Synthesis Output

- But then some tool is going to create the final implementation
- And you need to know when the tool is BSing you
  - Well maybe tools don't BS
  - But they do give you bogus output
    - You gave then an impossible circuit
    - You configured the tool wrong
    - The tool has a bug in it
- So even when you use synthesis
  - Need to know what the correct answer should look like
  - So you can rewrite your Verilog to fix the problem

Using Standard Cells – The Building Blocks

- 45nm cell library
- Notice lots of cells
  - Many the same function
- Must contain all information
  - For the tools to use



What Information Do We Need?

- Function
- Area
- Performance
- Power

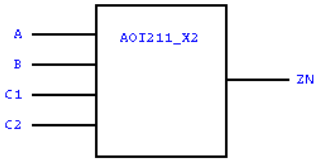
Data Sheet for AOI211

Datasheet for characterization corner: **NangateOpenCell_typical** , library **"NangateOpenCell"**. Data for cell **AOI211_X2** (Combinational) . Eventual numbers in *italic* show that the values have required interpolations due to internal data manipulation and/or non-matching LUT index templates. See notes for additional information.

Summary

General Cell Characteristics

Strength	2
Cell Area	1.330 um^2
Equation	ZN = "!(((C1 & C2) | B) | A)"
Type	Combinational
Input	A, B, C1, C2
Output	ZN



State Table				
A	**B**	**C1**	**C2**	**ZN**
H	X	X	X	L
X	H	X	X	L
X	X	H	H	L
L	L	X	L	H
L	L	L	X	H

Schematic

Capacitance

Capacitance [fF]	
A	1.11
B	1.10
C1	1.20
C2	1.21

- **NLDM Capacitance** [input capacitance table (capacitance, [fall|rise]_capacitance and [fall|rise]_capacitance_range attributes)]
- **Max Capacitance** [maximum allowed output capacitance]
- **CCS Receiver Capacitance** [arc and pin based receiver capacitance for CCS model]
- **ECSM Receiver Capacitance** [arc and pin based receiver capacitance for ECSM model]

Timing

Propagation Delay [ps]					
Input Transition [ps]		**7.50**		**600.00**	
Load Capacitance [fF]		**0.40**	**51.20**	**0.40**	**51.20**
A to ZN	**fall**	23.01	171.43	*59.20*	*376.89*
	rise	71.24	*553.36*	*169.94*	*725.82*
B to ZN	**fall**	21.30	*167.37*	37.82	*369.15*
	rise	64.53	*546.59*	*196.97*	*779.36*
C1 to ZN	**fall**	16.23	184.98	-11.84	373.61
	rise	31.73	514.04	180.30	774.81
C2 to ZN	**fall**	18.64	187.46	-3.64	343.18
	rise	40.83	523.25	207.75	785.99

Output Transition [ps]					
Input Transition [ps]		**7.50**		**600.00**	
Load Capacitance [fF]		**0.40**	**51.20**	**0.40**	**51.20**
A to ZN	**fall**	16.58	*142.21*	111.86	*249.75*
	rise	36.97	*471.10*	*93.80*	*483.78*
B to ZN	**fall**	13.95	*139.61*	108.50	*248.69*
	rise	36.97	*471.11*	*103.28*	*483.02*
C1 to ZN	**fall**	9.40	155.11	100.52	260.10
	rise	30.67	464.59	107.86	483.86
C2 to ZN	**fall**	9.40	155.10	72.16	210.88
	rise	36.50	470.93	101.72	488.21

- **Propagation Delay** [values from timing groups (cell_rise and cell_fall)]

Data Sheet, cont'd

Static Power Consumption

[leakage power data (leakage_power group and cell_leakage_power attribute)]

Leakage [nW]
17.95

When	Leakage [nW]
!A & !B & !C1 & !C2	10.73
!A & !B & !C1 & C2	23.87
!A & !B & C1 & !C2	11.08
!A & !B & C1 & C2	24.69
!A & B & !C1 & !C2	7.66
!A & B & !C1 & C2	16.69
!A & B & C1 & !C2	16.69
!A & B & C1 & C2	24.57
A & !B & !C1 & !C2	7.41
A & !B & !C1 & C2	16.50
A & !B & C1 & !C2	16.50
A & !B & C1 & C2	24.51
A & B & !C1 & !C2	12.47
A & B & !C1 & C2	21.59
A & B & C1 & !C2	21.58
A & B & C1 & C2	30.61

Dynamic Power Consumption

[values from internal_power groups in output pins (rise_power, fall_power a

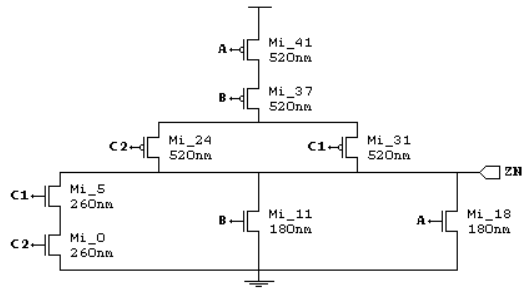
Dynamic Power Consumption (fall) for All Cells

Dynamic Power Consumption (rise) for All Cells

Load Capacitance [fF] / Input Transition [ps]			0.40	1.60	3.20	6.40	12.80	25.60	51.20
A to ZN	fall [uW/GHz]	7.50	3.07	3.07	3.08	3.08	3.10	3.12	3.15
		18.75	3.04	3.05	3.05	3.06	3.08	3.10	3.13
		37.50	3.06	3.06	3.06	3.07	3.08	3.09	3.12
		75.00	3.19	3.18	3.17	3.15	3.14	3.13	3.14
		150.00	3.68	3.63	3.57	3.50	3.41	3.33	3.26
		300.00	5.04	4.93	4.80	4.59	4.32	4.01	3.74
		600.00	8.20	8.04	7.87	7.47	6.87	6.10	5.27
	rise [uW/GHz]	7.50	4.81	4.83	4.85	4.86	4.88	4.91	4.94
		18.75	4.79	4.81	4.82	4.84	4.87	4.90	4.94
		37.50	4.78	4.79	4.81	4.83	4.85	4.88	4.93
		75.00	4.78	4.79	4.80	4.81	4.83	4.87	4.92
		150.00	4.87	4.87	4.86	4.86	4.86	4.88	4.92
		300.00	5.63	5.53	5.43	5.32	5.21	5.13	5.08
		600.00	8.40	8.12	7.83	7.39	6.88	6.40	6.01

* When="!B & !C1 & !C2"

ıematic



pacitance

NLDM Capacitance

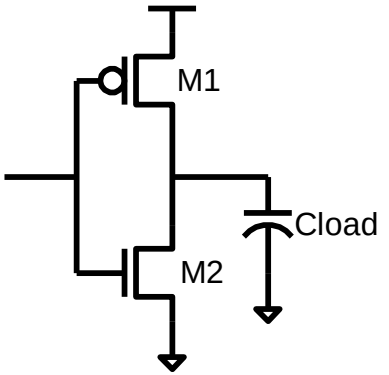
[input capacitance table (capacitance, [fall|rise]_capacitance and [f

te Capacitance for All Cells

Input	Cg [fF]	Fall			Rise		
		Typical	Min	Max	Typical	Min	Max
A	1.109	1.071	0.928	1.241	1.146	1.061	1.352
B	1.098	1.009	0.908	1.177	1.188	1.070	1.420
C1	1.196	1.077	1.013	1.344	1.315	1.147	1.693
C2	1.206	1.059	1.012	1.266	1.353	1.175	1.663

Inverter Delay

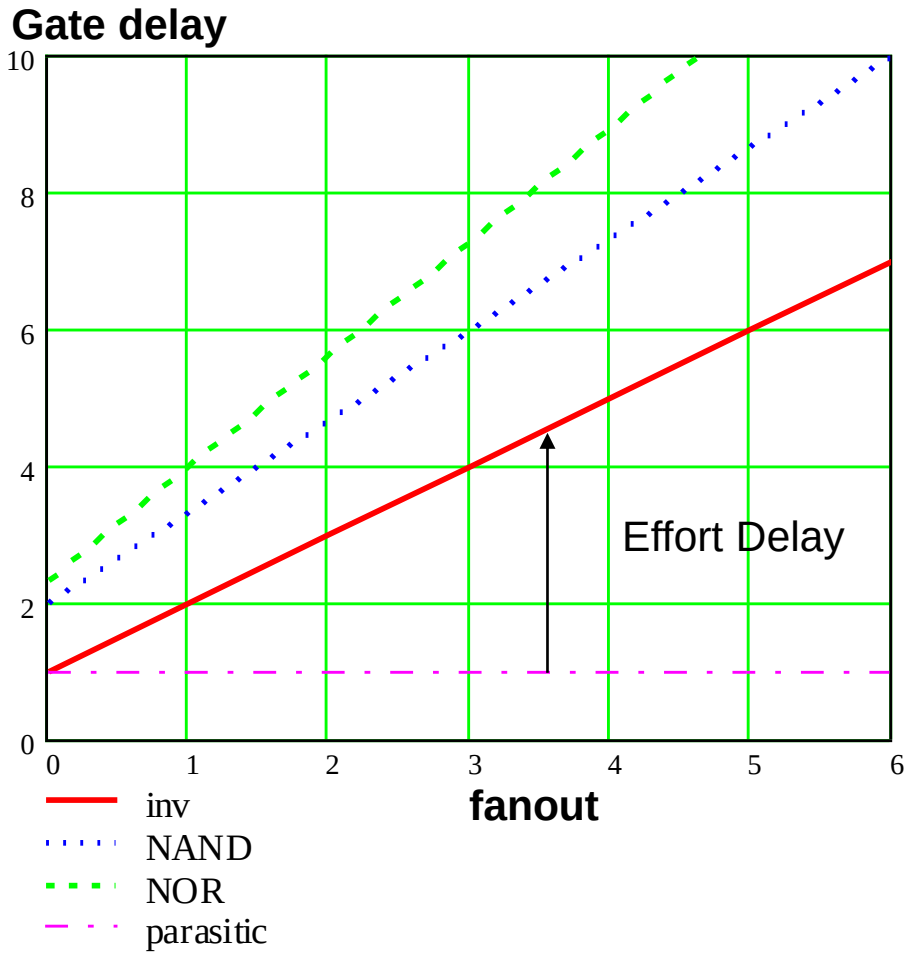
- Need a model for gate delay
  - Assume gate is sized for roughly equal rise and fall delays
    - $W_{M1} = 2W_{M2}$
  - RC model for gate delay



$$R_{drive}\cdot\left(C_{par}+C_{load}\right)\ \text{substitute}\ ,R_{drive}=R_{sq}\cdot\frac{L}{W}\ \rightarrow R_{sq}\cdot\frac{L}{W}\cdot\left(C_{par}+C_{load}\right)$$

$$R_{sq}\cdot\frac{L}{W}\cdot\left(C_{par}+C_{load}\right)\ \Big|\ \begin{array}{l}\text{substitute}\ ,W=\dfrac{C_{in}}{3\cdot C_g}\\ \text{substitute}\ ,C_{par}=\gamma\cdot C_{in}\\ \text{collect}\ ,C_{in}\end{array}\ \rightarrow 3\cdot R_{sq}\cdot L\cdot C_g\cdot\gamma+3\cdot R_{sq}\cdot L\cdot C_g\cdot\frac{C_{load}}{C_{in}}$$

Characterizing Gate Delay



- Graphical model
  - Each gate type is different
  - Y intercept is parasitic delay
  - Slope is called Logical Effort
- More complex gates
  - Have larger LE
  - Have larger parasitics
- Logical Effort
  - $\frac{Cin_{gate} * Rdrive_{gate}}{Cin_{inv} * Rdrive_{inv}}$

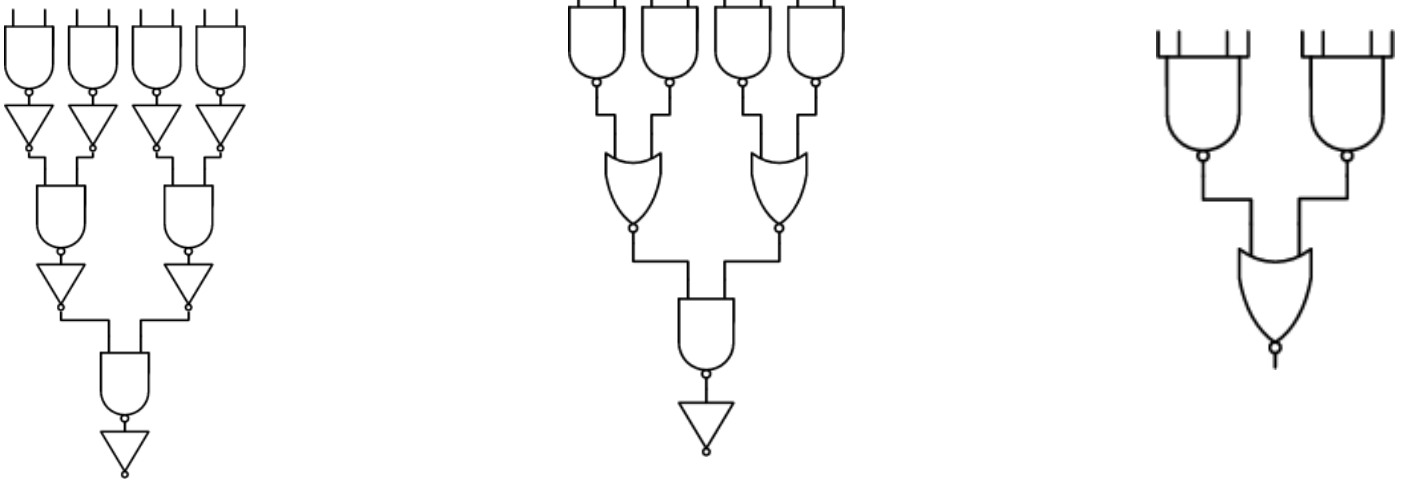
Simplifying Approximations for Hand Analysis

- Assume input capacitance is constant
  - Is a small approximation
- Assume that the output delay is linear on load
  - Another good assumption
  - Use RC model to get numbers
    - Transistors are not linear resistors
    - So the slope of delay can be off by a little bit
- Ignore input slope effects
  - Not as good, but ok for well sized circuits

What's Hard About Performance Optimization

- Didn't we cover the delay stuff last week:
  - Gate Delays are RC
  - Circuit delay is the "critical path" through logic

- Sizing gates can be tricky, even if the critical path is obvious:



- How to size these gates to minimize delay?

How to Size Gates for Minimum Delay

- What is hard about this problem?

- What constraints do you need to define problem?
  - To avoid trivial solutions

- Any ideas on how to start this problem
  - How to frame what we need to do?
  - Rules we can use to check our solutions
  - Insights about what the solution should look like

Root of the Problem

Needed Constraints

Approaches

Delay Optimization Problem #1

- You are given:
  - A fixed number of inverters
  - The size of the first inverter
  - The size of the load that needs to be driven
- Your goal:
  - Minimize the delay of the inverter chain
- You need two tools to solve this problem
  - An equation for how the inverter delay changes with size
    - Given in an earlier slide
  - A little calculus
- Notation:
  - Parasitic capacitance of an inverter = $\gamma \times$ Gate capacitance of that inverter (Cin of that inverter) where γ is a constant

A Little Math

- Total delay is the sum of the delays of all the inverters
  - Calling $\tau_{inv} := 3 \cdot R_{sq} \cdot L \cdot C_g \qquad \tau_{inv} = 2.16 \times 10^{-12}\,s$ (for 45nm)
  - Total delay = $\sum_{j=1}^{N} \tau_{inv} \cdot \left(\frac{C_{in_{j+1}}}{C_{in_j}} + \gamma \right)$
  - Where Cin_{N+1} is defined to be Cload
- The size of each inverter, 'j' affects two terms in the delay equations
  - The delay of gate j-1 (sets its load)
  - The delay of gate j (sets its strength)

At the Optimal Size

- Change in delay with respect to any Cinj should be zero

$$\frac{d}{dC_{in_j}}\text{Delay} = \tau_{inv}\cdot\left(\frac{1}{C_{in_{j-1}}}\right) - \tau_{inv}\cdot\left[\frac{C_{in_{j+1}}}{\left(C_{in_j}\right)^2}\right]$$

  - Remember Cinj proportional to Wj,
    - So this equation is true for Wj as well
- In the optimal case,
  - Wj is the geometric mean of Wj-1 and Wj+1
- For a chain
  - Forces all the inverters to have the same fanout $\left(=\frac{W_{j+1}}{W_j}\right)$
  - Fanout per stage = (Cload/Cin)^(1/N)
    - Cin is the gate capacitance of the first inverter in the chain
    - The one furthest from the output

Set the derivative equal to zero and solve. Get $C_j=\text{sqrt}(C_{j+1} * C_{j-1})$

Notice: Answer Does Not Depend on γ

- Why is this clearly the right result?

Delay Optimization Problem #2

- You are given:
  - The size of the first inverter
  - The size of the load that needs to be driven
- Your goal:
  - Minimize the delay of the inverter chain by
    - Finding the optimal number of gates
    - The sizes of the gates
- For a chain of length N, the fanout of each gate in the chain must match (from our first problem). Thus we want to find N that minimizes the following equation:

$$Delay = N \cdot \tau_{inv} \cdot \left[\left(\frac{C_{load}}{C_{in}} \right)^{\frac{1}{N}} + \gamma \right]$$

Number of Gates

- Can rewrite in terms of fanout, f

$$\frac{C_{load}}{C_{in}} = f^N \text{ solve}, N \rightarrow \frac{\ln\left(\frac{C_{load}}{C_{in}}\right)}{\ln(f)}$$

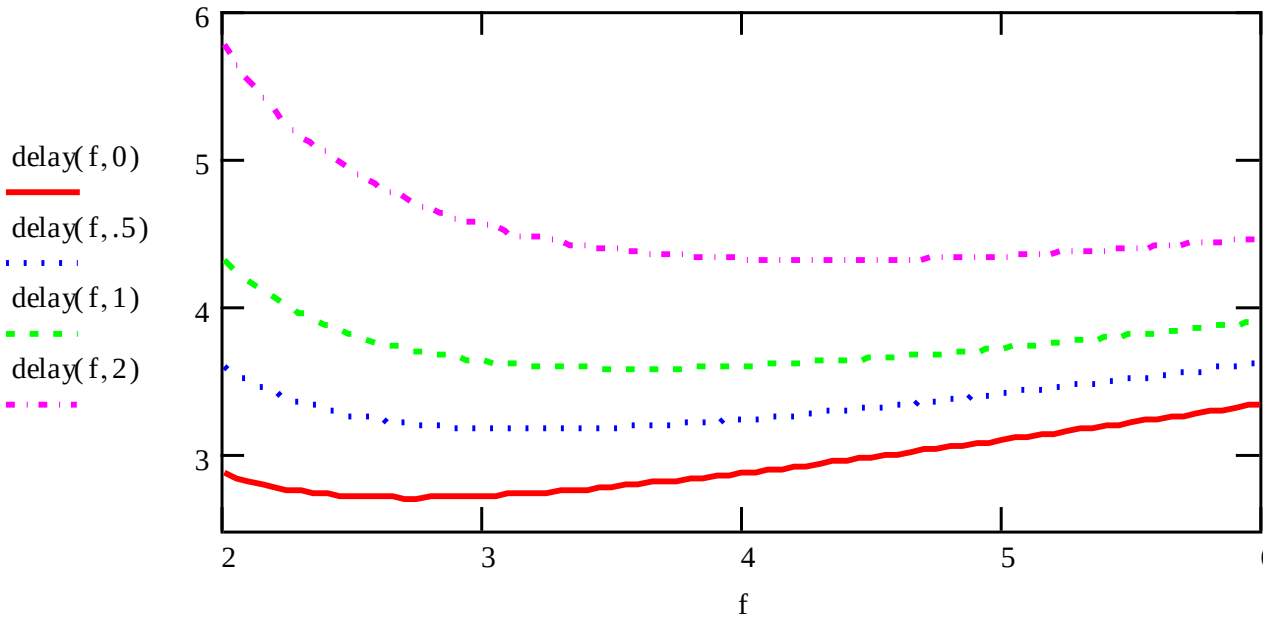
- Total chain delay is

$$\frac{\ln\left(\frac{C_{load}}{C_{in}}\right)}{\ln(f)} \cdot \tau_{inv} \cdot (f + \gamma)$$

- Can plot this delay as a function of f
  - Factor out ln(Cout/Cin) and τ_{inv}, since both are not a function of the fanout or the number of stages

Plot of Total Delay

$f := 2, 2.05 .. 6$ $\quad$ $delay(f, \gamma) := \frac{(f + \gamma)}{\ln(f)}$



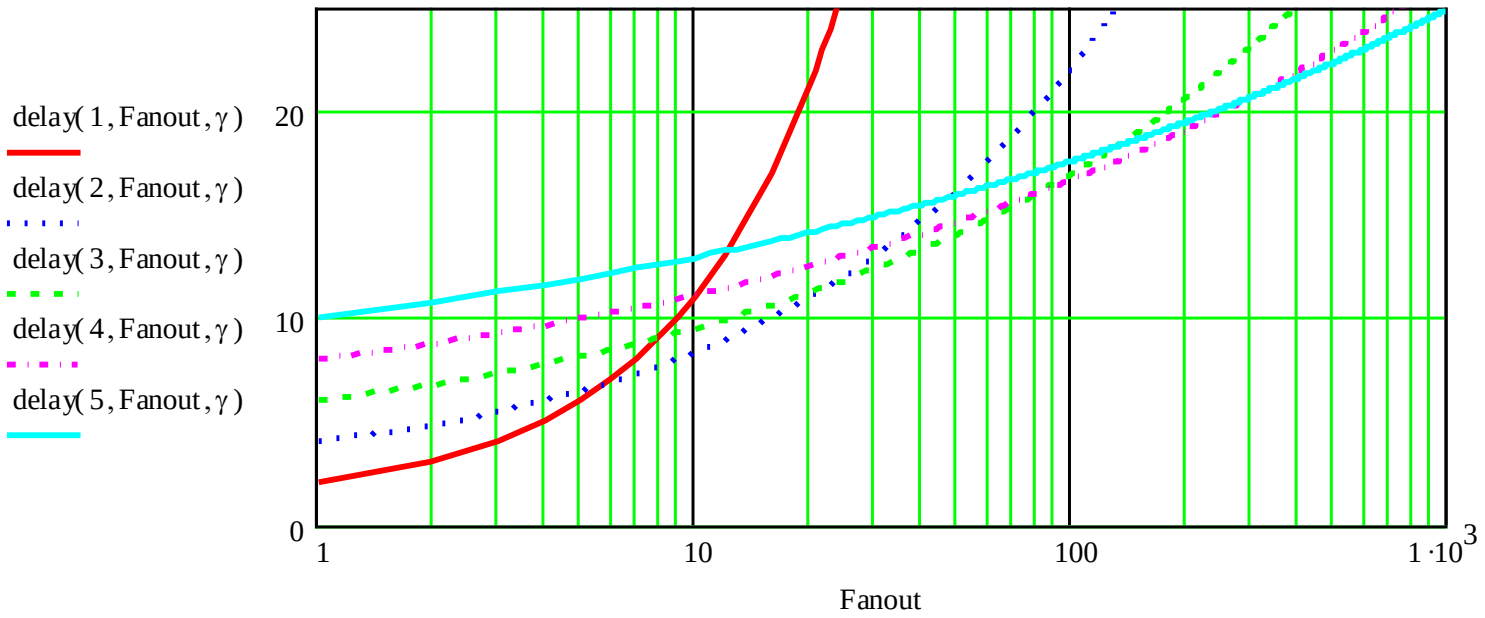
- Total delay vs. fanout for different values of gamma
  - Curves are very flat → can choose f around 4

Intuitive Explanation

- If you don't like math …
- A gate's delay consists of two terms
  - Parasitic delay (delay intrinsic to gate)
    - Independent of load
  - Effort delay
    - Proportional to Fanout of gate
- If you add more stages – have N stages
  - Get more parasitic delay (proportional to N)
  - But the fanout delay can decrease (if Fanout is large; > e)
    - N * Fanout $^{1/N}$
- Optimal number of stages balances these factors
  - Depends of the relative size of the parasitic effort delays

Delay vs. Fanout for Different Number of Stages

$$delay(N, Fanout, \gamma) := N \cdot \left(Fanout^{\frac{1}{N}} + \gamma \right) \qquad Fanout := 1, 2 .. 1000 \qquad \gamma := 1$$



- Plots delay of different length inverter chain vs. total fanout of the chain. Breaks come at around 6, 22, 75, 240

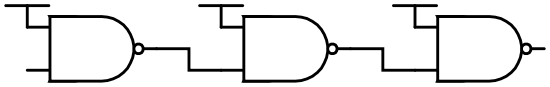
Logic Optimization Problem

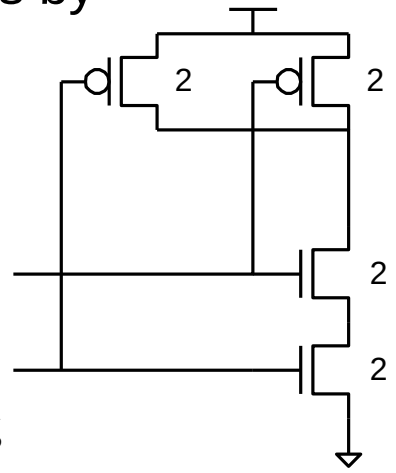
- Minimize delay
  - While performing needed logic function
  - Building blocks are quite flexible
    - NAND, NOR gates of any number of inputs
    - AOI and OAI gates too
  - Need to choose the “right” number of stages
  - And the function at each stage.

- Remember minimizing # stage might not be optimal
  - Could build any logic in one stage
  - Makes design “interesting”
    - But that is what computers are for

Delay Optimization Problem #3

- You are given:
  - The size of the first NAND gate
  - The size of the load that needs to be driven
  - For simplicity, assume Cpar occurs only at the output (there is a slightly complex issue depending on which input arrives first)
- Your goal:
  - Minimize the delay of a chain of NAND gates by
    - Finding the optimal number of gates
    - The sizes of the gates





$$\text{Delay} = \tau_{\text{NAND}} \cdot \left(\frac{C_{\text{load}}}{C_{\text{in}}} + \gamma \right)$$

This is the delay of a single gate, where Cload is the local load on this gate

$$\tau_{\text{NAND}} = 4 \cdot R_{\text{sq}} \cdot L \cdot C_g$$

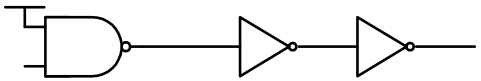
Delay for NAND Gates

- Can work out the delay of this string of gates
  - Formula looks the same as the formula for inverters
  - All that is different is the time constant for the gate
  - Optimization will yield the same results
    - Equalize the delay of each stage
    - Optimal fanout will be around 4

- But this is the wrong problem to solve
  - (That is the reason the answer is so boring)
  - Assumed that we only had NAND gates
  - Really have inverters too, and we could use them (as a buffer) to drive the large load. We don't need to use only NAND gates. Always use the fastest gate as a buffer.

Delay Optimization Problem #4

- You are given:
  - The size of the first gate
  - The size of the load that needs to be driven
- Your goal:
  - Minimize the delay of a chain of different kinds of gates by:
    - Adding the optimal number of inverters
    - Sizing the transistors in each of the gates



- What is new:
  - The τ and the γ for each gate are no-longer the same

Optimizing Delay

- Same basic rule as in the inverter case
  - Need to match the fanout part of the delay
    - If the NAND gate is gate j-1, and the inverter is gate j

$$\tau_{NAND} \cdot \frac{C_{in_j}}{C_{in_{j-1}}} = \tau_{inv} \cdot \left(\frac{C_{in_{j+1}}}{C_{in_j}} \right)$$

    - The fanout of each gate is no longer the same
  - Note this does not match the total delay
    - Parasitic delay of Inv and NAND might not match

$$Delay = \tau_{NAND} \cdot \left(\frac{C_{in_j}}{C_{in_{j-1}}} + \gamma_{NAND} \right) + \tau_{inv} \cdot \left(\frac{C_{in_{j+1}}}{C_{in_j}} + \gamma_{inv} \right)$$

Converting Arbitrary Logic to Inverter Sizing

- This is the same as the inverter problem
  - Except the delay slope with fanout is different
- We defined logical effort
  - As the ratio of the gate delay slope to that of an inverter
- So if we define effective fanout:
  - LE * Electrical fanout
- And solve the “inverter” problem with effective fanout
  - I can optimally size arbitrary logic!

Logical Effort Formalism

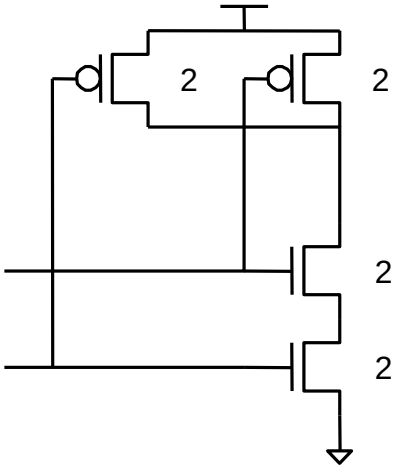
- Delay of a gate g: τ_g * (γ_g+ Cload / Cin)
- Let us express delays in terms of τ_{inv} i.e., Delay* = Delay/ τ_{inv} (helps us to "hide" tech. parameters in τ_{inv})
- Delay of logic gate has two components:
  - Delay* = Parasitic Delay + Effort Delay (d* = p + f)
- Effort delay again has two components:
  - Effort Delay = Logical Effort * Electrical Effort (f = g*h)
- Electrical Effort : h = Cload / Cin

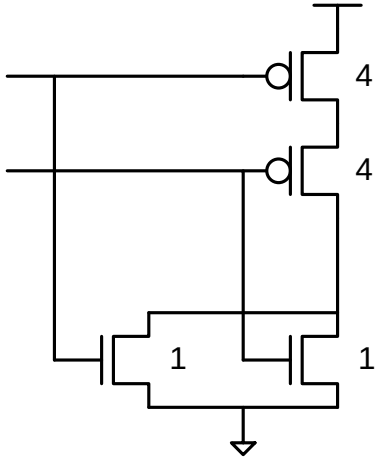
$$Logical\ Effort = \frac{\tau_{gate}}{\tau_{inv}} = \frac{Cin_{gate} \cdot Rdrive_{gate}}{Cin_{inv} \cdot Rdrive_{inv}}$$

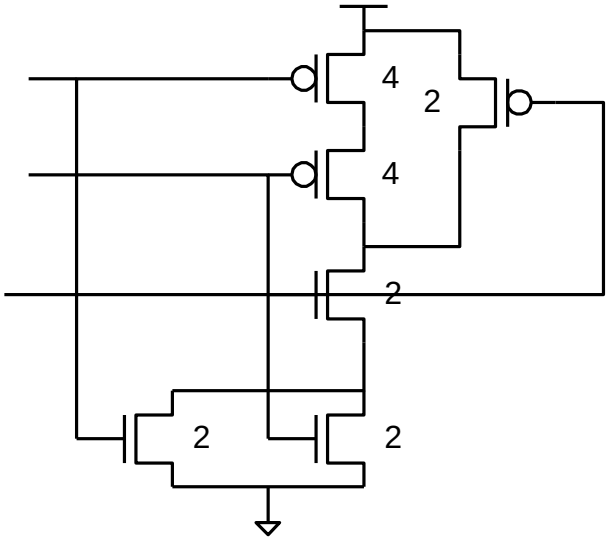
Watch out it's Cin (and NOT Cout) in the expression!

Calculating Logical Effort for a Gate

- Build the gates to have the same drive strength as a 2x pMOS, 1x nMOS inverter. The numbers on each transistor is relative to the 1x nMOS transistor in the inverter. The Cin of inverter is 3x.







- LE = 4/3 LE=5/3 LE=2; 4/3
- Note that the logical effort of all inputs does not always match

Optimizing Delay For Many Gates

Delay = $\tau_1 * (\gamma_1 + C2 / Cin) + \tau_2 * (\gamma_2 + C3 / C2) + \tau_3 * (\gamma_3 + C4 / C3) + \ldots$

- Or using Logical Effort
  - Delay* = $LE_1(\gamma_1 + C2 / Cin) + LE_2(\gamma_2 + C3 / C2) + LE_3(\gamma_3 + C4 / C3)$
- At optimal
  - $LE_1 C2 / Cin = LE_2 C3 / C2 = LE_3 C4 / C3 = \ldots$
  - So each term must equal $(LE_1 * LE_2 * LE_3 \ldots * LE_n \, Cout / Cin)^{1/n}$

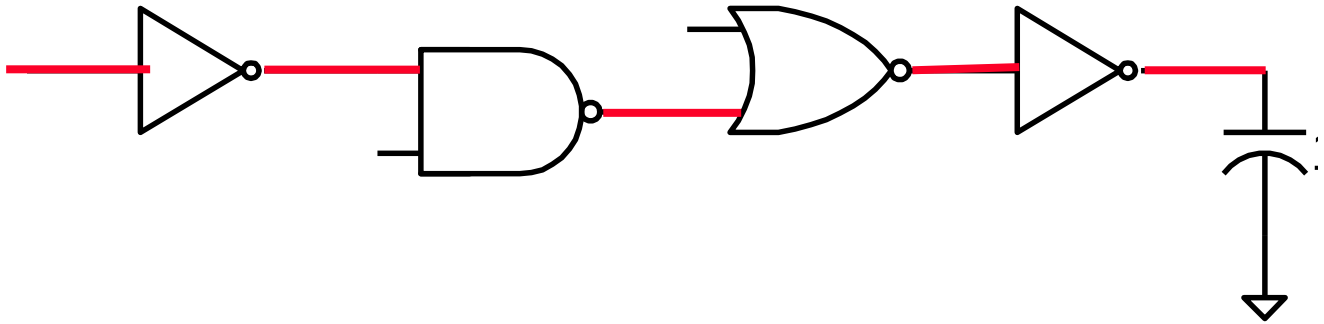
Chicken and Egg Problem

- To know total effective fanout need to know LE of each gate
  - But to know the LE, we need to know what gates we are using
  - And that means we need to know the number of stages

- Solution
  - Just guess
    - Turns out total LE is not a strong function of how it is implemented
  - With a starting point figure out about the right number of stages
    - Good heuristic for determining the number of stages:
    - Keep effective fanout per stage around 4.
  - Then change the design to use this number of stages
    - And recalculate the sizing

Gate Sizing Problem

- Suppose we want to size the gates on a given circuit path
- We will use the LE of the gate to help find the correct sizes
  - Know that the LE*Fanout for each gate should be the same
    - We will try to make this value close to 4



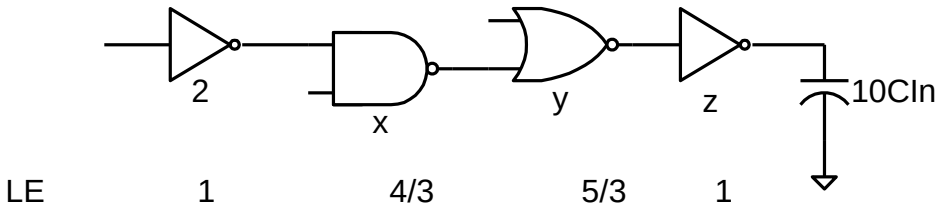
A Convention

- Before we show the math need to set a convention:
  - What does a gate of size ‘2’ mean?
  - For a gate, you have two options:
    - Can define it to mean it has twice the capacitance of an inverter

      OR
    - Can define it to mean it has ½ the resistance

- **In my notes the size is a measure of the input capacitance**
  - **A size 2 gate has twice the input capacitance of an inv (N = 1, P = 2)**

Using Logical Effort to Size Gates



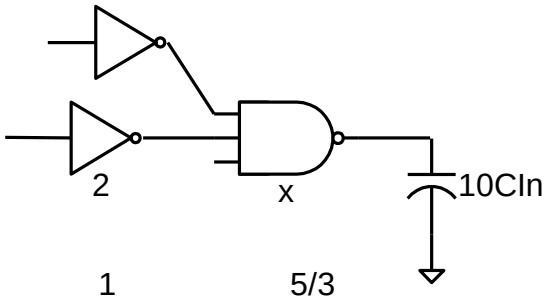
LE	1	4/3	5/3	1
Fanout	x/2	y/x	z/y	10/z

- Overall Fanout = Cout / 2*CIn = 5 (called path electrical effort in W&H)
- Multiply all LE's = 1*(4/3)*(5/3) = 2.2 = Path LE
- Path LE * Overall fanout = 2.2 * 5 = 11 = Effective Fanout (EF)

- We know that LE * Fanout of every stage must be equal to:
  - $(EF)^{1/n}$ assuming there are n logic stages
  - EF = 11, n = 4, LE*Fanout of each stage = $(11)^{1/4}$ = 1.8
    - Too small, remember we want to keep it close to 4
- Now you can calculate fanout of each stage using its LE

Reducing Number of Logic Stages

- Reformulate logic
  - If you have too low EF, use more complex gates, with fewer stages
  - Often need to use AND-OR-Invert gates or OR-AND-Invert gates
  - EF = 8.33, with two stages
  - EF per stage is 2.9 which is quite reasonable
    - X = 2*2.9 = 5.8



LE	1	5/3
Fanout	x/2	10/x

Note: this was the wrong optimization if the top input was critical. Since we have slowed down this path. Make sure you are sizing the critical path.

Logical Effort Summary

- Estimate the path effort
  - EF = Π LE * Cload/Cin
- Estimate the optimal number of stages
  - $N^* = \log_4(EF)$
- Estimate the minimum delay
  - 4N* + Parasitic delay
- Determine the actual number and type of gates
  - Fit the required logic in N stages, where N is close to N*
  - This may change slightly the path EF
- Determine the stage effort
  - New $EF^{1/N} = f$
- Working from either end, determine gate sizes
  - Cin = Cout*LE/f

Some Definitions

Term	Stage expression	Path expression
Logical effort	g (seeTable 1)	$G = \prod g_i$
Electrical effort	$h = \frac{C_{out}}{C_{in}}$	$H = \frac{C_{out\ (path)}}{C_{in\ (path)}}$
Branching effort	n/a	$B = \prod b_i$
Effort	$f = gh$	$F = GBH$
Effort delay	f	$D_F = \sum f_i$
Number of stages	1	N
Parasitic delay	p (seeTable 2)	$P = \sum p_i$
Delay	$d = f + p$	$D = D_F + P$

Analyzing Synthesis Results

- In most situations, you won't be sizing gates
  - A program (synthesis) will do that for you
- So how do I know if the synthesis is doing a good job
  - Or more likely if I need to change my Verilog code
  - Or my synthesis setup script
- Look at the timing report
  - Look at the delay of each stage (and the number of stages)
  - If many stage delays are under FO4 delay
    - You have too many stages
  - If many stage delays are much larger than FO4 delay
    - You have too few stages
    - There were wire caps the synthesis tool did not know about

Ok, But What About Power?

- The previous derivation was to optimize timing
  - We did not consider energy
  - What should the rules be for a energy constrained design?

- Are min delay results useful at all for these systems?
  - What changes?

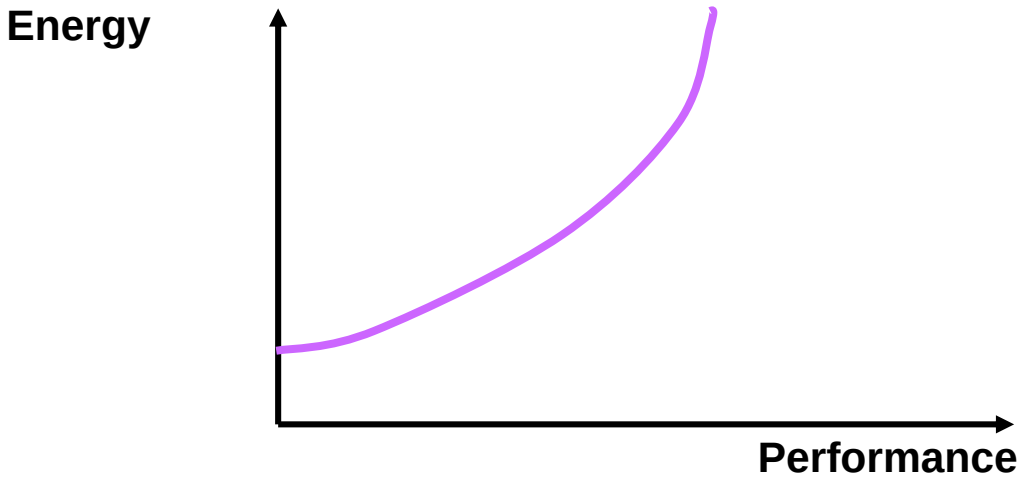
Constrained Optimization

- The key is to think about marginal cost
  - If your resources are limited
  - Want the most reward for each dollar you spend
  - You want the highest margin return for the resource

- If you have a lot of resources
  - You would expect the return on the later dollars is lower
  - Law of diminishing returns
    - As you spend more resources, margin costs go up

What Does This Have to Do With VLSI?

- Power and performance are related to each other
  - Follow a law of diminishing returns too



- There is some minimum energy to enter the game
- There is some maximum performance possible
  - Approaching either has very high marginal costs
    - Costs lots of performance to get to min energy, and …

Min Delay Points Are Very Inefficient for Power

- Do we want to get to min delay point?
  - Min delay means no change in W improves performance
    - Sound great
  - But this also means the change in delay with W = 0

- Changing W changes the capacitance
  - Which changes the power of the circuit
  - So the minimum delay point means that the marginal change in delay for a change in energy is 0, or we were willing to pay any amount of energy to improve performance.

We are throwing power away

A Better Optimization

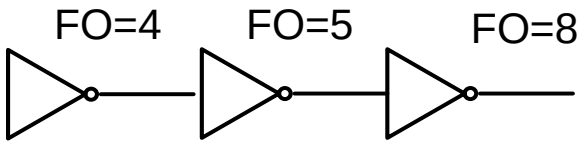
- Make the delay minimization a constrained minimization
  - Set a energy limit per transition
  - For sizing, this corresponds to a limit on total transistor Wt
    - Wt = Sum(Wi)
  - Solve (using Lagrange multipliers, etc.)
- More intuition (for me) is gained by looking at marginal cost

$$-\frac{\partial E_{Sw} / \partial W_i}{\partial D / \partial W_i} = \frac{ec_i}{\tau_{nom} \cdot \left(h_{eff,i} - h_{eff,i-1} \right)}$$

- ec_i = Energy consumed by gate "i"
  - To match marginal costs, higher power gates (larger ec_i) must run at higher fanout than optimal (all fanouts the same)

Power Optimal Buffer Chain

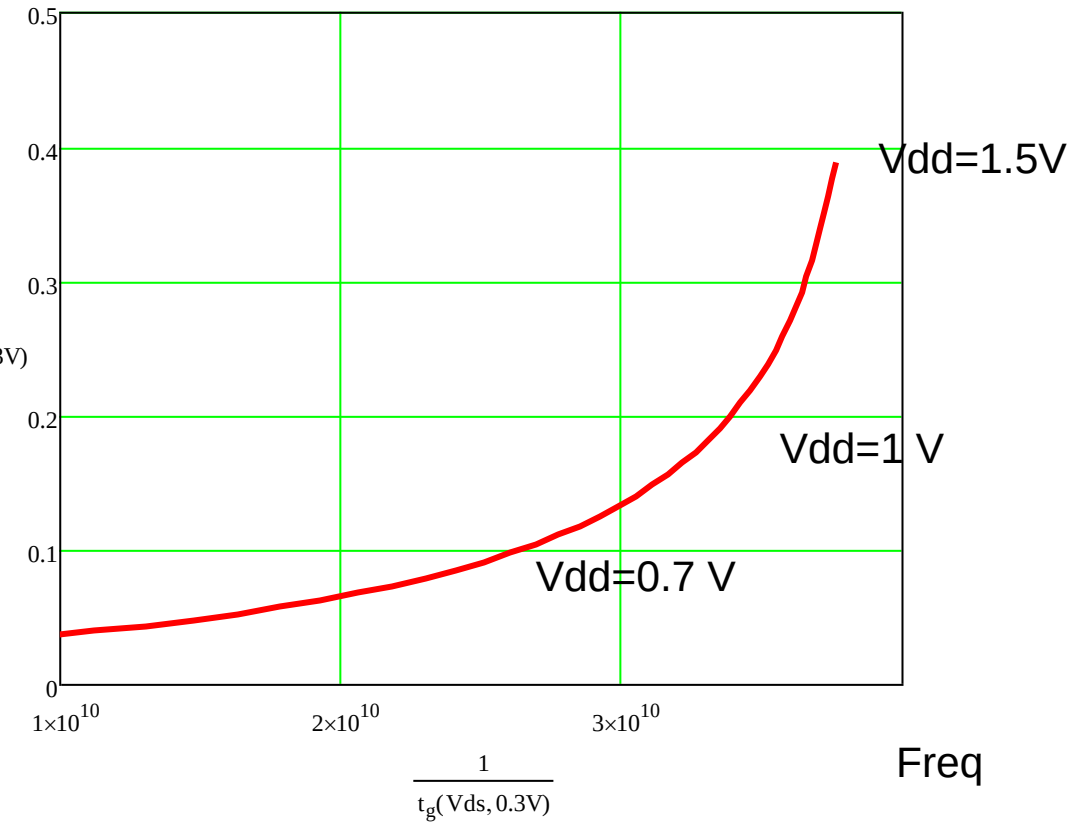
- The effective fanout is not the same
  - Last buffer has higher energy, so higher fanout



- Notice that the change in fanout depends on gate energy
  - In an buffer chain this decreases exponentially
    - As you move away from the output
  - Which means a buffer or 2 away from output
    - Energy/stage is small compared to output
    - So use min delay solution!
  - Real logic is more complex, but same principle applies
    - Increase fanout on high energy gates

Changing Vdd to Save Energy

- Vdd also effects power and performance
  - Energy = αCV^2F
  - Rsq = f(Vdd) too
- Plot of energy/switch
  - Vs. 1/delay
- Need to match
  - Vdd marginal cost
  - Sizing marginal cost



+ What Happens With Fixed Wire Capacitance?

- Suppose 4 inverters: Inv1, In2, Inv3, Inv4 with Inv4 driving Cload
  - We can't neglect wire cap Cw at Inv3 output
- If Inv1 size is not fixed
  - Make all the gates large, so Cw is small compared Cin4
  - But this is very energy inefficient
- If Inv1 is fixed at Cin
- The delay equations becomes:

  - $D^* = Cin2/Cin + \gamma + Cin3/Cin2 + \gamma + (Cin4+Cw)/Cin3 * \gamma + Cload/Cin4 + \gamma$

  When you differentiate with respect to Cin3, will get another term

+ Wire Loads Continued

- $(Cin2)^2 = Cin1 * Cin3$
- $(Cin3)^2 = Cin2 * (Cin4 + Cw)$
- $(Cin4)^2 = Cin3 * (Cload)$

$Cin2 / Cin1 = Cin3 / Cin2 = (Cin4 + Cw) / Cin3$

$Cin4 / Cin3 = Cload / Cin4$

- Notice that the fanout goes DOWN after the wire load
  - This minimizes the delay but is energy inefficient
  - Mostly don't run min delay solutions with wires