Optimizing Delay

- Optimizing delay can be broken into two categories
  - Gate Size selection
  - Transistor sizing
- Gate size selection is done in a standard cell design approach in which you have a library that offers multiple drive strength cells and pick the cells sizes that give the highest speed for a design
  - Current synthesis tools do a good job
- Transistor sizing is done in a custom design in which you size individual transistors during the design process to optimize delay
  - quality depends on individual designer
  - some synthesis help available
  - simulation iteration a tempting option but can be time consuming

Gate Size Selection

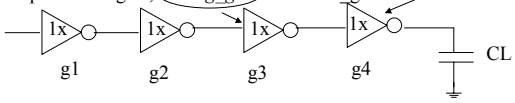
- Many algorithms for gate size selection exist
- One iterative approach is known as the *Tilos* algorithm

Assumptions:
1. Can compute the delay along a path of gates
2. Have multiple gate sizes to choose from

Will yield good results for a path delay

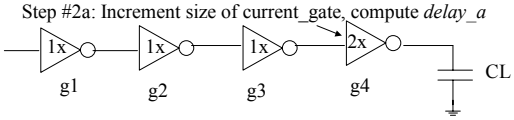
Tilos Algorithm

Step #1: Start with Minimum gate sizes, set *current_gate* equal to last gate, *driving_gate* to current_gate –1.



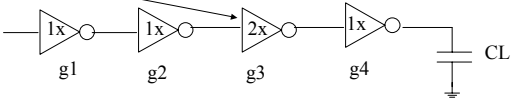
Measure delay, call this *last_delay* .

Step #2a: Increment size of current_gate, compute *delay_a*



Tilos Algorithm (cont.)

Step #2b: Restore current_gate size. Increment size of driving_gate, compute *delay_b*



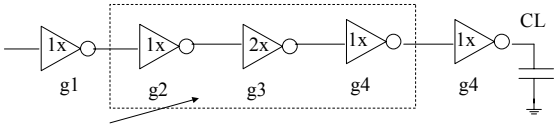
Step #3: Compare delays A, B against *last_delay*. Whichever shows the greatest improvement, use this new gate configuration and set *last_delay* equal to the new delay.
Repeat Steps #2, #3 until no further delay improvement.

Set *current_gate* to *driving_gate*, *driving_gate* to *current_gate-1* and repeat until all gates sized (an exception: the first gate size is considered a FIXED size as in an input buffer).

Some Observations

To save execution time, do have to compute entire path delay.

Computing changes in delay in a 'window' around sized-gate



Compute delay changes here

Also, gate sizes do not have to be exact to get near optimum delay. If optimum gate size happens to be 2.5x, a choice of 2X or 3X will yield good results. This means that rough estimation of gate sizes or transistor sizes can often be satisfactory.

Rules of Thumb

- Keep fan-in low to keep #transistors in series low (for sub-micron, often <= 3).
- Keep fan-out < 5
- Along a critical path, the minimum delay is achieved if each stage delay is about equal
- Keep rise/fall times about equal

Estimating Gate Delay, Transistor sizing

- Would be nice to have a "back of the envelope" method of sizing gates/transistors that would be easy to use and would yield reasonable results
- Sutherland/Sproull/Harris book "Logic Effort: Designing Fast CMOS Circuits" introduces a method called "Logical Effort"
- Chapter 1 of the book is posted on the Morgan-Kaufman website (www.mkp.com, search for author names)
  - Download this chapter, READ IT!
- We will attempt to apply this method during the semester to the circuits that we will look at.
- Will look at static CMOS application first (these notes taken from that chapter).

Gate Delay Model

Delay will always be normalized to dimensionless units to isolate effects of fabrication process

$$d_{abs} = d * \tau$$

Where τ (Tau) is the delay of a minimum sized inverter driving an identical inverter with *no parasitics*. Tau is NOT the no-load delay of an inverter. Also, it is not the delay of a 1x inverter driving a 1x inverter since this includes the delay contributions due to parasitics.

Delay of a logic gate is composed of the delay due to *parasitic delay p* (no load delay) and the delay due to load (*effort delay* or *stage effort f*)

$$d = f + p$$

Logical effort, Electrical Effort

The *stage effort f* (delay due to load) can be expressed as a product of two terms:

$$f = g * h$$

So delay is

$$d_{abs} = (f + p) * \tau$$
$$= (g*h + p) * \tau$$

g captures properties of the logic gate and is called the *logical effort.*
h captures properties of the load and is called the *electrical effort.*

RC model versus Logical Effort Model

On the surface, this does not look different from the model discussed earlier:

Logical Effort:

$$d_{abs} = (g*h + p) * \tau$$

Previous RC model

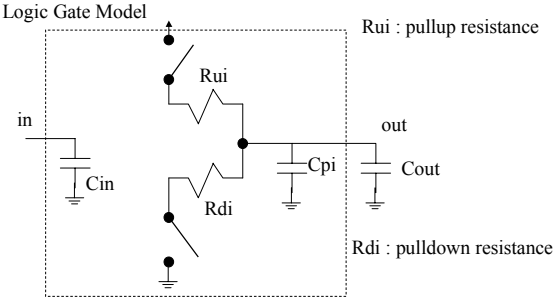
Gate delay = K * Cload + no-load delay

Where K represented the pullup/pulldown strength of the PMOS/NMOS tree.

It would help to see how the RC model can be used to derive the logical effort model.

Derivation of Logical Effort Equations via RC model

Logic Gate Model



Cpi: parasitic cap of gate

Tau

Tau (τ) is the absolute delay of a 1x inverter driving a 1x inverter *with no parasitics.* We assume equal pullup/pulldown Rinv, and Cin = Cinv, so:

$$Tau = \kappa * Rinv * Cinv$$

where κ is a constant characteristic of the fabrication process that relates RC time constants to delay.

Note: Tau is NOT the no-load delay of an inverter. Also, it is not the delay of a 1x inverter driving a 1x inverter since this includes the parasitic delay! This means that determing Tau cannot be done via one delay measurement.

Template Circuit

A template circuit is chosen as the basis upon which other gates are scaled. The scaling factor is α .

Ct is the input cap of the template.

Rt is the pullup or pulldown resistance of the template.

Cpt is the parasitic capacitance of the template.

$Cin = \alpha * Ct$	input cap scales up
$Ri = Rui = Rdi = Rt / \alpha$	channel resistance scales down
$Cpi = \alpha * Cpt$	parasitics scale up

RC Delay

$Dabs = \kappa \ Ri \ (Cout + Cpi)$

$= \kappa \ (Rt/\alpha) \ Cin \ (Cout/Cin) + \kappa \ (Rt/\alpha) \ (\alpha \ Cpt)$

$= (\kappa \ Rt \ Ct) \ (Cout/Cin) + \kappa \ Rt \ Cpt$

Written in this form, can see relation to logical effort model:

$Dabs = \tau \ (gh + p)$

$\tau = \kappa \ Rinv \ Cinv$ (previous definition)

$g = (Rt \ Ct)/(Rinv \ Cinv)$ Note: if template = 1X inverter, then g = 1 !!!!

$h = Cout/Cin$

$p = (Rt \ Cpt)/(Rinv \ Cinv)$

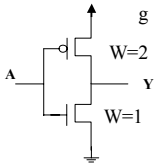
Note: book value of Pinv = 1 only true if Cpt (parasitics) = Cinv (Cgate)!!

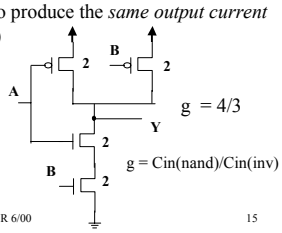
Logical Effort (*g*)

In the Sutherland/Sproull model, the logical effort *g* factor is normalized to a minimum sized inverter for static CMOS.

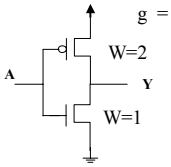
So g for an inverter is equal to 1.

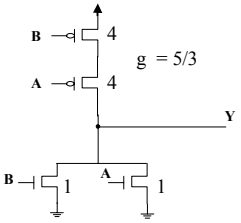
Logical effort *g* of other gates represents how much more input capacitance a gate must present to produce the *same output current* as the inverter (the template gate)





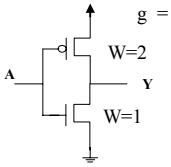
Logical Effort inverter vs nor2

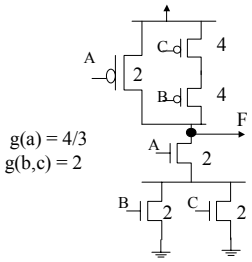




Intuitive result, Nor2 *g* is higher than Nand2 *g*

Logical Effort inverter vs Complex gate





g(a) = 4/3
g(b,c) = 2

Intuitive result, worse case g of complex gate higher than Nand2 or Nor2.

In general, more inputs, more series transistors, the higher the g value.

Logical Effort vs. Electrical Effort

- The value for logical effort *g* depends on what gate is chosen as the template gate (g=1)
  - Choosing a different template gate will alter 'g' values for the other gates in your library
- The *g* value captures the effects of varying number of inputs, and transistor topology on more complex gates than your template gate
- More complex gates will require more logical effort to produce the same output current as the template gate, and will also present a higher input load
- The logical effort for a 1x Nand2, 2X Nand2, 4X Nand2 are all the same – the effect of the extra load by the larger transistors is captured by the *electrical effort parameter*

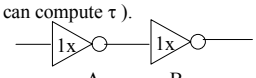
Logical Effort vs. Electrical Effort

- The electrical effort *h* parameter is used to capture the driving capability of the gate via transistor sizing and also the effect of transistor sizes on loading
- Electrical effort *h* is defined as
  Cout / Cin
  where Cout is the load capacitance, Cin is the input capacitance of the gate.
- Note that *h* for a gate will reduce as the transistors become wider since Cin increases (Cout is assume fixed).

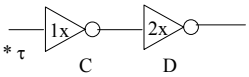
The Parasitic Delay *p*

- Note that the parasitic delay (no-load) *p* is a constant and independent of transistor size; as you increase the transistor sizes the capacitance of the gate/source/drain areas increase also which keeps no-load delay constant
- To measure P (once P is known, can compute τ).

Method #1

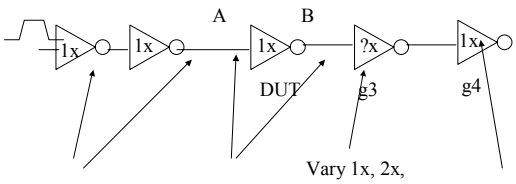


A_delay = (g*h + p) * τ = (1*1 + p) * τ
τ = (A_delay)/(1+p)



C_delay = (g*h + p) * τ = (1*2 + p) * τ
C_delay = (2+p) (A_delay)/(1+p)
p = (2*A_delay – C_delay)/ (C_delay-A_delay)

Method #2: A Better Way to Measure P, Tau



realistic waveform shaping

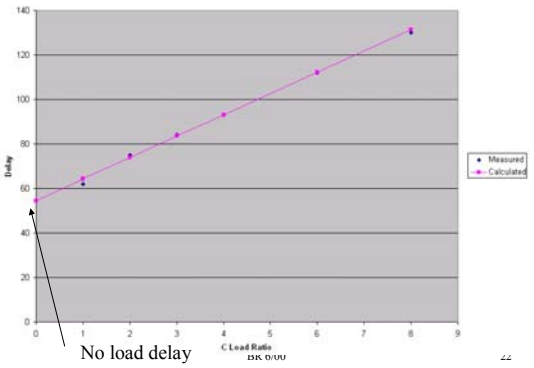
Measure delay from A to B

Vary 1x, 2x, 4x, 6x, 8x

Cload ratio is G3/DUT

Fixed, end load to prevent Miller effect on G3

Plot Delay, Fit to Straight line (delay = mX + b)



No load delay

Tau, Pinv

By definition, ginv = 1.0

From fitted line of mx + b, Tau can be calculated at any point as:

delay = tau (g*h + Pinv)
= tau * g *h + tau * Pinv

When X=0, delay = tau*Pinv = b (y-intercept).

So:

Tau = (delay_measured – b)/Cload

When Tau is known, can compute Pinv

Method #1 vs Method #2

Using Vdd = 2.5, Leda 0.25u

	Tau	Pinv	Pnand2	Pnand4
method1	8.4	6.6	11	23
method2	9.6	5.7	12	30

Differences mainly due to realistic waveshaping of inputs.

Measuring Actual Logical Effort

The Sproull textbook has you calculate logic effort (g) but it can be measured.

When replace all gates in test circuit with Nand2, and plot:

delay = M_{nand2}*X + B

versus

delay = M_{inv}*X + B

the ratio of M_{nand2}/M_{inv} is the logical effort of the Nand2 since the Cload ratios are the same, and Tau is the same.

Logical Effort of Nand2, Nand4

	Nand2 g	Nand4 g
Book	1.33	2
Measured	1.6	2.2

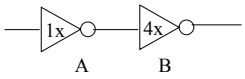
Parasitic Delay of Other Gates

- Normalizing the parasitic delay to that of the inverter can be useful for normalization purposes.
- Some typical values according to Southerland/Sproull:

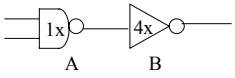
inverter	p_{inv} = 1.0
N-input nand	n* p_{inv}
N-input nor	n* p_{inv}

Will use these values for example purposes.

Delay Estimation



A_delay = g*h + p = 1*(CinB/CinA) + 1
= 1*(4*CinA/CinA) + 1 = 4 + 1 = 5 time units

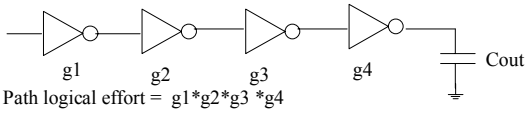


A_delay = g*h + p = (4/3)*(CinB/CinA) + 2*1
Cin_B = 4*3 = 12. Cin_A = 4
A_delay = (4/3)*(12/4) + 2 = 4 + 2 = 6 units

Nand2 worse because of higher parasitic delay than inverter.
Note that g*h term was same for both because NAND2 sized to provide same current drive.

MultiStage Delay

- Recall rule of thumb that said to balance the delay at each stage along a critical path
- Concepts of logical effort and electrical effort can be generalized to multistage paths



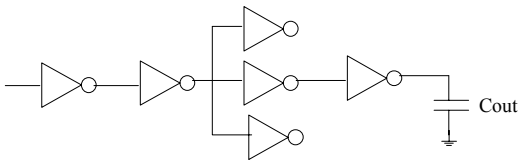
Path logical effort = g1*g2*g3 *g4

In general, Path logic effort G = Π g(i)

Path electrical effort H = Cout / Cin_{first_gate}

Must remember that electrical effort only is concerned with effect of logic network on input drivers and output load.

Off Path Load



Off path load will divert electrical effort from the main path, must account for this. Define a *branching effort b* as:

b = (Con_path + Coff_path) / Con_path

The branching effort will modify the electrical effort needed at that stage. The branch effort *B* of the path is:

B = Π b(i)

Path Effort *F*

Path effort F is:

F = path logic effort * path branch effort * path electrical effort
= G * B * H

Path branch effort and path electrical effort is related to the electrical effort of each stage:

B * H = Cout/Cin * Π b(i) = Π h(i)

Our goal is choose the transistor sizes that effect each stage effort h(i) in order to minimize the path delay!!!!!!!!

Minimizing Path Delay

The absolute delay will have the parasitic delays of each stage summed together.

However, can *focus on just Path effort F* for minimization purposes since parasitic delays are constant.

For an N-stage network, *the path delay is least when each stage in the path bears the same stage effort.*

$f(min) = g(i) * h(i) = F^{1/N}$

Minimum achievable path delay

$D(min) = N * F^{1/N} + P$

Note that if N=1, then d = f + p, the original single gate equation.

Choosing Transistor Sizes

Remember that the stage effort h(i) is related to transistor sizes.

$f(min) = g(i) * h(i) = F^{1/N}$

So

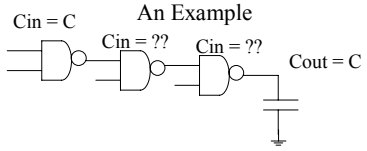
$h(i)\ min = F^{1/N} / g(i)$

To size transistors, start at end of path, and compute:

$Cin(i) = gi * Cout(i) / f(min)$

Once Cin(i) is know, can distribute this among transistors of that stage.

An Example



Size the transistors of the nand2 gates for the three stages shown.

Path logic effort = G = g0 * g1 * g2 = 4/3 * 4/3 * 4/3 = 2.37

Branching effort B = 1.0 (no off-path load)

Electrical effort H = Cout/Cin = C/C = 1.0

Min delay achievable $= 3* (G*B*H)^{1/3} + 3 (2*pinv)$
$= 3 *(2.37*1*1)^{1/3} + 3 (2*1.0) = 10.0$

An example (cont.)

The effort of each stage will be:

$f\ min = (G*B*H)^{1/3} = (2.37*1.0*1.0)^{1/3} = 1.33 = 4/3$

Cin of last gate should equal:

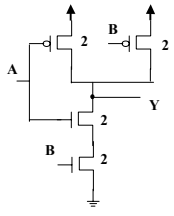
Cin last gate (min) = gi * Cout (i) / f(min)
= 4/3 * C / (4/3) = C

Cin of middle gate should equal:

Cin middle gate = gi * Cin last gate / f(min)
= 4/3 * C/ (4/3) = C

All gates have same input capacitance, distribute it among transistors.

Transistor Sizes for Example

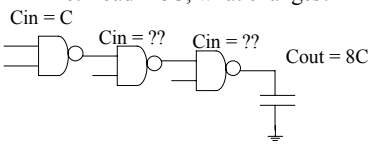


Where gate capacitance of

2 *W *L Mosfet = C/2

Choose W accordingly.

Let Load = 8C, what changes?



Size the transistors of the nand2 gates for the three stages shown.

Path logic effort = G = g0 * g1 * g2 = 4/3 * 4/3 * 4/3 = 2.37

Branching effort B = 1.0 (no off-path load)

Electrical effort H = Cout/Cin = **8**C/C = **8.0**

Min delay achievable = 3* $(G*B*H)^{1/3}$ + 3 (2*pinv)
= 3 *$(2.37*1*\mathbf{8})^{1/3}$ + 3 (2*1.0) = **14.0**

8C Load Example (cont.)

The effort of each stage will be:

f min = $(G*B*H)^{1/3}$ = $(2.37*1.0*\mathbf{8})^{1/3}$ = **2.67** = **8/3**

Cin of last gate should equal:

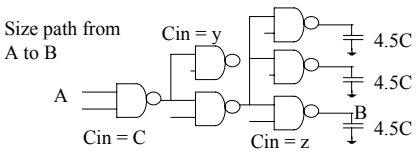
Cin last gate (min) = gi * Cout (i) / f(min)
= 4/3 * **8C** / (**8/3**) = 4C

Cin of middle gate should equal:

Cin middle gate = gi * Cin last gate / f(min)
= 4/3 * **4C**/ (**8/3**) = 2C

Note that each stage gets progressively larger, as is typical with a multi-stage path driving a large load.

Example 1.6 from Chapter 1



Path logic effort G = g0 * g1 * g2 = 4/3 * 4/3 * 4/3 = 2.37
Branch effort, 1st stage = (y+y)/y = 2.
Branch effort, 2nd stage = (z+z+z)/z = 3
Path Branch effort B = 2 * 3 = 6.
Path electrical effort H = Cout/Cin = 4.5C/C = 4.5
Path stage effort = F = G*B*H = 2.37*6*4.5 = 64.

Min delay = $N(F)^{1/N} + P$ = $3*(64)^{1/3}$ + 3(2pinv) = 18.0 units

Example 1.6 from Chapter 1 (cont)

Stage effort of each stage should be:
f(min) = $(F)^{1/N}$ = $(GBH)^{1/N}$ = $(64)^{1/3}$ = 4

Determine Cin of last stage:

Cin(z) = g * Cout / f(min) = 4/3 * 4.5C / 4 = 1.5 C

Determine Cin of middle stage:

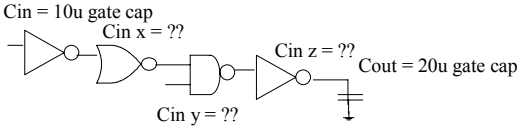
Cin(y) = g * (3*Cin(z))/ f(min) = 4/3 * (3*1.5C) / 4 = 1.5C

Is first stage correct?

Cin(A) = g * (2*Cin(y))/f(min) = 4/3 *(2*1.5C)/4 = C.

Yes, self-consistent.

Example 1.10 from Chapter 1



Path logic effort G = g0 * g1 * g2 * g3 = 1*5/3 * 4/3 * 1 = 20/9
Path Branch effort B = 1
Path electrical effort H = Cout/Cin = 20/10 = 2
Path stage effort = F = G*B*H = (20/9)*1*2 = 40/9

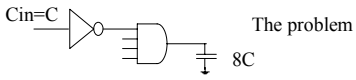
For Min delay, each stage has effort $(F)^{1/N}$ = $(40/9)^{1/4}$ = 1.45

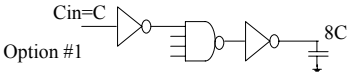
z = g * Cout/f(min) = 1*20/1.45 = 14
y = g * Cin(z)/f(min) = 4/3 * 14 / 1.45 = 13
x = g * Cin(y)/f(min) = 5/3 * 13 / 1.45 = 15

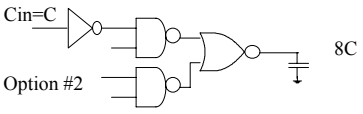
Misc Comments

- Note that you never size the first gate. This gate size is assumed to be fixed (same as in the Tilos algorithm) – if you were allowed to size this gate you find that the algorithm would want to make it as large as possible.
- This is an estimation algorithm. The author claims that sizing a gate by 1.5x too big or two small still results in path delay within 5% of minimum.

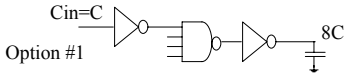
Evaluating different Structure options







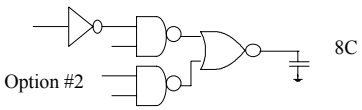
Option #1



Path logic effort G = g0 * g1 * g2 = 1*6/3 * 1 = 2
Path Branch effort B = 1
Path electrical effort H = Cout/Cin = 8C/C = 8
Path stage effort = F = G*B*H = 2*1*8 = 16

Min delay: = N* $(F)^{1/N}$ + P
= 3 *$(16)^{1/3}$ + (pinv + 4*pinv + pinv)
= 3 *(2.5) + 6 = 13.5

Option #2



Path logic effort G = g0 * g1 * g2 = 1*4/3 * 5/3 = 20/9
Path Branch effort B = 1
Path electrical effort H = Cout/Cin = 8C/C = 8
Path stage effort = F = G*B*H = 20/9*1*8 = 160/9

Min delay: = N* $(F)^{1/N}$ + P
= 3 *$(160/9)^{1/3}$ + (pinv + 2*pinv + 2*pinv)
= 3 * 2.6 + 5 = 12.8

Option #2 appears to be better than Option #1, by a slight margin.