High Speed CMOS Circuit Design

Lecture 2: Static Design

(c) 1997 David Harris

1.0 Introduction to Simulation

Fabricating a chip is a lengthy and expensive process. Designers need to estimate the delay of circuits and test the functionality of unusual ideas without repeating the fabrication process too many times. Simulation is the art of building a model of a physical chip, then mathematically evaluating the model. It is much cheaper than fabrication, but is only as good as the quality of the model. Moreover, learning what is important to model and what can be ignored can be challenging and evaluating a detailed model takes large amounts of time.

There are many levels of model accuracy. HSPICE is one of the most detailed simulation tools; it can use complex transistor models and solve the differential equations to predict currents and voltages. However, even HSPICE is very limited because the user must know what cases to simulate. Transistors vary by at least a factor of 2 in performance over processing extremes; therefore a single simulation cannot possibly match all chips. Moreover, two transistors on the same chip do not perform identically. If a circuit's operation depends on the degree to which ideally identical devices match, the designer must identify the possible mismatch and include it in the simulation. Similarly, if power supply or thermal variations impact circuit performance, they may have to be modeled. As with any computer program, garbage in-garbage out.

Even with fairly simple models, HSPICE takes too long to run on large circuits. Individual gates are acceptable; larger modules such as 64 bit adders push the limits of what can be simulated in a reasonable amount of time. Therefore, HSPICE is generally reserved to characterize a library and verify potentially risky circuits. Other less detailed simulators are used to verify the entire chip and estimate the timing of all the paths. These other tools, such as static timing analyzers, base timing information upon simpler models curve fit to data from a modest number of SPICE simulations.

2.0 Using HSPICE

This section gives a brief example of using HSPICE. If you have never used the simulator, you will probably want a more detailed tutorial. For instance, a tutorial and lengthy reference manual are available at:

`http://yake.ecn.purdue.edu/~seib/ee255/spicehandouts.html`

Those who like a hard copy of a reference manual may prefer the 3-volume HSPICE User's Manual. Volume I is especially handy for power users who want to take advantage of advanced measurement and optimization commands.

The following code is an example of a SPICE deck which measures the delay through an inverter. The first line should always be a comment because it is ignored by SPICE.

```
* example.hsp

* Written 9/13/97 by David Harris
+ harrisd@leland.stanford.edu
* This spice deck measures the delay of an
* inverter with a sharp ramp input.

**************************************
* Set supply and library
**************************************

.param Supply=2.5* Set voltage
.protect    * Don't print the contents of library
.lib '~/lib/cad/spice/hp0.6u/opConditions.lib' TT
* Load the library
.unprotect  * Resume printing SPICE deck

.opt scale=0.35u* Define lambda
* (half minimum channel length)

**************************************
* Define power supply
**************************************

.global Vdd Gnd
Vdd   Vdd   Gnd   'Supply'

**************************************
* Define Subcircuits
**************************************

.subckt inv In Out N=8 P=16
* Assumes 5 lambda of diffusion on the source/drain
m1    Out In Gnd Gnd nmos l=2 w=N
+                         as='5*N' ad='5*N'
+                         ps='N+10' pd='N+10'
m2    Out In Vdd Vdd pmosl='2' w=P
+                         as='5*P' ad='5*P'
+                         ps='P+10' pd='P+10'
.ends

**************************************
* Top level simulation netlist
**************************************

x1    In    Out   inv   * Inverter
C1    Out   Gnd   0.1pF
```

```
************************************
* Stimulus
************************************

* Format of pulse input:
* pulse v_initial v_final t_delay t_rise t_fall
* t_pulsewidth t_period

Vin     In      Gnd     pulse 0 'Supply' 0.2ns 0.05ns
+ 0.05ns 1ns 2ns

************************************
* Measurements
************************************

* Measure delay through inverter
.measure invRise
+       TRIG v(In)  VAL='Supply/2' FALL=1
+       TARG v(Out) VAL='Supply/2' RISE=1
.measure invFall
+       TRIG v(In)  VAL='Supply/2' RISE=1
+       TARG v(Out) VAL='Supply/2' FALL=1
.measure invAvg param='(invRise + invFall)/2'

.tran .01ns 2ns
.plot V(In), V(Out)

************************************
* End of Deck
************************************
.end
```

Gate delays are significantly affected by internal parasitics, which must be modeled with the AS, AD, PS, and PD terms for the areas and perimeters of source/drain diffusions. When layout information is not available, assuming a contacted diffusion region is reasonable. Similarly, wire capacitance is very important in many designs. It can be difficult to estimate without layout data. At the very least, estimated capacitances of long wires should be included. If other wires are not modeled, budget for your circuit becoming slower when layout becomes available.

3.0 HSPICE Productivity Tips

The easiest way to improve your productivity with HSPICE is to minimize your use of the simulator! HSPICE only gives numerical results, which are only as accurate as the model you provide. Unless you need a large amount of precision, it is faster to estimate delays with a static timing analyzer.

If you absolutely need to use HSPICE, you may want to keep a log of each run you use on a particular problem. Comment in the log about the date and time of the run, the measured result, the changes from the previous run, and what you learned by making the change.

Such a log often reveals when a circuit is being run through SPICE dozens of times with minor tweaks between runs, consuming days of engineering effort while making minimal improvement in circuit performance.

Such careful and limited use of SPICE is the most important step toward high productivity circuit design. However, there are a few technical features of HSPICE not found in other SPICE simulators that also help productivity. They include waveform viewing, good programming style support, measurement statements, and optimization.

3.1 Waveform Viewing

By placing a .options post statement in your HSPICE deck, you can direct the simulator to write binary dumps of the complete simulation results to disk. For example, when you run a .TRAN transient analysis, the simulator will produce a deck_name.tr0 file listing all circuit voltages and currents as a function of time.

You can then run the mwaves waveform viewer on the deck_name.tr0 file to plot any voltages and currents of interest.

Waveform viewing is most useful when a circuit is not behaving the way you expect. You can trace the problem from the outputs which are behaving strangely to the inputs and see at which point internal nodes behave unexpectedly. Frequently, this reveals simple typos in the SPICE deck instrumentation code which would have been difficult to find without a waveform viewer.

3.2 Good Software Design

Remember that SPICE decks are just software. All the good practices of software design apply to SPICE deck design as well. These include:

- Format decks to be easy to read
- Use self-explanatory node and parameter names (never name a node foo!)
- Use comments
- Use parameters to make constants easy to change
- Use hierarchy to make circuits more comprehensible

If you write clear and easy to understand SPICE decks, you will be rewarded years later when you need to revisit the deck and discover why the fabricated chip is not acting as expected. You often can also reuse much of your deck for similar simulations.

Parameters are useful for any feature of a circuit that might change. Transistor widths are especially popular. If you expect to port the circuit to another process in the future, making channel length a parameter by specifying λ is also useful.

SPICE decks take advantage of hierarchy through the .subckt command. The deck below combines many of these elements of good software design. The inverter subcircuit is given default transistor widths that can be overridden by a call to the subcircuit. Channel length and diffusion parasitics are given as a function of lambda. The assumptions are commented and the variable names are easy to read (as opposed to naming In and Out n1 and n2).

```
**********************************************
* Normal Skew Inverter
**********************************************
.subckt inv In Out N=8u P=16u
* Assumes 5 lambda of diffusion on the source/drain
m1     Out In Gnd Gndnmosl='2*lambda' w=N
+                 as='5*lambda*N' ad='5*lambda*N'
+                 ps='N+10*lambda' pd='N+10*lambda'
m2     Out In Vdd Vddpmosl='2*lambda' w=P
+                 as='5*lambda*P' ad='5*lambda*P'
+                 ps='P+10*lambda' pd='P+10*lambda'
.ends
```

3.3 Measure Statements

As noted in the earlier example, HSPICE measure statements are very useful to report simulated results. With other versions of SPICE, one often must plot tables of output voltages, then manually read off propagation delays.

The most common use of measure statements is to compute the time between a trigger event and a target event. Another common use is to compute functions of other measured variables, such as the average of rise and fall delays. Examples of these uses are:

```
* Measure delay through inverter from Inb to Inv
.measure invRise
+       TRIG v(Inb)  VAL='Supply/2' FALL=1
+       TARG v(Inv)  VAL='Supply/2' RISE=1
.measure invFall
+       TRIG v(Inb)  VAL='Supply/2' RISE=1
+       TARG v(Inv)  VAL='Supply/2' FALL=1

* Compute the average delay
.measure invAvg param='(invRise + invFall)/2'
```

CAD tools can automatically generate SPICE decks, then pull the measured results out of deck_name.mt0 files.

More details on .MEASURE can be found in the Simulation Output and Controls chapter of the HSPICE Users Manual (page 3-13 of the 1996 edition). Even derivatives and integrals can be measured!

3.4 Sweeps & Optimization

Often, it is interesting to see how an output varies for different values of a parameter and thus to find the optimal value of the parameter. This can be done by tweaking the value of the parameter by hand and running many simulations, a tedious process. HSPICE automates the process by automatically sweeping specified parameters across various values and by actually tweaking parameters for you until a condition you specify is optimized.

For instance, to measure the delay of various fanout inverters, you could request the following analysis:

```
.tran 0.1ns 10ns SWEEP fanout 0 8 2
```

This instructs HSPICE to run five separate simulations, assigning the parameter fanout the value 0, 2, 4, 6, and 8.

Better yet, HSPICE can tune parameters for you to optimize a value. For example, it could find the P/N ratio of an inverter which minimizes average delay. To do this, the deck needs a list of parameters to adjust, a measurement which should be optimized, a target value for the measurement, and a request to optimize. Such a deck is shown below:

```
* pn.hsp

* Written 9/13/97 by David Harris
* harrisd@leland.stanford.edu
* This spice deck optimizes the P/N ratio of an
* inverter for minimum average delay.
*
* The deck uses a first inverter to shape the input
* slope, a second inverter to measure, a third
* inverter as a load, and a fourth
* inverter as load on the load.

*****************************************************
* Set supply and library
*****************************************************

.param Supply=2.5       * Set voltage
.protect    * Don't print the contents of library
.lib '~/lib/cad/spice/hp0.6u/opConditions.lib' TT
.unprotect  * Resume printing SPICE deck

.param lambda=0.35u     * Define lambda
.param fanout=4         * Define fanout of gate

* Save results of simulation for viewing
.options post

*****************************************************
* Define power supply
*****************************************************
.global Vdd Gnd
Vdd   Vdd   Gnd   'Supply'

*****************************************************
* Define Subcircuits
*****************************************************
.subckt     inv In Out N=8u P='N*pnratio'
* Assumes 5 lambda of diffusion on the source/drain
```

```
m1    Out In Gnd Gnd nmos l='2*lambda' w=N
+                 as='5*lambda*N' ad='5*lambda*N'
+                 ps='N+10*lambda' pd='N+10*lambda'
m2    Out In Vdd Vdd pmos l='2*lambda' w=P
+                 as='5*lambda*P' ad='5*lambda*P'
+                 ps='P+10*lambda' pd='P+10*lambda'
.ends

*****************************************************
* Top level simulation netlist
*****************************************************
x1    In    Inb   inv   * set appropriate slope
x2    Inb   Inv   inv   M='fanout' *DUT
x3    Inv   Out3  inv   M='fanout * fanout'* load
x4    Out3  Out4  inv   M='fanout * fanout * fanout'

*****************************************************
* Stimulus
*****************************************************
* Format of pulse input:
* pulse v_initial v_final t_delay t_rise t_fall
* t_pulsewidth t_period

Vin In Gnd pulse 0 `Supply' 1ns 0.1ns 0.1ns 4ns 10ns

*****************************************************
* Measurements
*****************************************************
* Measure delay through inverter x2

.measure invR
+       TRIG v(Inb)  VAL='Supply/2' FALL=1
+       TARG v(Inv)  VAL='Supply/2' RISE=1
.measure invF
+       TRIG v(Inb)  VAL='Supply/2' RISE=1
+       TARG v(Inv)  VAL='Supply/2' FALL=1

* Compute the average delay
.measure invA param='(invR + invF)/2' GOAL = 0

.param pnratio = opt1(2, 0.5, 3)
* Search ratios between 0.5 and 3 starting 2
.model optmod opt itropt=30 * maximum of 30 iterations
* on the search
.tran .01ns 12ns SWEEP OPTIMIZE=opt1 RESULTS=invA
+ MODEL=optmod

*****************************************************
* End of Deck
*****************************************************
.end
```

This deck finds a pnratio of 1.38 to minimize average delay for the particular process.

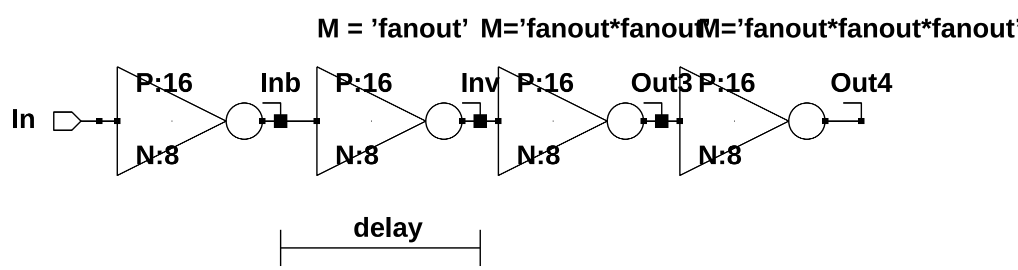
HSPICE optimization is notorious for not converging or converging on the wrong answer. In general, nonlinear optimization problems are tricky and can't be guaranteed to converge; HSPICE's algorithms don't do a very good job when optimizations involve more than one variable. The moral is to only use optimization for simple problems and to sanity check the results.

4.0 Measuring Process Parameters

One of the most important applications of a detailed circuit simulator like HSPICE is to extract process parameters which can be used by other simpler simulators and estimation schemes. For example, to use the hand estimation techniques we have been studying, we need to know the values of R, C, τ, and a FO4 delay. We can compute these values with two HSPICE simulations.

First consider delay, i.e. τ and the FO4 delay. The delay of a FO4 inverter in τ is $t_{intrinsic}$ + fanout * (1+pnratio). Fanout is 4 and we'll assume we design the gate with a 2:1 P/N ratio. Thus, the delay is 12+$t_{intrinsic}$. $t_{intrinsic}$ depends on the diffusion and wire parasitics; we have estimated it earlier as 3, for a total FO4 delay of 15τ. Even if our intrinsic delay varied from 1.5 to 4.5, 50% estimation errors, the FO4 delay would only vary by 10%. Thus, defining τ to be 1/15 of a FO4 delay is a good approximation. Therefore, we can build a test deck to measure the FO4 delay, as shown in Figure 5.

FIGURE 1. Inverter Delay Schematic

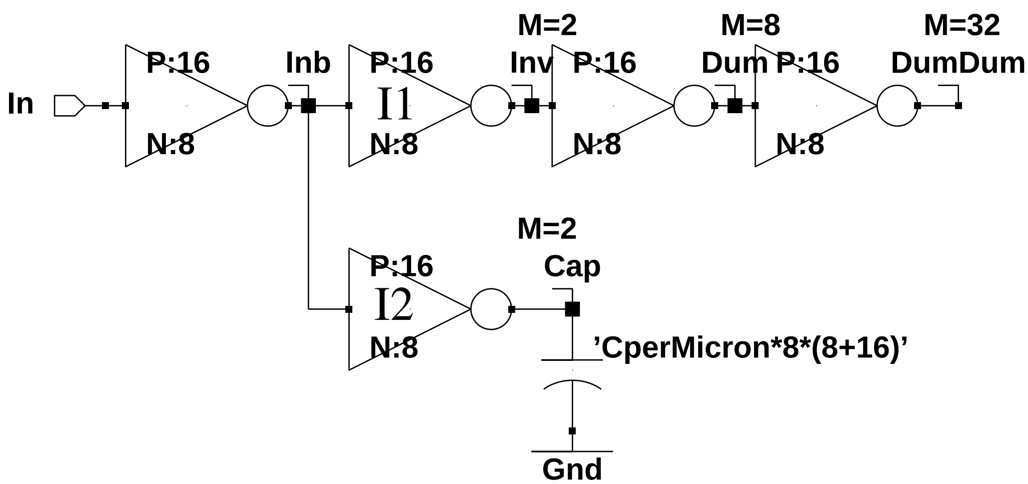


The fanout parameter is set to 4; M is a multiplier indicating M copies of a gate should be ganged in parallel. The first inverter is used to produce an appropriate input slope on node Inb. The second inverter is the FO4 inverter being measured. The third inverter is a load providing a fanout of 4 to the second inverter. Interestingly, the effective gate capacitance of the third inverter depends on how fast its output switches because there is some coupling between gate and drain of each transistor that gets magnified by the Miller effect when the output switches. Therefore, we provide a fourth inverter to load the load and set approximately the correct Miller effect.

A HSPICE deck can now model this circuit and measure the delay from Inb to Inv. Since rise and fall delays are unlikely to match, we can use the average delay as the FO4 delay, then compute τ as 1/15 of the measured FO4 delay.

Next, we can use HSPICE's optimization feature to extract C. The circuit in Figure 6 shows a chain of fanout-of-4 inverters. The top part of the chain drives inverters as loads, while the bottom part drives a linear capacitor. If we adjust the parameter CperMicron until inverter I2 has the same delay as inverter I1, then the capacitor must have the same average capacitance as the load I1 drives. This optimizer can be instructed to adjust CperMicron until the difference between the average delay of I1 and I2 is 0, corresponding to the value of C for a 1 micron wide transistor.

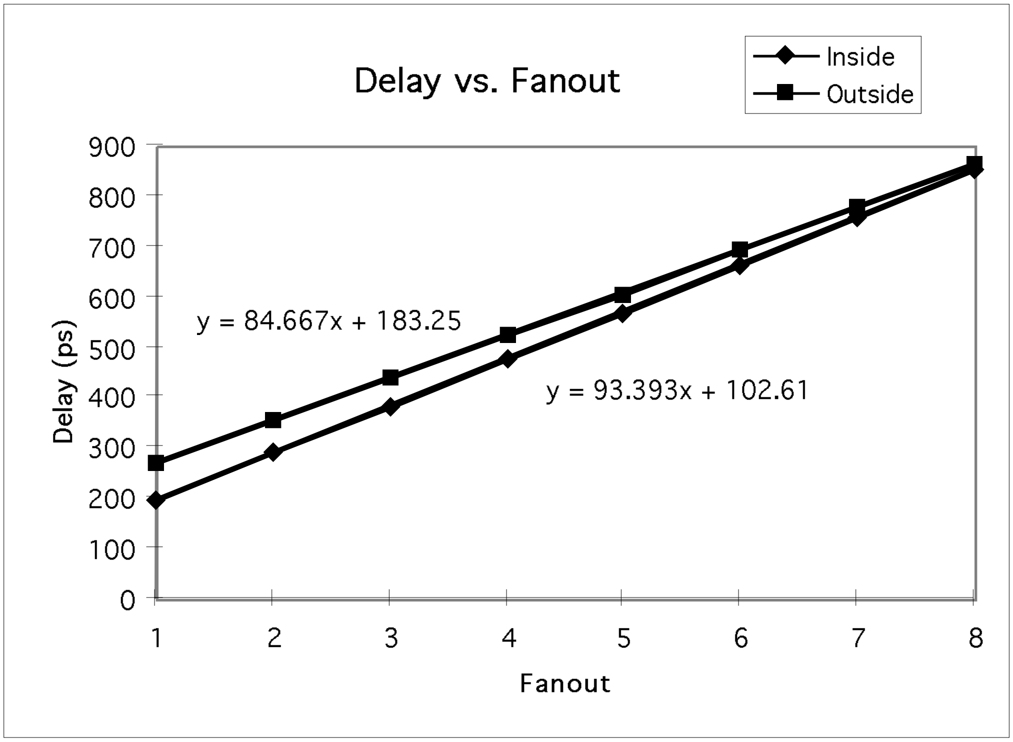
FIGURE 2. Circuit for capacitance extraction



Once C and τ are known, R can be computed as τ/C for a one micron wide gate. Resistance scales inversely with gate width.

Logical efforts can also be obtained by simulation. Simulate the delay of a inverter at several fanouts (e.g. 2, 4, and 6), and fit the delay to the equation $t = a_{inv} + b_{inv} * f$. Similarly, fit the delay of another gate of interest to the equation $t = a_{gate} + b_{gate} * f$. The logical effort of the gate can then be shown to be b_{gate} / b_{inv}, corresponding to how much longer it takes the gate to drive a particular fanout than an inverter would take. In most processes, the curve fit is remarkably close to the measured data, justifying the a + b*f model. For example, delay of a 2-input NOR gate is plotted in Figure 7. Notice that the delay depends on the input; inputs closer to the rail are slower.

FIGURE 3. Curve fitting delay vs. fanout for 2-input NOR (0.8 μm process)



5.0 Matching & Simulation Corners

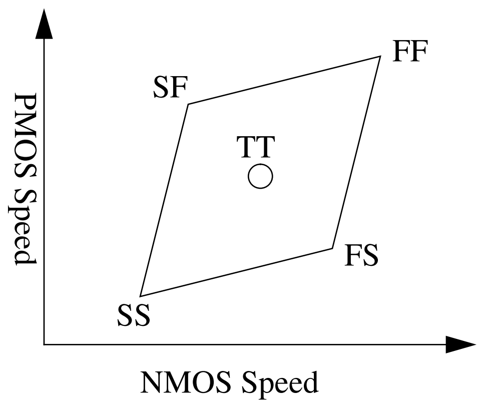
The physical parameters of a chip vary from die to die and across individual dice. In order to keep yield high, a chip should operate correctly over a wide range of process variation. The best and worst case processing over which the chip should work are called process corners. The amount of variation across a chip is called process tilt.

Different dice may see different:

- Polysilicon linewidth (channel length)
- NMOS Vt
- PMOS Vt
- Oxide thickness
- Parasitic capacitances

Some of these parameters such as oxide thickness correlate well between NMOS and PMOS transistors because they are set for both kinds in a single step. Others such as threshold voltages correlate poorly because they are set by different processing steps. Therefore, the performance of NMOS and PMOS transistors may vary independently. For each kind of transistor, we identify typical (T), fast (F), and slow (S) processing. A process can therefore be characterized by two letters, one for NMOS and one for PMOS. This is shown in Figure 8:

FIGURE 4. Process Corners



Notice how the FF corners and SS corners are more extreme than the FS and SF corners. This is because some parameters like oxide thickness track for both flavors of transistors and therefore cannot improve one flavor while making the other flavor worse.

Environmental variations, particularly of voltage and temperature, can make as much difference as processing. Therefore High and Low voltage and temperature corners are generally defined. For example, a chip normally operating at VDD=2.5 volts may use a high VDD of 2.75 and low VDD of 2.25. A high temperature may be 120 degrees C and a low temperature may be 0 degrees C for a part operating in the commercial temperature environment, since the actual transistor temperature on a hot die may far exceed the ambient temperature. Finally, an additional letter is sometimes added representing metal performance.

The uses of various corners are summarized in Table 1. The SF and FS corners are important for ratioed circuits such as pseudo-NMOS which fail if the PMOS is too strong, but run very slowly when the PMOS is too weak. The last two corners are important for self-timed circuits and other applications where there are races. If the amount of wire and gate delay are not equal, margin must be provided so the race does not cause failure in either corner.

TABLE 1. Process Corners

NMOS	PMOS	Wire	Supply	Temp	Check For
T	T	T	Nom	Med 85	typical chips
F	F	F	+10%	Low 0	max power
S	S	S	-10%	Hi 125	slowest chips
F	S	T	Low	High	ratioed circuit failure
S	F	T	High	Low	ratioed circuit failure
F	F	S	High	Low	gates outracing wire
S	S	F	Low	High	wire outracing gates

Process tilt is becoming increasingly important, but is not yet well characterized for most processes. It comes from systematic and random variations of parameters across a die. Many important parameters such as clock skew are influenced by such tilt; for example a clock distribution network that uses fast transistors and high voltage in one corner of the die but slow transistors and low voltage in the other corner will show skew between the clock edges as a result. There will be a certain amount of variation even between very close transistors such as differential pairs in a sense amplifier, resulting in offset voltages on the sense amplifier. It is a challenge to properly model the variation to understand worst case offset voltages. Layout techniques, such as drawing the transistors in the same orientation, are also important to get well matched transistors. Gate length is influenced by the etch rates of plasmas, which are in turn influenced by the amount of polysilicon in the area to be etched. Therefore, dummy polysilicon lines are often used around gates, such as clock buffers, that should be identical across the chip, so that the local polysilicon density is nearly constant.

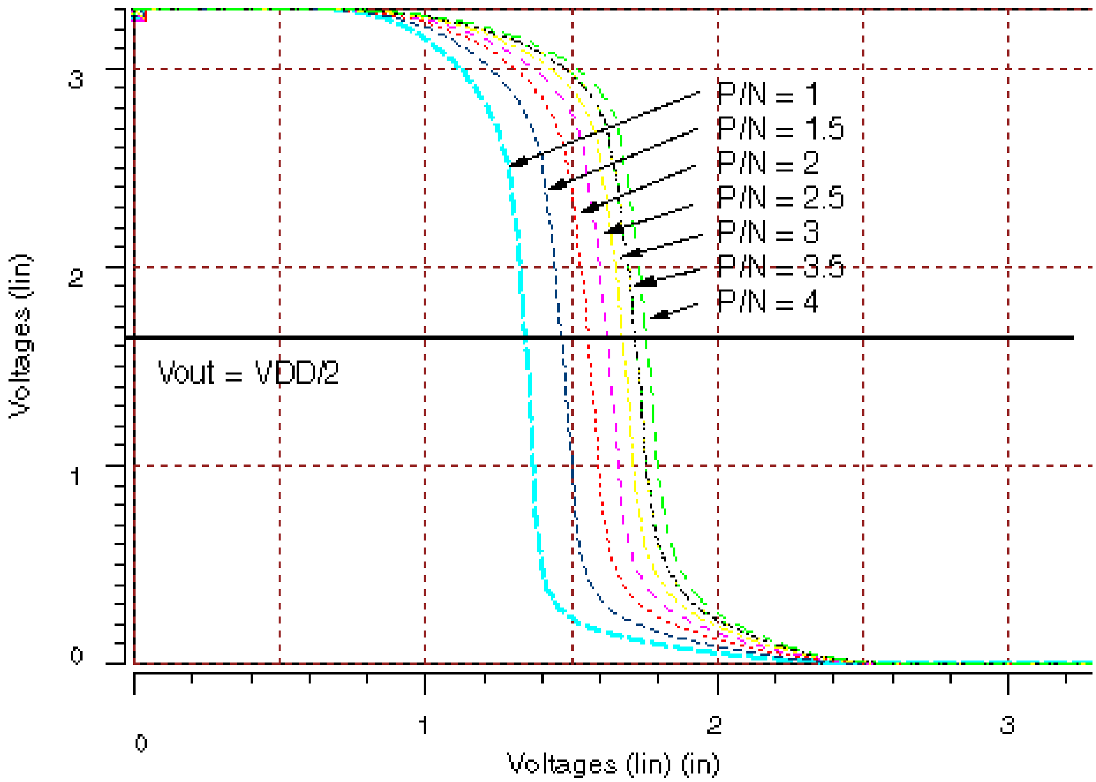
6.0 P/N Ratios

Static CMOS gates are a "ratioless" circuit family, meaning that the gates will work correctly for any ratio of PMOS sizes to NMOS sizes. However, the ratios do influence switching threshold and delay, so it is important to optimize the P/N ratio for high speed designs. In this section, we will explore the DC transfer characteristics of various ratios, the question of sizing for equal rise/fall resistance or minimum delay, and skewing gates to favor critical edges.

6.1 Switching Threshold

Figure 1 shows the output voltage as a function of input voltage for inverters with various P/N ratios.

FIGURE 5. Inverter Switching Thresholds



Notice how increasing the P/N ratio also raises the input voltage for which the output reaches VDD/2. In the process shown, a P/N ratio of 2.5 centers the curve so that an input of VDD/2 produces an output of VDD/2.

6.2 Skewing Gates

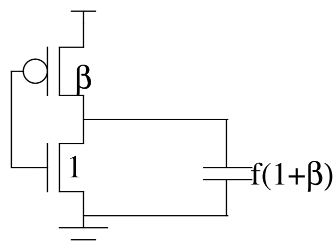
Using a higher or lower P/N ratio favors rising or falling outputs, respectively. For example, with a P/N ratio of 4/1, the input does not have to fall as far as VDD/2 before the output could switch. We call such a circuit a "high skewed" gate and use it on paths where the critical transition is a rising output. Similarly, a 1/1 P/N ratio could be used in a "low skewed" gate for critical falling outputs. A 2/1 P/N ratio roughly centers the gate; depending on relative mobilities, a ratio of 2/1-3/1 may be needed. Even higher or lower skews may be used, but they greatly slow the non-critical edge and severely reduce noise margins so a range of 1/1 to 4/1 is generally preferred.

P/N ratios apply to other static CMOS gates besides inverters. For example, a normal skew NAND2 gate uses equal sized NMOS and PMOS transistors because the NMOS are in series. A high-skew NAND2 doubles the PMOS width, while a low-skew NAND2 doubles the NMOS width. Similarly, a normal skew NOR2 gate uses PMOS transistors four times the NMOS width. A high skew NOR2 uses 8x PMOS, while a low skew NOR2 uses 2x PMOS transistors. Skewing NOR gates high is rarely done because such large PMOS transistors are needed.

6.3 Improving Average Delay

Normal skew gates have equal rise and fall resistances. However, this is not optimal for average circuit delay. By using a smaller P/N ratio, the input load can be significantly reduced while only somewhat slowing the rising output. Thus, the average delay of a gate decreases, though the rise and fall times become unbalanced. Consider an inverter driving a fanout of f with an NMOS transistor sized at one unit and a PMOS transistor sized β times larger, as shown in Figure 2. Suppose the gate has equal rise and fall times for $\beta = k$ (i.e. 2). Neglect parasitic capacitances because they turn out to not affect the conclusions.

FIGURE 6. P/N ratio of inverter



The falling delay is $f(1+\beta)\,\tau$. The rising delay is $(k/\beta)f(1+\beta)\,\tau$. Thus, the average delay of the gate is:

$$\frac{1}{2}f(1+\beta)\left(1+\frac{k}{\beta}\right)\tau \qquad \text{(EQ 1)}$$

We can solve for the P/N ratio β that minimizes delay by taking the derivative and setting it to 0:

$$\frac{dDelay}{d\beta} = 0 = \frac{RC}{2}f\left(1-\frac{k}{\beta^2}\right) = 0 \Rightarrow \beta = \sqrt{k} \qquad \text{(EQ 2)}$$

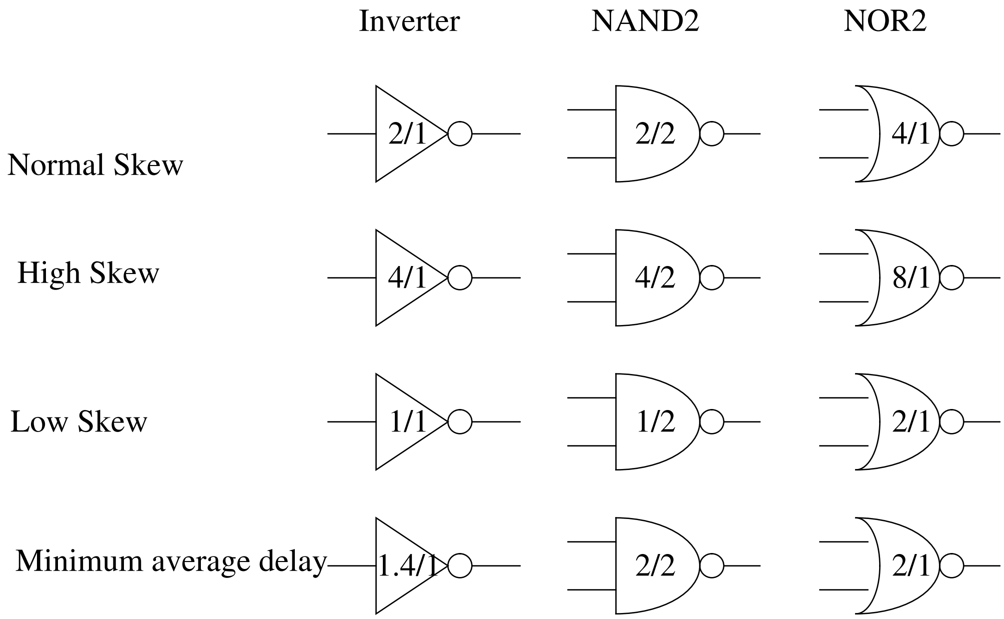
Therefore, the optimal P/N ratio to minimize average path delay is the square root of the ratio that gives equal rise/fall resistances. Since the mobility ratios are 2-3, the best P/N ratios for average delay are 1.4-1.7; 1.5 is a convenient number to use.

For more complex gates, the same analysis holds: average delay is optimized by setting the P/N ratio to the square root of that which gives equal rise/fall resistances. Thus, a NOR3 gate which requires a P/N ratio of 6 for equal rise/fall resistance can be designed instead with a ratio of sqrt(6) = 2.45, greatly reducing area and power as well as average delay!

6.4 Summary

Figure 3 summarizes the P/N ratios of simple gates sized for equal rise/fall resistance (normal skew), high skew, low skew, and minimum average delay. Unless otherwise specified, we will generally assume normal skew gates to keep calculations simple. However, a cell library will usually be sized for minimum average delay.

FIGURE 7. P/N Ratios



7.0 Introduction to Circuit Families

Chips can be optimized at many levels for speed, area, power, yield, etc. For years, the microarchitectural level was most important and innovations such as pipelining and superscalar execution raised processor performance. Now that increasing IPC is very difficult, circuit optimizations are more important for continuing to improve performance. Designers have looked at many circuit families in hopes of boosting performance.

Unfortunately, most circuit families suffer from critical flaws which mean the families are best relegated to old journal articles and dusty Ph.D. theses. Only a few families are relevant to general-purpose high speed circuit design. Static (a.k.a. complementary) CMOS gates are the simplest and most widely used. Pass-transistor logic, in its many flavors, also has niches where it effective. Pseudo-NMOS gates are an efficient way to implement wide NOR gates. Dynamic circuits are trickier to use, but offer the highest performance of any general-purpose circuit family.

In this lecture, we will examine the three important static logic families: static CMOS, pass-transistor logic, and pseudo-NMOS. In the next lecture, we will look at a variety of dynamic logic techniques. In the third lecture, we will consider a few special-purpose circuit styles as well as circuit pitfalls which plague many ill-conceived circuits.

The emphasis will be on high performance. Low power design is also becoming increasingly important and will be covered in a later lecture; in general, static circuits are better than dynamic circuits for low power consumption.

8.0 Static CMOS Logic

Static CMOS gates are the simplest and easiest to use circuit family. Their advantages include:

- Very robust -- "nearly idiotproof!"
- "Zero" DC power (leakage => nothing is zero in the real world)
- Low AC power
- Insensitive to process variation, if you wait long enough
- Scales well to low voltage
- Handled well by synthesis tools and simulators
- Well understood

Thus, static CMOS gates should be used by default unless there is a good reason to do otherwise. Designers have a few tricks to squeeze maximum performance out of static gates. They include topology selection, skewing gates, optimizing for critical inputs, and perhaps tapering stacks. Let's discuss each further.

8.1 Topology Selection

The most important part of creating fast circuits is to choose a good topology of gates. The right number of stages depends on the total gain of the path and should result in a gain of about 4 per stage, though anything in the range 2-6 is not bad. Thus, if very little logic must be performed but the fanout is large, simple stages like 2-input gates and inverters should be used. If lots of logic must be performed but little fanout is necessary, complex multiple input gates should be used. CMOS logic has the advantage of supporting arbitrary AND-OR-invert structures as well as simple NAND and NOR gates.

Nevertheless, beyond a certain point, complex gates are always slower than two stages of simpler gates. This occurs because delay from internal parasitics increases quadratically with the number of series transistors. The maximum number of series transistors in a typical CMOS gate should be about 4 NMOS or 3 PMOS. An exception is that 5 series NMOS devices in domino gates are sometimes acceptable, especially since the bottom one may be a wide clocked transistor that turns on early.

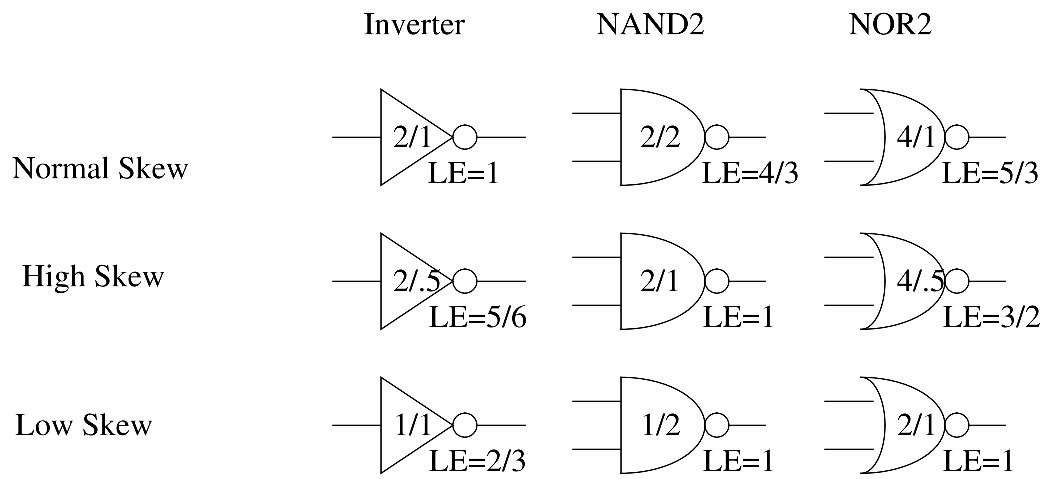
DeMorgan's law provides a fair amount of flexibility choosing between NAND and NOR gates. In general, NAND gates are preferable because their series stacks consist of fast NMOS transistors rather than slower PMOS transistors. This is clear from the logical efforts of gates.

8.2 Skewed Gates

When a transition in a particular direction is known to be most critical, CMOS gates may be skewed to favor that transition, as mentioned in a previous lecture. This speeds up the critical edge at the expense of the other edge. Care must be taken to ensure the other edge does not become a performance limiter. How do we compute the logical effort of such skewed gates?

The logical effort of a skewed gate is equal to it's input capacitance divided by the input capacitance of an inverter that has the same drive strength on the critical transition. For example, Figure 1 shows the sizing and logical efforts of skewed inverters, NAND gates, and NOR gates.

FIGURE 8. P/N Ratios

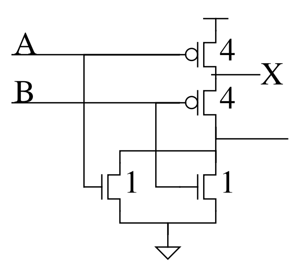


As expected, skewed gates have lower logical effort because less input capacitance is dedicated to the non-critical edge.

8.3 Critical Inputs

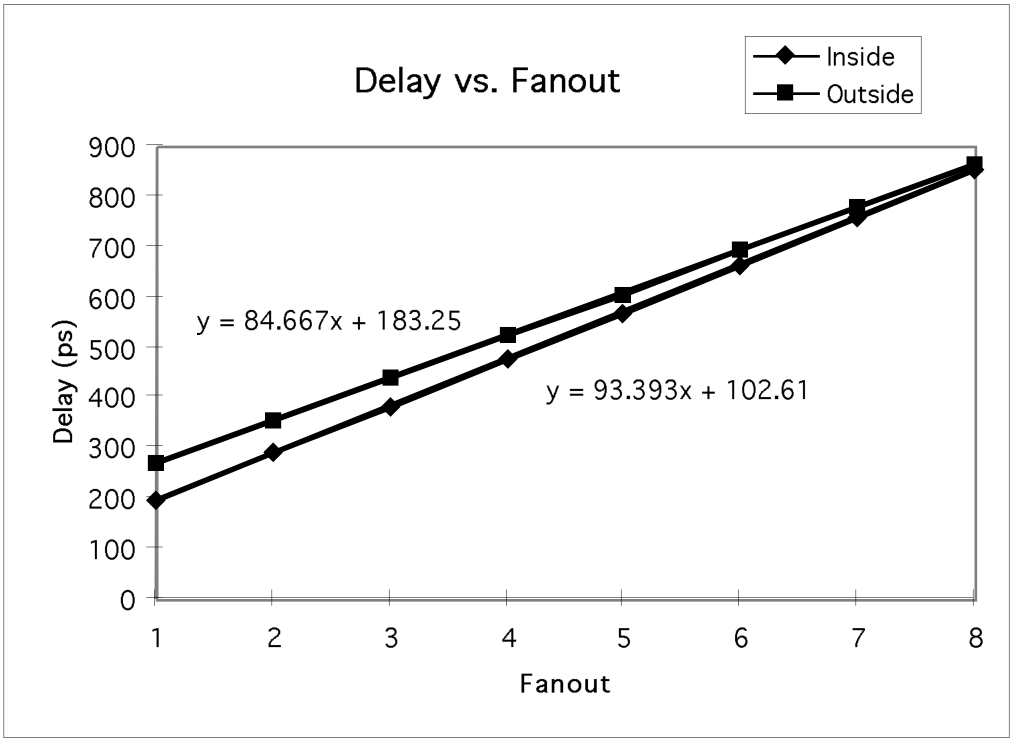
The delay of a gate depends on the input pattern. For example, consider a 2-input NOR gate shown in Figure 2. Input A is closest to the rail and is thus called the outside input. Input B is closest to the middle and thus is called the inside input.

FIGURE 9. 2-input NOR gate



Consider the delay from A and B falling to the output rising. If A falls first, it can charge up the capacitance on node X. When B arrives, the output will respond quickly because X is already set and because charge sharing between node X and the output will further reduce the delay. On the other hand, if B falls first, X will be low. When A arrives, the output will rise more slowly because both the output capacitance and node X must be charged up. This is confirmed by the measured data in Figure 3 which shows the inner input to be substantially faster then the outer input for small loads. When both inputs switch simultaneously, the gate is even slower because neither transistor is fully turned on during the input transition time.

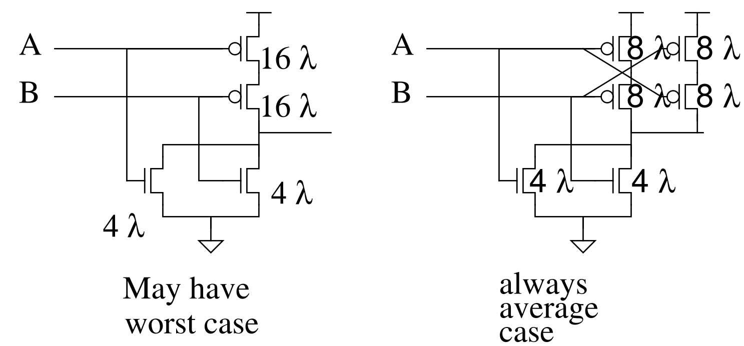
FIGURE 10. Curve fitting delay vs. fanout for 2-input NOR (0.8 μm process)



When fanout becomes large, the load capacitance dominates the internal capacitance, so the inner input has less of an advantage. More subtly, when the inner input switches last, it causes some negative feedback because node X will droop, reducing the V_{gs} of the inner input. This negative feedback is most important for slow input slopes because such slopes keep the inner transistor in saturation for longer periods of time, during which current depends quadratically rather than just linearly on V_{gs}. The outer input does not experience this problem because its source cannot droop. In the simulation used to produce Figure 3, the input slopes increased along with output slopes. Thus, the inside input experiences more negative feedback at higher fanouts. For very large fanouts, generally well beyond the useful loading range of a gate, the inside input can become slower than the outside input.

When possible, early signals should drive transistors close to the rails for best speed. Sometimes the arrival times of signals are unknown. In such a case, it may still be possible to get an average delay instead of worst-case delay, especially when transistors are large enough that they had to be folded anyway. Such a scheme is shown in Figure 4:

FIGURE 11. Twisting stacks for average delay



8.4 Tapering Stacks

As we have seen earlier, delay of long series transistor stacks increases quadratically with the number of series transistors because both resistance and parasitic capacitance increase proportionally to the number of transistors. It would be convenient to be able to build taller stacks without parasitic delay getting out of hand. To do this, some designers have proposed tapering their transistor stacks to use wider transistors near the rails. This leads to an interesting derivation which suggests gate delay really can increase only linearly with stack height. Unfortunately, it overlooks several practical problems that overshadow the potential benefits. Let's look more closely at tapering stacks, then see why it is usually a bad idea.

Figure 5 shows a stack of three transistors with contacts between each. The delay is quadratic in the number of series transistors. Figure 6 shows a tapered stack of three series transistors. The transistors near the bottom of the stack are larger, providing more current to discharge the internal parasitics. With suitable choice of tapering, the delay can be shown to increase only linearly with the number of series transistors.

FIGURE 12. Contacted 3 transistor stack

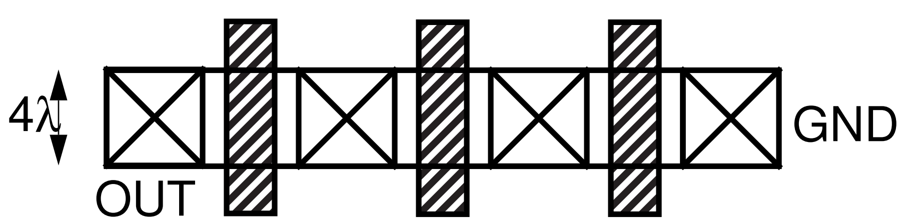
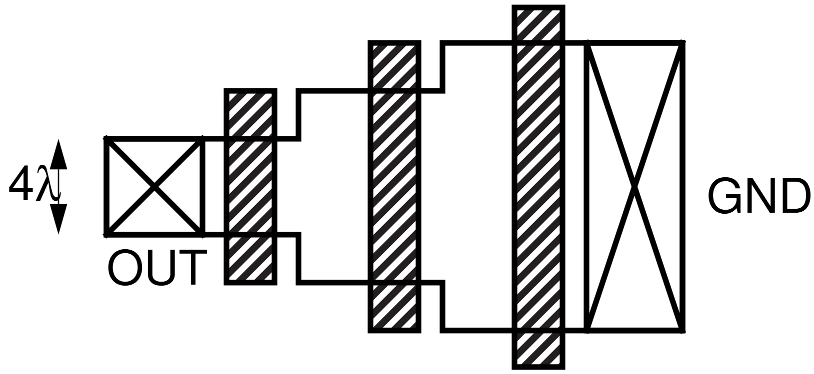


FIGURE 13. Tapered 3 transistor stack

There are two drawbacks with this tapering. One is that the transistors near the rail must be quite large. This increases the loading on previous stages. If the previous stages were also critical, the increased delay of the previous stage may swamp out savings in delay of the tapered stage.



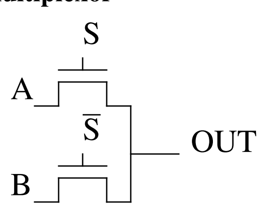
More subtly, the spacing between gates in a tapered stack is larger than the spacing between uncontacted transistors. Therefore, tapering an uncontacted tack can actually increase diffusion parasitics. Thus, it is usually a good idea only to taper stacks at contacted nodes.

Despite these limitations, tapering is occasionally useful. If non-critical early signals can drive transistors close to the rails, widening these transistors is beneficial. Since the signals were early anyway, their load can be increased without overall performance penalty. These wide transistors near the rails can now supply more current when the critical inputs arrive near the middle of the gate. A common application of this is to double the width of the clocked pulldown transistor in a dynamic gate. As long as the clock arrives well before the critical inputs, this will speed up the dynamic gate. A drawback is increased clock loading and power consumption.

9.0 Pass-Transistor Logic

Another popular static circuit family is pass-transistor logic. The idea is to use transistors as switches to efficiently implement multiplexors and related structures. The simplest 2-input multiplexor imaginable can be implemented with just two pass transistors, as shown in Figure 7. It has two major limitations: the output does not swing rail-to-rail, and the output is not buffered so the gate cannot be cascaded to drive arbitrary other gates. These limitations can be overcome by adding explicit buffers and by either employing both NMOS and PMOS transistors in parallel to pass both 0's and 1's effectively or by providing a level restoration circuit on the output. These options lead to several flavors of pass-transistor logic: CPL, DPL, SRPL, LEAP, etc. We'll look at these variations, then consider how to size pass transistors and conclude with recommended applications.

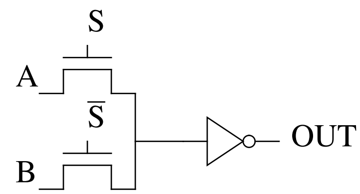
FIGURE 14. Simple Pass-transistor multiplexor



9.1 Pass-transistor Variants

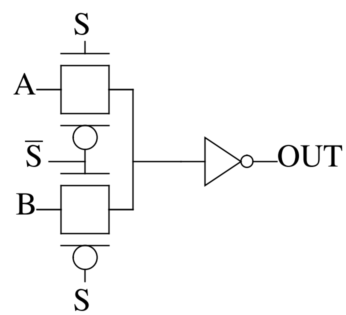
All pass-transistor circuits should end with a buffer (usually an inverter) to decouple the output from the inputs, allowing pass-transistor gates to be cascaded. If the simple pass-transistor multiplexor of Figure 7 were just buffered as shown in Figure 8, the circuit would have power-consumption problems. When the selected input is high, node X will only rise to VDD-Vt. This leaves the PMOS transistor in the inverter right at the border of ON and OFF. Thus, some DC power consumption could be expected. To avoid this, better pass-transistor families pull even the internal nodes rail-to-rail.

FIGURE 15. Buffered pass-transistor multiplexor



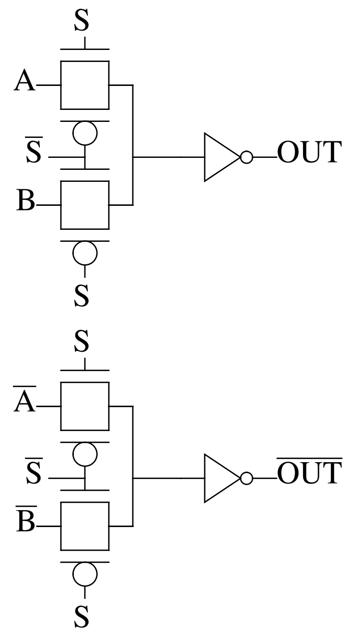
One way to avoid this problem is to gang PMOS transistors in parallel with the NMOS transistors, thus passing both 0's and 1's well, as shown in Figure 9. These parallel structures are called pass-gates or transmission gates.

FIGURE 16. Transmission-Gate multiplexor



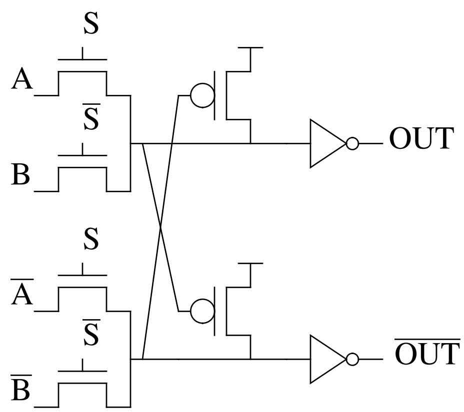
Unfortunately, to cascade such transmission gates, both true and complementary outputs must be available to drive the NMOS and PMOS gate inputs of subsequent stages. This can be done either with inverters, which add extra gates to the critical path, or by building dual-rail logic with accepts both true and complementary inputs and produces both true and complementary outputs. The second scheme is called double pass-transistor logic (DPL) and is illustrated in Figure 10.

FIGURE 17. DPL multiplexor



By now, we've grown from 2 transistors in the simple multiplexor to 12 transistors for DPL! Moreover, the PMOS transistors are slow and add substantial diffusion loading. For most circuits, DPL is not competitive. Since the poor performance from the PMOS transistors, it would be reasonable to remove the PMOS transistors from the transmission gates and instead use a cross-coupled level restoring circuit, as shown in Figure 11. This is known as complementary pass-transistor logic (CPL).

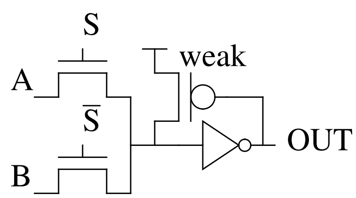
FIGURE 18. CPL multiplexor



CPL is fast because it only uses NMOS transistors in the signal path and because the cross-coupled PMOS devices are fairly fast for level restoration. The circuit is ratioless in the sense that it will eventually produce the correct answer no matter how transistors are sized, but care should be taken sizing the PMOS devices to make them strong enough to be effective, yet weak enough to not slow falling transitions too badly.

All dual-rail logic families require twice as many transistors and wires as the single-rail equivalents. Thus, sometimes, designers may prefer single-rail pass-transistor gates even if extra inverters are required when complementary results are needed. Level restoration can still be performed with feedback, this time taken from the output, as shown in Figure 12. The scheme is called LEAP and produces dense logic. Unfortunately, the feedback PMOS is slower than CPL. Moreover, the circuit is ratioed, meaning that if the PMOS device is too strong, the output may get trapped low and never be able to return high. Thus, the PMOS must be weak enough that it can be overridden even in the SF process corner. Moreover, the resistance of the gate driving inputs A and B, and the wire resistance to these inputs, must be low. Finally, performance gets much worse as VDD drops to a few multiples of the Vt and the threshold drop thus becomes more significant.

FIGURE 19. LEAP multiplexor



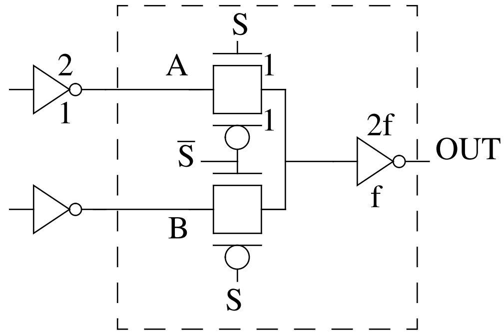
Transmission gates are susceptible to back-driving. Suppose in the circuit above, S=0 and $\overline{S}$ = 1. Let S rise to 1, but suppose that $\overline{S}$ is produced by an inverter and does not immediately switch. During the time that both select lines are 1, node A and node B are connected by a path through the two NMOS transistors. This may cause glitches that could propagate to other circuits which receive signals A and B. This is an especially serious problem if A or B are dynamic nodes or could drive dynamic gates that can’t tolerate glitches. Even when there is no circuit failure, back-driving burns extra power. Thus, it is important to design the select signals to arrive at nearly the same time.

A final caveat of all pass-transistor families is that layout is more difficult and less dense than for static CMOS gates. Routing between source/drain diffusions can increase internal parasitic capacitances and slow gates more than expected.

9.2 Logical Effort of Pass Transistors

Pass-transistors may have inputs to source/drain diffusion as well as to gates. Their logical effort therefore depends not only on the pass-transistor gate itself, but also on the driver. For instance, consider the transmission gate multiplexor of Figure 13, with the A and B inputs driven by inverters.

FIGURE 20. Logical Effort of transmission-gate circuit

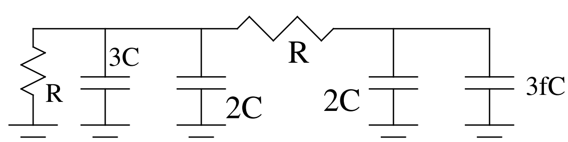


Notice how the PMOS and NMOS transistors are sized equally in the transmission gate. This is because the PMOS is only very important for pulling the output from VDD-Vt to VDD. Sizing it larger would greatly increase diffusion capacitance while only slightly reducing transmission gate resistance.

The resistance of the transmission gate depends on whether it is pulling up or down. Suppose that a transistor pulling the wrong way (i.e. an NMOS transistor pulling high or PMOS transistor pulling low) has on average about twice its usual resistance. Thus, pulling high, the transmission-gate resistance is 2R (NMOS) in parallel with 2R (PMOS) = R. Pulling low, the resistance is R (NMOS) in parallel with 4R (PMOS) = 0.8R. This is close enough that we could approximate the resistance as R in both directions.

Therefore, the equivalent circuit of the input inverter driving the output inverter through the transmission gate is:

FIGURE 21. Equivalent circuit for transmission-gate delay calculation



and the Elmore delay from input A is R(3C+2C) + (R+R)(2C+3fC) = 9+6f τ. To get drive strength equal to a unit inverter, we would have to double the transistors sizes. Thus, the logical effort from input A is 6/3 = 2. The logical effort from input S is only 4/3 because only pass-transistor gates must be driven.

Pass-transistor gates can have substantial internal diffusion capacitances. For gates which use full transmission-gates, cascading more than 2 or 3 transmission gates without buffering causes unacceptably slow switching. CPL gates using only NMOS transistors resemble NMOS series stacks and thus can have 3 or 4 series transistors before slowing greatly.

9.3 Recommendations

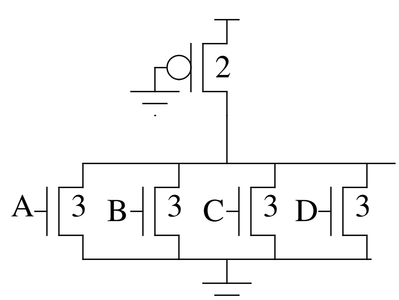
Pass-transistor logic families are primarily beneficial for multiplexors and other circuits that can be built from multiplexors. Examples include XOR gates, alone or in full adders and parity circuits. Wide multiplexors, such as those needed in superscalar execution unit bypass paths, also are efficient to implement with pass transistors. Many papers have been published purporting to prove that the author's favorite pass-transistor logic family offers improved speed and lower power than static CMOS. Such comparisons almost always are based on the delay of full adders, which are implemented well with pass transistors and poorly with static CMOS. Moreover, the comparisons often use suboptimal static CMOS implementations! A fairer comparison finds that for most general circuits, static CMOS is better. Nevertheless, designers should keep pass-transistor logic in their book of tricks and apply it in the circumstances where it is useful.

10.0 Pseudo-NMOS Logic

Once upon a time, NMOS logic ruled the MOS world. Gates were constructed with NMOS pulldowns and resistive pullups. Since actual resistors take large amounts of area in a conventional process, the resistors were built from transistors that were always ON. The NMOS logic gates efficiently implemented NOR functions, in contrast to CMOS logic gates which favor NAND functions. The resistive pullups cause NMOS gates to dissipate DC power when their output is low. This became excessive when tens of thousands of gates were integrated into a single chip, so NMOS fell out of favor and was replaced by CMOS which dissipates almost no DC power.

Nevertheless, the idea of NMOS circuits is still useful for implementing wide NOR gates. The resistive pullup can be built from a PMOS transistor with its gate tied to ground. Such a gate is shown in Figure 15.

FIGURE 22. Pseudo-NMOS NOR4 gate

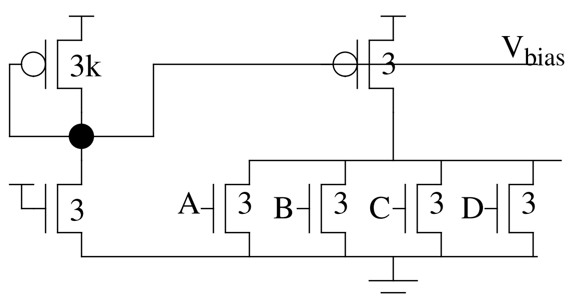


The PMOS transistor must be ratioed such that in the SF corner it is weak enough that the NMOS transistors can pull the output sufficiently low. Sufficiently low depends on the noise margins that can be tolerated. In the FS corner, the PMOS pullup is therefore very slow.

The logical effort is the ratio of input capacitance to that of an inverter with the same drive strength. This depends on whether the gate is pulling up or pulling down. The NMOS transistors produce a current proportional to 3/R. The PMOS draws off a current of roughly 1/R that would otherwise have been available to charge a load. Thus, the total output current is proportional to 3/R - 1/R = 2/R, corresponding to a resistance of R/2. An inverter must be sized twice minimum for this resistance, with an input capacitance of 6. Therefore, the logical effort for falling transitions is 3/6 = 1/2! Unfortunately, the pullup is performed through only a single PMOS transistor with an effective resistance of R. A unit-sized inverter also has resistance R and input capacitance 3. Therefore, the logical effort pulling up is 3/3 = 1. Unless the critical path only involves the pulldown, a conservative designer may size with the larger logical effort. Nevertheless, this logical effort of 1 is still quite good and does not depend on the number of inputs to the NOR gate!

One way to improve the tracking of the P/N ratio across process corners and thus avoid the weak pullup performance is to supply a bias to the PMOS transistor that depends on the processing. This can be done with a current mirror. For example, the current mirror shown in Figure 16 sets the PMOS current to be 1/k that of an NMOS transistor, independent of process. k is usually chosen to be 2-3, depending on desired output low levels. Thus, noise margins will be better in the SF corner and performance will be better in the FS corner.

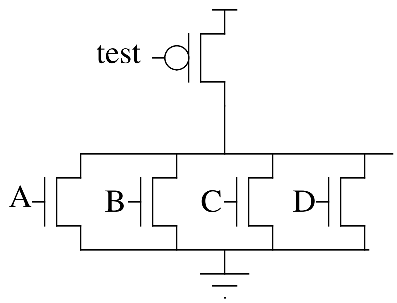
FIGURE 23. Current mirror control of pseudo-NMOS gates



Care must be taken with the bias generator. If a single bias were generated centrally and routed to all pseudo-NMOS gates, the bias would be very susceptible to IR drops, power supply variation, and noise coupled on to the routing. Hence, care must be taken to keep the analog bias voltage stable and the bias must usually be generated locally to avoid power supply noise across the chip.

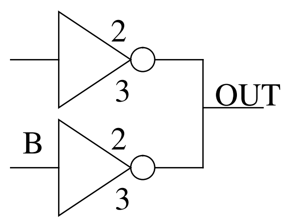
Pseudo-NMOS gates also suffer from DC power consumption. This is a problem for IDDQ test methods which measure the DC current to detect short circuits. One solution is to connect the PMOS gate to a special test signal instead of ground. During IDDQ test, the test signal is asserted high to turn off the PMOS pullups. During normal operation, the test signal stays low.

FIGURE 24. Pseudo-NMOS gate with power-down test mode



A variant on pseudo-NMOS gates is Johnson's symmetric NOR. Such gates consist of two or more inverters with their outputs tied together. When ratioed appropriately, the NMOS transistors can overpower the PMOS transistors. Consider the symmetric 2-input NOR gate in Figure 18. When both inputs are high, the output is low and no power is consumed. When one input is high, the NMOS transistor in one inverter overpowers the PMOS transistor in the other inverter to keep the output nearly low. Some power is consumed. When both outputs fall low, both PMOS transistors turn on, pulling the output high faster than a regular pseudo-NMOS gate could operate. No power is consumed in the high state either.

FIGURE 25. Symmetric 2-input NOR gate



As usual, the logical effort of the gate is the ratio of the input capacitance of the gate to the input capacitance of an inverter with the same drive strength. The worst case is when one input is high and the other is low, causing contention. The effective resistance can be estimated by subtracting the current drawn by the PMOS from the current pulled by the NMOS. The current is thus 3/R - 1/R = 2/R, corresponding to a resistance of R/2. Just to check, the case of both PMOS devices pulling up in parallel also has an effective resistance of R/2. This is the same resistance as a double-sized inverter, with input capacitance 6. Therefore, the logical effort of this NOR gate is 5/6! This is quite good.

It is interesting to observe that with the opposite ratioing such that PMOS transistors always override NMOS transistors, a symmetric NAND gate could be constructed. Symmetric NANDs use such large PMOS transistors that they are not very efficient and should be avoided.

11.0 Summary of Static Circuit Families

In summary, conventional static CMOS gates are the most generally useful circuit family. They can be optimized by choosing good topologies, by placing critical inputs near the center, and by skewing gates where appropriate. Pass-transistors offer advantages for multiplexors and related circuits, such as XOR gates and full adders. Level restoration is the major challenge in using pass-transistors. Pseudo-NMOS gates are good for wide NOR structures where they eliminate the PMOS stack, but suffer from DC power consumption and ratio challenges.