High Speed CMOS VLSI Design

Lecture 2: Logical Effort & Sizing

(c) 1997 David Harris

1.0 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.

1.1 Switching Threshold

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

FIGURE 1. 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.

1.2 Skewing Gates

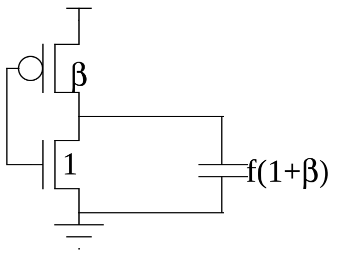
Using a higher or lower P/N ratio favors rising or falling outputs, respectively. For example, with a P/N ratio of 4/1, the input does not have to fall as far as VDD/2 before the output could switch. We call such a circuit a “high skewed” gate and use it on paths where the critical transition is a rising 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.

1.3 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 2. 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:

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

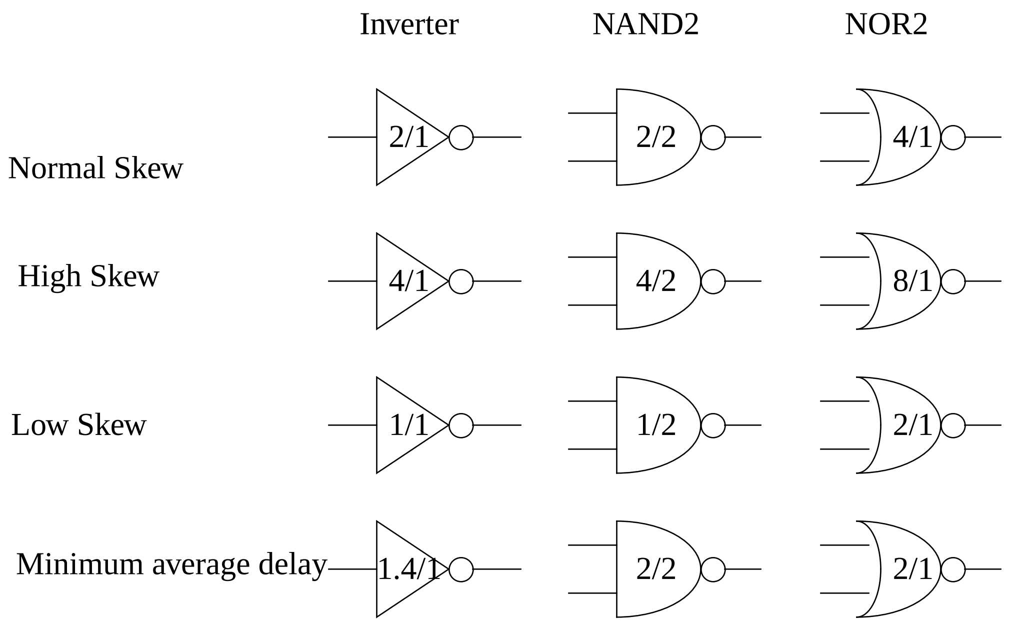
Therefore, the optimal P/N ratio to minimize average path delay is the square root of the ratio that gives equal rise/fall resistances. Since the mobility ratios are 2-3, the best P/N ratios for average delay are 1.4-1.7; 1.5 is a convenient number to use.

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!

1.4 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 3. P/N Ratios



2.0 Logical Effort & Path Sizing

We have seen that a chain of inverters is fastest when it uses a fanout of about 4 per stage. Also, the fanout should be equal for each stage. This idea can be generalized to help size arbitrary paths with gates other than inverters. The key is to replace fanout with a property called gain, which compensates for the reduced drive / unit of input capacitance of more complex gates.

2.1 Terminology

Start with some terminology:

- Block Spec (don’t leave home without it)

  Functionality, Cin, Cout, max delay
- Drive Strength: 1 / (effective resistance of gate)
- Logical Effort (LE)

  Ratio of input capacitance of gate to input capacitance of normal skew inverter with same drive strength.
- Gain: Cout/Cin * LE (or fanout * LE)
- Side Branch

  A load attached to a node on a path but which is not part of the path.
- Capacitance Transformation

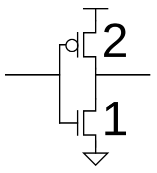
  Process of working from output to input capacitance

  Cin = Cout * LE / Gain

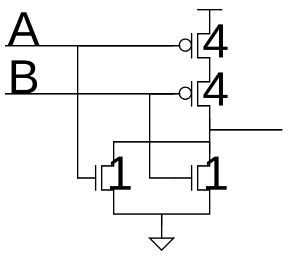
2.2 Logical Effort

Since we already know how to size paths with inverters, we can size paths with other gates by converting the other gate to somehow resemble an inverter. For inverters, we set the fanout of each gate to be identical. For arbitrary gates, we set the gain of each gate to be identical, where gain is the fanout times “logical effort.” Logical effort is the compensating factor of how much more input capacitance the gate presents compared to an inverter; it is defined as the ratio of the input capacitance of a gate to the input capacitance of a normal skew inverter with the same drive strength. Gain and fanout are identical for an inverter because inverters have a logical effort of 1 by definition. A few examples of logical effort calculation are shown in EQ 4.

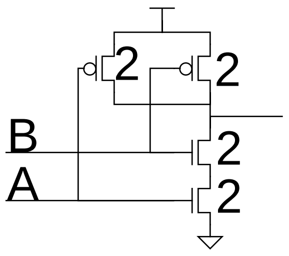
FIGURE 4. Logical Efforts of various gates



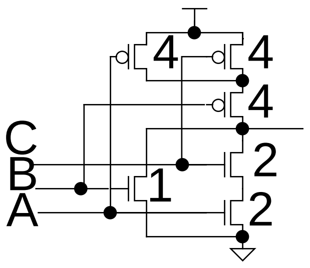
Inverter: LE=1



NOR: LE=5/3



NAND:LE=4/3



AOI: LE=2 for A,C; 5/3 for B

If the topology of the circuit is fixed and there are no side branches, the gain per stage is easy to compute. The logical effort of the entire path is the product of the logical efforts of each stage. For a path that has a final output capacitance Cout and initial input capacitance Cin, the path fanout is Cout/Cin. The gain of the whole path is thus the product of the logical efforts times Cout/Cin. For a path with N stages, the gain per stage is Nth root of the total path gain:

$$\sqrt[n]{\frac{Cout}{Cin}\prod_{stages} LE_i} \qquad \text{(EQ 3)}$$

The objective of sizing with logical effort is to quickly size a path to near optimum delay. Therefore, it is good to be sloppy, keeping results to 1.5 significant figures in your head rather than using a calculator heavily. The relationship between delay and sizing is nearly flat for a design near optimal size, so modest errors in sizing have almost no effect on delay.

2.3 AND Gate Example

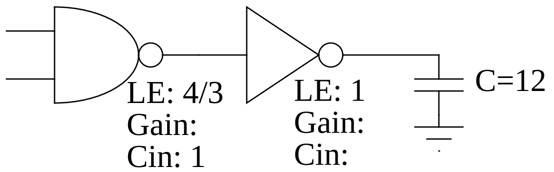
Consider a simple 2-input AND gate as an example for sizing with logical effort. The block specification involves computing the AND function as quickly as possible. The block has a maximum input capacitance of one unit (where unit could be anything, such as 12 λ of gate capacitance) and must drive a 12 unit load. The topology selected is a NAND2 gate followed by an inverter. The steps for sizing the path are:

- Sketch the path and label each gate with logical effort LE

- Compute or guess a gain per stage
- Work backward from the output assigning gain to each gate and doing a capacitance transformation to deduce the input capacitance

The path is shown in Figure 5:

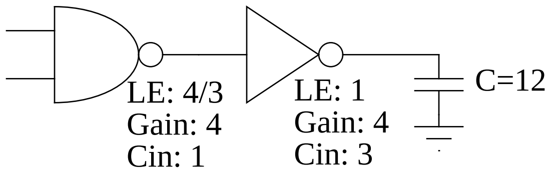
FIGURE 5. AND gate path



In this case, computing the gain per stage is straightforward. The fanout of the entire block is 12/1 = 12. The logical effort of the entire path is 4/3 * 1 = 4/3. Thus, the gain of the path is fanout times logical effort, or 16. The gain per stage is the square root of the path gain since there are two stages, i.e. 4.

Now we assign the gain of 4 to the inverter and perform a capacitance transformation to compute the input capacitance of the inverter: Cin = Cout * LE / Gain = 12 * 1 / 4 = 3. Finally, we assign the gain of 4 to the NAND gate and perform a capacitance to sanity check our sizing, concluding the NAND gate has Cin = Cout * LE / Gain = 3 * 4/3 / 4 = 1, agreeing with the specification:

FIGURE 6. AND gate path (completed)



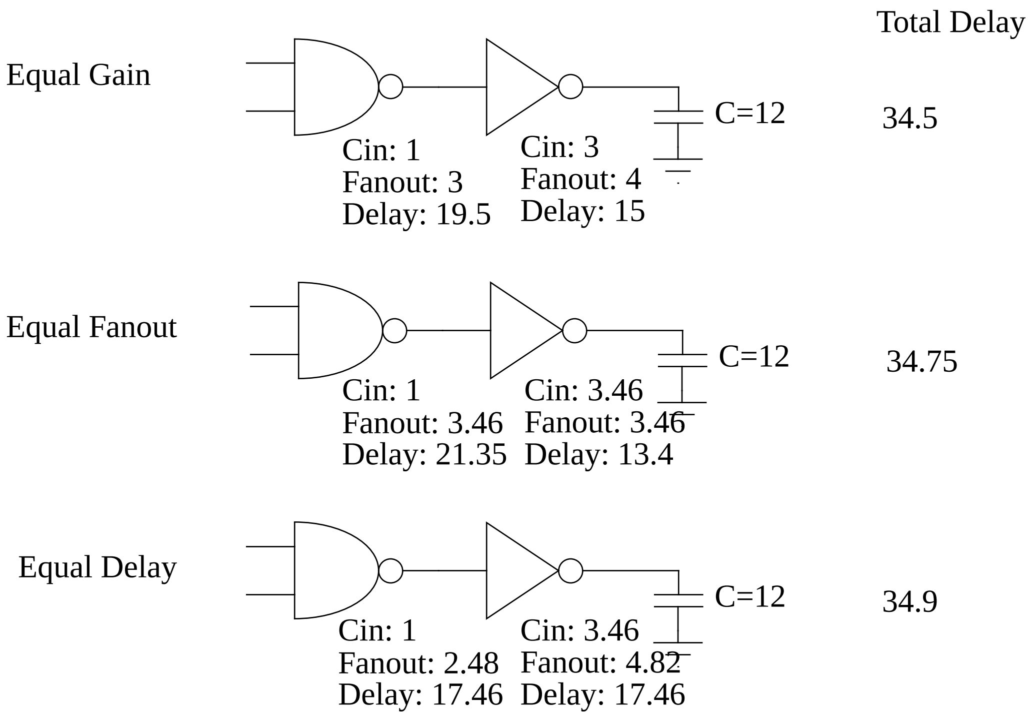
Once input capacitances have been assigned, we need to map the gates to actual PMOS and NMOS transistor sizes. If we have a cell library available, we can select the closest sizes from the library. For example, select a NAND gate from the library with close to 1 unit of capacitance and an inverter with close to 3 units of capacitance. Actual delay is a weak function of size as we have seen so even if the library contains cells substantially different in size than the desired sizes, delay will still be close to optimal. This is how synthesis programs produce tolerable results with a very coarse set of gate sizes in their library. If a cell library is not available, assign P/N ratios based on the desired skew of the gate. For instance, suppose the unit of capacitance in this example corresponds to 20 λ of gate capacitance. The NAND gate with 1 unit of input capacitance would use 10 λ NMOS and 10 λ PMOS transistors. The inverter with 3 units of input capacitance would use 20 λ NMOS and 40 λ PMOS transistors. If units were larger, all transistors would be proportionally larger and delays would remain the same.

2.4 Alternate sizing techniques

Many designers are not familiar with logical effort but have other rules of thumb for sizing paths. One rule is to use equal fanout per stage; another is to use equal delay per stage. All three rules are equivalent for paths consisting of inverters, but give different results for paths with a mix of gates. For comparison, look at what each of the rules produces for the inverter size, for the delay of each gate, and for the overall path delay.

We estimate that the delay of a fanout of f inverter is 3+3f and fanout of f NAND2 is 7.5 + 4f, as given by RC models. For equal fanout per stage, each gate must have a fanout of sqrt(12). For equal delay per stage, we solve $7.5+4f_1 = 3+3f_2$, subject to $f_1f_2 = 12$. For equal gain per stage, we have already solved the problem above. The results are summarized in Figure 7:

FIGURE 7. Different Sizing Methodologies

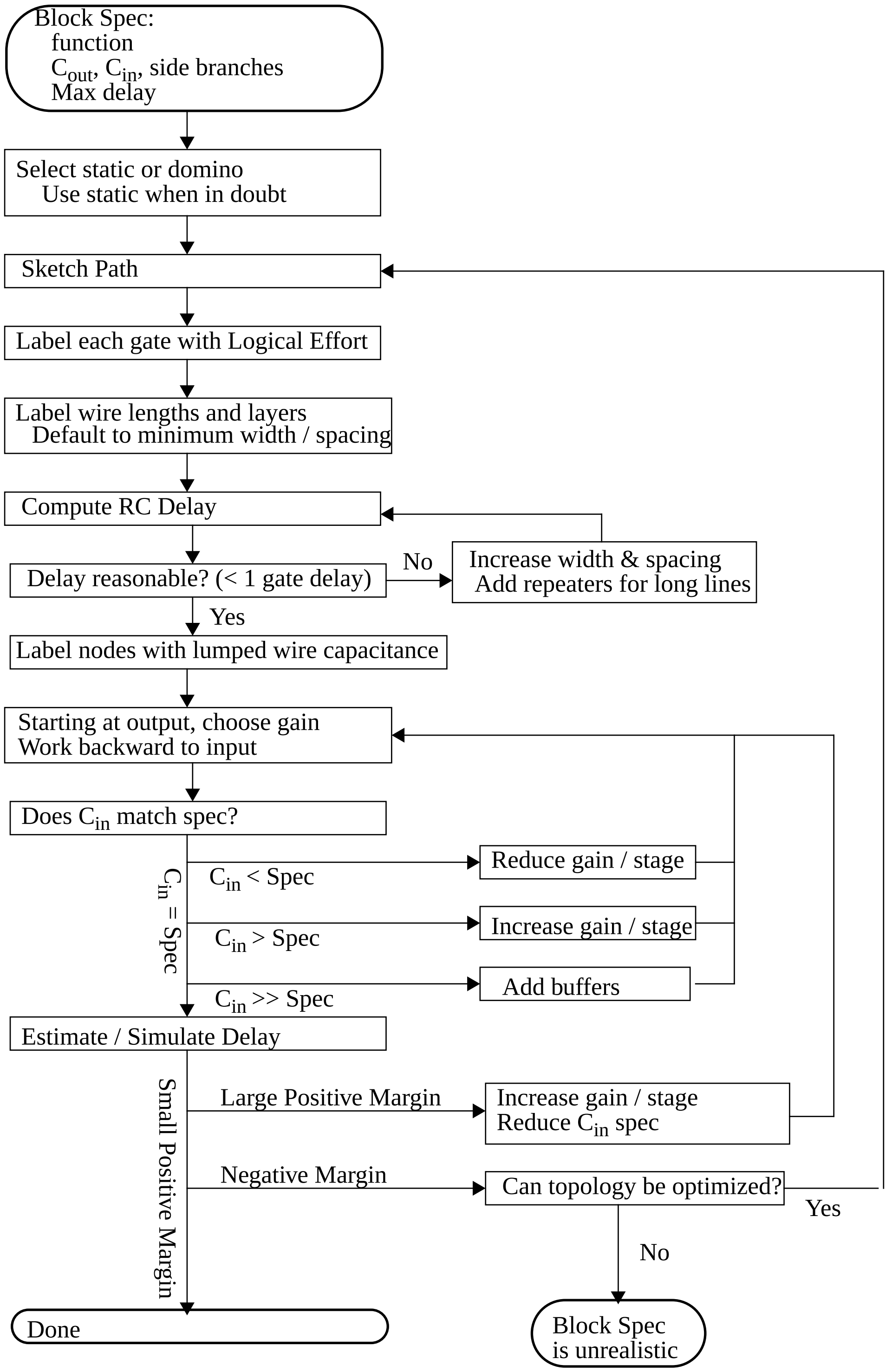


Equal gain per stage can be proven to minimize overall path delay. Equal fanout per stage is the easiest to manually compute because only sizes need to be known. Equal delay per stage is easy for CAD tools to do because the tools only need to deal with delays. The equal fanout and equal delay methods are sometimes used because total delay is a weak function of actual sizes near the optimum; as the figure shows, substantially different fanouts all give within 2% of optimal delay. Even for very complex paths with a wide range of logical efforts, the alternate methods are generally within 5-10% of optimum. Nevertheless, we will emphasize equal gain sizing because it is theoretically best and gives intuition about the relative complexities of gates and a model of why gates with lower logical efforts run faster. However, we will exploit the weak relationship between exact size and total delay to round to convenient integer sizes and keep math simple.

2.5 Sizing Flow

Now we can generalize the sizing process to a flowchart representing the entire path design steps:

FIGURE 8. Sizing Flow Chart



The design flow starts with a block specification for the task. A circuit style should be selected; domino circuits offer higher speed but also take longer to design and usually dissipate more power. Therefore, it is usually wise to try a static design as long as the static design can meet specification. Sketch a network of gates that implements the desired function and annotate it with long wires. Also mark the logical effort of each gate. Gate sizing depends on the wires driven by the gates, so it is best to design the wires first. Estimate the RC delay of the wire given a predicted wire pitch and layer, as we will describe later. If the delay is short, accept it. If the delay is significant, consider adjusting the pitch or inserting repeaters to reduce the delay. Once the wires are sized, they may be treated as lumped wire capacitance for sizing.

Now apply logical effort to size the path. Estimate the gain per stage, either by computing the total gain of a path if the path is simple or by making a guess if the path is too complex. Work from the output backward doing capacitance transformations to find the input capacitance of each gate, until the first gate is reached. Check if the input capacitance meets specification. If the input capacitance is lower than specification, the gain per stage may be reduced, speeding up the circuit. If the capacitance is higher than specification, the gain per stage must be increased. If the input capacitance is much higher than specification, requiring gains per stage much larger than 6 to meet specification, the path can be sped up by inserting buffers at the end, increasing the number of stages and therefore reducing the gain per stage. Optimum delay is generally around a gain of 4 per stage; if the gain is outside the range of 2-6, the path probably has too many or too few stages and performance may be improved by adding buffers (for high gains) or collapsing several simple gates into a single more complex gate (for low gains) when practical.

Once the input capacitance spec is met, simulate or estimate the delay of the circuit. In the lucky event that there is positive margin, i.e. the circuit is faster than need be, consider increasing the gain per stage of the final stage(s) to reduce the sizes and power consumption of the block while slowing down the circuit. If there is negative margin, the designer's first impulse may be to upsize gates to try increasing speed. This is a tragic mistake because the circuit has already been sized with logical effort to achieve maximum speed. Increasing the size of an internal gate will speed up the gate, but increase the load on the previous gate causing a net loss of performance. Increasing the input capacitance of the block can speed up the entire block but only passes the problem to another designer who now has a larger output capacitance to drive, and also increases area and power of the block. Instead of further tweaking gate sizes, the designer should examine the circuit topology and try to optimize it or switch circuit styles. If the topology is thoroughly optimized and the delay target cannot be reached, either the delay target or the functional spec of the module must be changed; sizing cannot help. Hopefully, however, an optimized block meets timing goals and the task is done.

2.6 Priority Encoder Example

Consider the fifth bit of a priority encoder as another example. The function of the encoder is to produce a high output if the fifth input is true and all four previous bits are false. This function is illustrated logically in Figure 9. The output load is 200 λ of gate capacitance. The maximum input load on any input is 20 λ of gate capacitance. Minimize delay. A more practical CMOS implementation of the path is shown in Figure 10, labeled with logical efforts of each gate.

FIGURE 9. Logical representation of fifth bit of priority encoder5/3

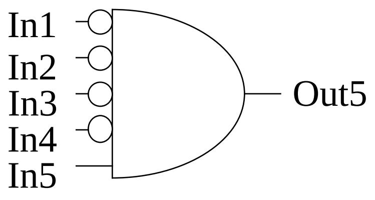
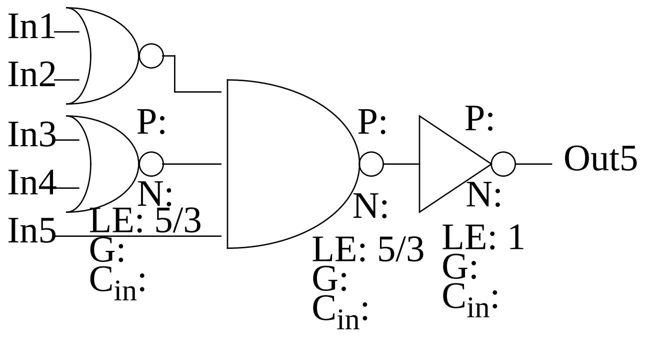
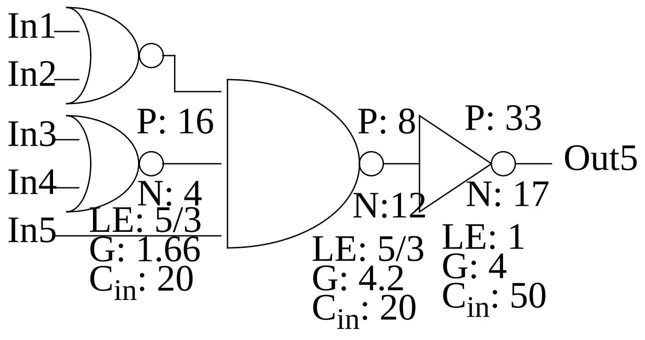


FIGURE 10. Practical CMOS implementation of fifth bit of priority encoder



Since In5 is not buffered, it sets a limit that the NAND3 gate must have an input capacitance of 20 λ to meet the block spec. Thus, we can compute the total gain from In5 to Out5: Cout/Cin * logical efforts of the NAND3 and inverter = 200 / 20 * 5/3 * 1 = 50/3, or about 16. Therefore, the gain per stage through the two gates is 4. Doing a capacitance transformation sets the invert input capacitance to be Cout * LE / Gain = 200 * 1 / 4 = 50. Finally, the NOR gates can be sized. They are sized as large as possible given the input capacitance spec, at 20. Finally, we can assign PMOS and NMOS sizes assuming normal skew gates. The result is shown in Figure 11:

FIGURE 11. Fully sized priority encoder

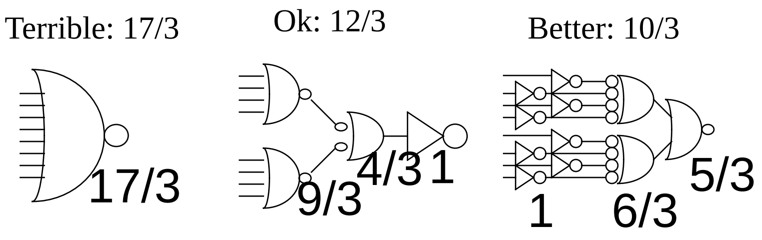


This example shows how constraints on the capacitance of certain nodes in a path cause different gates to necessarily have different gains. If the capacitance constrain on In5 were lifted, the delay from In1 to Out5 could be improved by making each stage have an equal gain (of about 3). Side loads also impose different gains for different gates and make sizing an iterative process for most circuits complex enough to be interesting.

2.7 Conclusions

We have seen how to use logical effort to size paths for minimal delay. With some practice, the technique becomes fast and easier than tweaking sizes in a circuit simulator. However, it is important not to get wrapped up in optimal sizing and lose the bigger picture. The topology of a circuit is more important than the exact sizing. For instance, it is good to choose gates with a low logical effort from the start, as illustrated in 3 ways of building an 8-input NOR gate:

FIGURE 12. 8-input NOR gate designs



In general, NANDs are better than NORs in static CMOS. Consider using domino, transmission gates, or other appropriate circuit styles. Layout issues also greatly impact delay; minimizing diffusion capacitance and floorplanning well to shorten critical interconnect makes the difference between high performance custom designs and slow place & routed designs. Finally, remember not to let final driver stages powering huge loads get too large. The huge drivers impact area and power while bringing very little speed gain; frequently it is a wiser tradeoff to use a gain of 8 or even 12 on the final driver of a huge load, but thereby reduce the gain of the previous stages in the path and only lose a small amount of speed.