To illustrate the effect of the bootstrap capacitance on a circuit, Figure 4.28(b) shows two inverter pairs. The top pair has an extra bit of capacitance between the input and output of the second inverter. The bottom pair has the same amount of extra capacitance from input to ground. When *x* falls, nodes *a* and *c* begin to rise (Figure 4.28(c)). At first, both nodes see approximately the same capacitance, consisting of the two transistors and the extra 3 fF. As node *a* rises, it initially bumps up *b* or "lifts *b* by its own bootstraps." Eventually the nMOS transistors turn ON, pulling down *b* and *d*. As *b* falls, it tugs on *a* through the capacitor, leading to the slow final transition visible on node *a*. Also observe that *b* falls later than *d* because of the extra charge that must be supplied to discharge the bootstrap capacitor. In summary, the extra capacitance has a greater effect when connected between input and output as compared to when it is connected between input and ground.

Because C_{gd} is fairly small, bootstrapping is only a mild annoyance in digital circuits. However, if the inverter is biased in its linear region near $V_{DD}/2$, the C_{gd} is multiplied by the large gain of the inverter. This is known as the *Miller effect* and is of major importance in analog circuits.

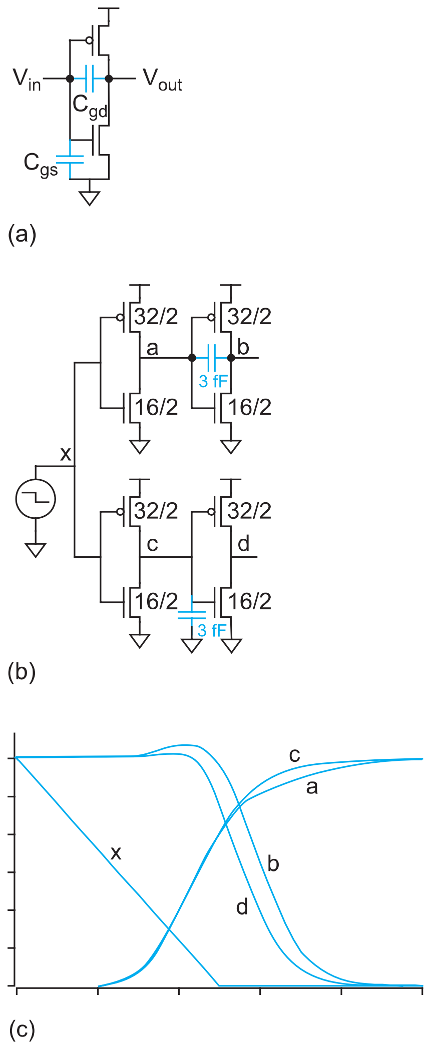


FIGURE 4.28 The effect of bootstrapping on inverter delay and waveform shape

4.5 Logical Effort of Paths

Designers often need to choose the fastest circuit topology and gate sizes for a particular logic function and to estimate the delay of the design. As has been stated, simulation or timing analysis are poor tools for this task because they only determine how fast a particular implementation will operate, not whether the implementation can be modified for better results and if so, what to change. Inexperienced designers often end up in the "simulate and tweak" loop involving minor changes and many fruitless simulations. The method of Logical Effort [Sutherland99] provides a simple method "on the back of an envelope" to choose the best topology and number of stages of logic for a function. Based on the linear delay model, it allows the designer to quickly estimate the best number of stages for a path, the minimum possible delay for the given topology, and the gate sizes that achieve this delay. The techniques of Logical Effort will be revisited throughout this text to understand the delay of many types of circuits.

4.5.1 Delay in Multistage Logic Networks

Figure 4.29 shows the logical and electrical efforts of each stage in a multistage path as a function of the sizes of each stage. The path of interest (the only path in this case) is marked with the dashed blue line. Observe that logical effort is independent of size, while electrical effort depends on sizes. This section develops some metrics for the path as a whole that are independent of sizing decisions.

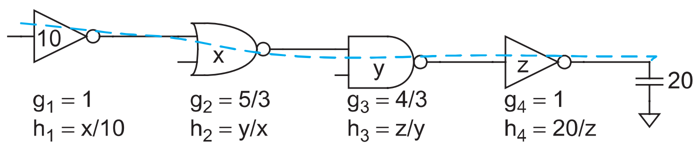


FIGURE 4.29 Multistage logic network

The *path logical effort* G can be expressed as the products of the logical efforts of each stage along the path.

$$G = \prod g_i \tag{4.32}$$

The *path electrical effort* H can be given as the ratio of the output capacitance the path must drive divided by the input capacitance presented by the path. This is more convenient than defining path electrical effort as the product of stage electrical efforts because we do not know the individual stage electrical efforts until gate sizes are selected.

$$H = \frac{C_{\text{out(path)}}}{C_{\text{in(path)}}} \tag{4.33}$$

The *path effort* F is the product of the stage efforts of each stage. Recall that the stage effort of a single stage is $f = gh$. Can we by analogy state $F = GH$ for a path?

$$F = \prod f_i = \prod g_i h_i \tag{4.34}$$

In paths that branch, $F \neq GH$. This is illustrated in Figure 4.30, a circuit with a two-way branch. Consider a path from the primary input to one of the outputs. The path logical effort is $G = 1 \times 1 = 1$. The path electrical effort is $H = 90/5 = 18$. Thus, $GH = 18$. But $F = f_1 f_2 = g_1 h_1 g_2 h_2 = 1 \times 6 \times 1 \times 6 = 36$. In other words, $F = 2GH$ in this path on account of the two-way branch.

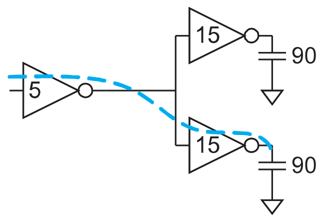


FIGURE 4.30 Circuit with two-way branch

We must introduce a new kind of effort to account for branching between stages of a path. This *branching effort* b is the ratio of the total capacitance seen by a stage to the capacitance on the path; in Figure 4.30 it is $(15 + 15)/15 = 2$.

$$b = \frac{C_{\text{onpath}} + C_{\text{offpath}}}{C_{\text{onpath}}} \tag{4.35}$$

The *path branching effort* B is the product of the branching efforts between stages.

$$B = \prod b_i \tag{4.36}$$

Now we can define the path effort F as the product of the logical, electrical, and branching efforts of the path. Note that the product of the electrical efforts of the stages is actually BH, not just H.

$$F = GBH \tag{4.37}$$

We can now compute the delay of a multistage network. The *path delay* D is the sum of the delays of each stage. It can also be written as the sum of the *path effort delay* D_F and *path parasitic delay* P:

$$\begin{aligned} D &= \sum d_i = D_F + P \\ D_F &= \sum f_i \\ P &= \sum p_i \end{aligned} \tag{4.38}$$

The product of the stage efforts is F, independent of gate sizes. The path effort delay is the sum of the stage efforts. The sum of a set of numbers whose product is constant is minimized by choosing all the numbers to be equal. In other words, the path delay is minimized when each stage bears the same effort. If a path has N stages and each bears the same effort, that effort must be

$$\hat{f} = g_i h_i = F^{1/N} \tag{4.39}$$

Thus, the minimum possible delay of an N-stage path with path effort F and path parasitic delay P is

$$D = NF^{1/N} + P \tag{4.40}$$

This is a key result of Logical Effort. It shows that the minimum delay of the path can be estimated knowing only the number of stages, path effort, and parasitic delays without the need to assign transistor sizes. This is superior to simulation, in which delay depends on sizes and you never achieve certainty that the sizes selected are those that offer minimum delay.

It is also straightforward to select gate sizes to achieve this least delay. Combining EQs (4.21) and (4.22) gives us the *capacitance transformation* formula to find the best input capacitance for a gate given the output capacitance it drives.

$$C_{\text{in}_i} = \frac{C_{\text{out}_i} \times g_i}{\hat{f}} \tag{4.41}$$

Starting with the load at the end of the path, work backward applying the capacitance transformation to determine the size of each stage. Check the arithmetic by verifying that the size of the initial stage matches the specification.

Example 4.13

Estimate the minimum delay of the path from A to B in Figure 4.31 and choose transistor sizes to achieve this delay. The initial NAND2 gate may present a load of 8 λ of transistor width on the input and the output load is equivalent to 45 λ of transistor width.

SOLUTION: The path logical effort is $G = (4/3) \times (5/3) \times (5/3) = 100/27$. The path electrical effort is $H = 45/8$. The path branching effort is $B = 3 \times 2 = 6$. The path effort is $F = GBH = 125$. As there are three stages, the best stage effort is $\hat{f} = \sqrt[3]{125} = 5$. The path parasitic delay is $P = 2 + 3 + 2 = 7$. Hence, the minimum path delay is $D = 3 \times 5 + 7 = 22$ in units of τ, or 4.4 FO4 inverter delays. The gate sizes are computed with the capacitance transformation from EQ (4.41) working backward along the path: $y = 45 \times (5/3)/5 = 15$. $x = (15 + 15) \times (5/3)/5 = 10$. We verify that the initial 2-input NAND gate has the specified size of $(10 + 10 + 10) \times (4/3)/5 = 8$. The transistor sizes in Figure 4.32 are chosen to give the desired amount of input capacitance while achieving equal rise and fall delays. For example, a 2-input NOR gate should have a 4:1 P/N ratio. If the total input capacitance is 15, the pMOS width must be 12 and the nMOS width must be 3 to achieve that ratio.

We can also check that our delay was achieved. The NAND2 gate delay is $d_1 = g_1 h_1 + p_1 = (4/3) \times (10 + 10 + 10)/8 + 2 = 7$. The NAND3

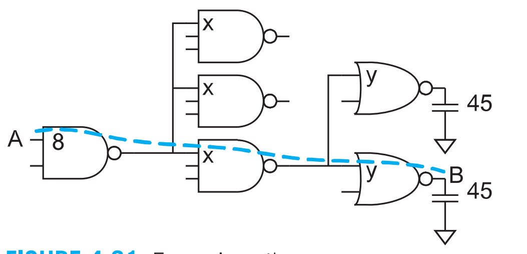


FIGURE 4.31 Example path

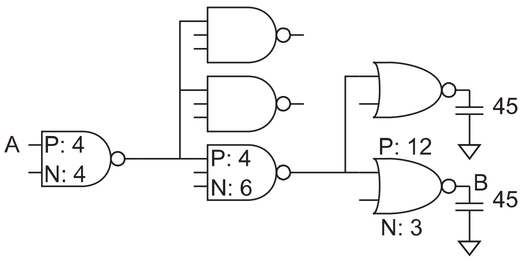


FIGURE 4.32 Example path annotated with transistor sizes

gate delay is $d_2 = g_2h_2 + p_2 = (5/3) \times (15 + 15)/10 + 3 = 8$. The NOR2 gate delay is $d_3 = g_3h_3 + p_3 = (5/3) \times 45/15 + 2 = 7$. Hence, the path delay is 22, as predicted.

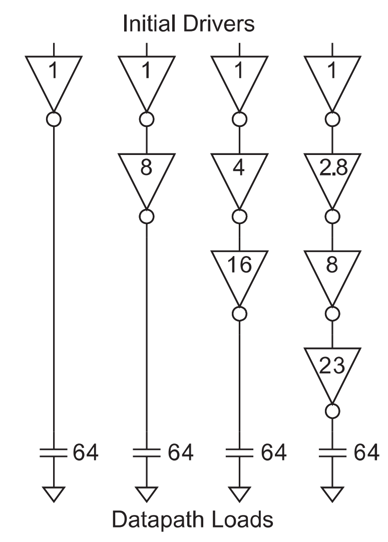
Recall that delay is expressed in units of τ. In a 65 nm process with $\tau = 3$ ps, the delay is 66 ps. Alternatively, a fanout-of-4 inverter delay is 5τ, so the path delay is 4.4 FO4s.

Many inexperienced designers know that wider transistors offer more current and thus try to make circuits faster by using bigger gates. Increasing the size of any of the gates except the first one only makes the circuit slower. For example, increasing the size of the NAND3 makes the NAND3 faster but makes the NAND2 slower, resulting in a net speed loss. Increasing the size of the initial NAND2 gate does speed up the circuit under consideration. However, it presents a larger load on the path that computes input A, making that path slower. Hence, it is crucial to have a specification of not only the load the path must drive but also the maximum input capacitance the path may present.

4.5.2 Choosing the Best Number of Stages

Given a specific circuit topology, we now know how to estimate delay and choose gate sizes. However, there are many different topologies that implement a particular logic function. Logical Effort tells us that NANDs are better than NORs and that gates with few inputs are better than gates with many. In this section, we will also use Logical Effort to predict the best number of stages to use.

Logic designers sometimes estimate delay by counting the number of stages of logic, assuming each stage has a constant "gate delay." This is potentially misleading because it implies that the fastest circuits are those that use the fewest stages of logic. Of course, the gate delay actually depends on the electrical effort, so sometimes using fewer stages results in more delay. The following example illustrates this point.



N:	1	2	3	4
f:	64	8	4	2.8
D:	65	18	15	15.3

Fastest

FIGURE 4.33 Comparison of different number of stages of buffers

Example 4.14

A control unit generates a signal from a unit-sized inverter. The signal must drive unit-sized loads in each bitslice of a 64-bit datapath. The designer can add inverters to buffer the signal to drive the large load. Assuming polarity of the signal does not matter, what is the best number of inverters to add and what delay can be achieved?

SOLUTION: Figure 4.33 shows the cases of adding 0, 1, 2, or 3 inverters. The path electrical effort is $H = 64$. The path logical effort is $G = 1$, independent of the number of inverters. Thus, the path effort is $F = 64$. The inverter sizes are chosen to achieve equal stage effort. The total delay is $D = N\sqrt[N]{64} + N$.

The 3-stage design is fastest and far superior to a single stage. If an even number of inversions were required, the two- or four-stage designs are promising. The four-stage design is slightly faster, but the two-stage design requires significantly less area and power.

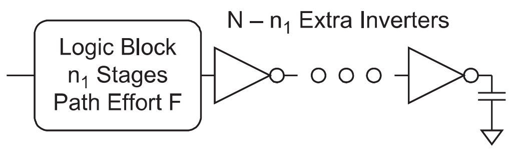


FIGURE 4.34 Logic block with additional inverters

In general, you can always add inverters to the end of a path without changing its function (save possibly for polarity). Let us compute how many should be added for least delay. The logic block shown in Figure 4.34 has n_1 stages and a path effort of F. Consider adding $N - n_1$ inverters to the end to bring the path to N stages. The extra inverters do not change the path logical effort but do add parasitic delay. The delay of the new path is

$$D = NF^{1/N} + \sum_{i=1}^{n_1} p_i + \left(N - n_1\right) p_{\text{inv}} \tag{4.42}$$

Differentiating with respect to N and setting to 0 allows us to solve for the best number of stages, which we will call $\hat{N}$. The result can be expressed more compactly by defining

$$\rho = F^{1/\hat{N}}$$

to be the best stage effort.

$$\begin{aligned} \frac{\partial D}{\partial N} &= -F^{1/N} \ln F^{1/N} + F^{1/N} + p_{\text{inv}} = 0 \\ &\Rightarrow p_{\text{inv}} + \rho\left(1 - \ln \rho\right) = 0 \end{aligned} \tag{4.43}$$

EQ (4.43) has no closed form solution. Neglecting parasitics (i.e., assuming $p_{\text{inv}} = 0$), we find the classic result that the stage effort $\rho = 2.71828$ (e) [Mead80]. In practice, the parasitic delays mean each inverter is somewhat more costly to add. As a result, it is better to use fewer stages, or equivalently a higher stage effort than e. Solving numerically, when $p_{\text{inv}} = 1$, we find $\rho = 3.59$.

A path achieves least delay by using $\hat{N} = \log_\rho F$ stages. It is important to understand not only the best stage effort and number of stages but also the sensitivity to using a different number of stages. Figure 4.35 plots the delay increase using a particular number of stages against the total number of stages, for $p_{\text{inv}} = 1$. The x-axis plots the ratio of the actual number of stages to the ideal number. The y-axis plots the ratio of the actual delay to the best achievable. The curve is flat around the optimum. The delay is within 15% of the best achievable if the number of stages is within 2/3 to 3/2 times the theoretical best number (i.e., ρ is in the range of 2.4 to 6).

Using a stage effort of 4 is a convenient choice and simplifies mentally choosing the best number of stages. This effort gives delays within 2% of minimum for p_{inv} in the range of 0.7 to 2.5. This further explains why a fanout-of-4 inverter has a "representative" logic gate delay.

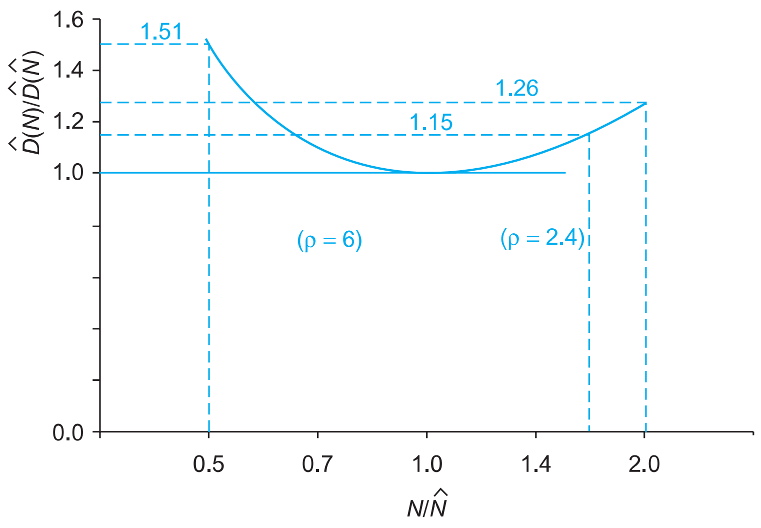


FIGURE 4.35 Sensitivity of delay to number of stages

4.5.3 Example

Consider a larger example to illustrate the application of Logical Effort. Our esteemed colleague Ben Bitdiddle is designing a decoder for a register file in the Motoroil 68W86, an embedded processor for automotive applications. The decoder has the following specifications:

- 16-word register file
- 32-bit words
- Each register bit presents a load of three unit-sized transistors on the word line (two unit-sized access transistors plus some wire capacitance)
- True and complementary versions of the address bits $A[3:0]$ are available
- Each address input can drive 10 unit-sized transistors

As we will see further in Section 12.2.2, a 2^N-word decoder consists of 2^N N-input AND gates. Therefore, the problem is reduced to designing a suitable 4-input AND gate. Let us help Ben determine how many stages to use, how large each gate should be, and how fast the decoder can operate.

The output load on a word line is 32 bits with three units of capacitance each, or 96 units. Therefore, the path electrical effort is $H = 96/10 = 9.6$. Each address is used to compute half of the 16 word lines; its complement is used for the other half. Therefore, a $B = 8$-way branch is required somewhere in the path. Now we are faced with a chicken-and-egg dilemma. We need to know the path logical effort to calculate the path effort and best number of stages. However, without knowing the best number of stages, we cannot sketch a path and determine the logical effort for that path. There are two ways to resolve the dilemma. One is to sketch a path with a random number of stages, determine the path logical effort, and then use that to compute the path effort and the actual number of stages. The path can be redesigned with this number of stages, refining the path logical effort. If the logical effort changes significantly, the process can be repeated. Alternatively, we know that the logic of a decoder is rather simple, so we can ignore the logical effort (assume $G = 1$). Then we can proceed with our design, remembering that the best number of stages is likely slightly higher than predicted because we neglected logical effort.

Taking the second approach, we estimate the path effort is $F = GBH = (1)(8)(9.6) = 76.8$. Targeting a best stage effort of $\rho = 4$, we find the best number of stages is $N = \log_4 76.8 = 3.1$. Let us select a 3-stage design, recalling that a 4-stage design might be a good choice too when logical effort is considered. Figure 4.36 shows a possible 3-stage design (INV-NAND4-INV).

The path has a logical effort of $G = 1 \times (6/3) \times 1 = 2$, so the actual path effort is $F = (2)(8)(9.6) = 154$. The stage effort is $\hat{f} = 154^{1/3} = 5.36$. This is in the reasonable range of 2.4 to 6, so we expect our design to be acceptable. Applying the capacitance transformation, we find gate sizes $z = 96 \times 1/5.36 = 18$ and $y = 18 \times 2 /5.36 = 6.7$. The delay is $3 \times 5.36 + 1 + 4 + 1 = 22.1$.

Logical Effort also allows us to rapidly compare alternative designs using a spreadsheet rather than a schematic editor and a large number of simulations. Table 4.4 compares a number of alternative designs. We find a 4-stage design is somewhat faster, as we suspected. The 4-stage NAND2-INV-NAND2-INV design not only has the theoretical best number of stages, but also uses simpler 2-input gates to reduce the logical effort and parasitic delay to obtain a 12% speedup over the original design. However, the 3-stage design has a smaller total gate area and dissipates less power.

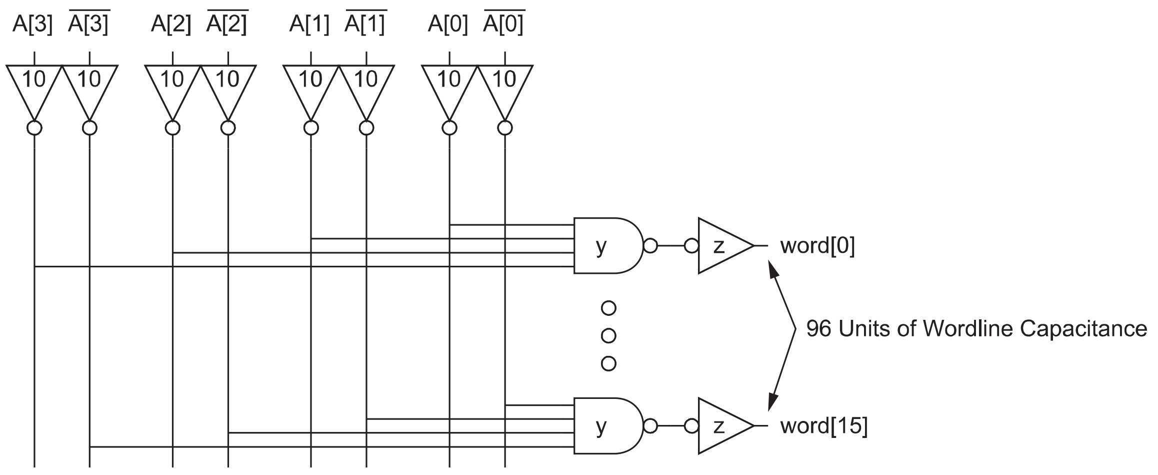


FIGURE 4.36 3-stage decoder design

TABLE 4.4 Spreadsheet comparing decoder designs

Design	Stages N	G	P	D
NAND4-INV	2	2	5	29.8
NAND2-NOR2	2	20/9	4	30.1
INV-NAND4-INV	**3**	**2**	**6**	**22.1**
NAND4-INV-INV-INV	4	2	7	21.1
NAND2-NOR2-INV-INV	4	20/9	6	20.5
NAND2-INV-NAND2-INV	4	16/9	6	19.7
INV-NAND2-INV-NAND2-INV	5	16/9	7	20.4
NAND2-INV-NAND2-INV-INV-INV	6	16/9	8	21.6

4.5.4 Summary and Observations

Logical Effort provides an easy way to compare and select circuit topologies, choose the best number of stages for a path, and estimate path delay. The notation takes some time to become natural, but this author has poured through all the letters in the English and Greek alphabets without finding better notation. It may help to remember d for "**d**elay," p for "**p**arasitic," b for "**b**ranching," f for "e**f**fort," g for "lo**g**ical effort" (or perhaps **g**ain), and h as the next letter after "f" and "g." The notation is summarized in Table 4.5 for both stages and paths.

The method of Logical Effort is applied with the following steps:

1. Compute the path effort: $F = GBH$
2. Estimate the best number of stages: $\hat{N} = \log_4 F$
3. Sketch a path using: $\hat{N}$ stages
4. Estimate the minimum delay: $D = \hat{N}F^{1/\hat{N}} + P$
5. Determine the best stage effort: $\hat{f} = F^{1/\hat{N}}$
6. Starting at the end, work backward to find sizes: $C_{\text{in}_i} = \dfrac{C_{\text{out}_i} \times g_i}{\hat{f}}$

TABLE 4.5 Summary of Logical Effort notation

Term	Stage Expression	Path Expression
number of stages	1	N
logical effort	g (see Table 4.2)	$G = \prod g_i$
electrical effort	$h = \frac{C_{\text{out}}}{C_{\text{in}}}$	$H = \frac{C_{\text{out(path)}}}{C_{\text{in(path)}}}$
branching effort	$b = \frac{C_{\text{onpath}} + C_{\text{offpath}}}{C_{\text{onpath}}}$	$B = \prod b_i$
effort	$f = gh$	$F = GBH$
effort delay	f	$D_F = \sum f_i$
parasitic delay	p (see Table 4.3)	$P = \sum p_i$
delay	$d = f + p$	$D = \sum d_i = D_F + P$

CAD tools are very fast and accurate at evaluating complex delay models, so Logical Effort should not be used as a replacement for such tools. Rather, its value arises from "quick and dirty" hand calculations and from the insights it lends to circuit design. Some of the key insights include:

- The idea of a numeric "logical effort" that characterizes the complexity of a logic gate or path allows you to compare alternative circuit topologies and show that some topologies are better than others.
- NAND structures are faster than NOR structures in static CMOS circuits.
- Paths are fastest when the effort delays of each stage are about the same and when these delays are close to four.
- Path delay is insensitive to modest deviations from the optimum. Stage efforts of 2.4–6 give designs within 15% of minimum delay. There is no need to make calculations to more than 1–2 significant figures, so many estimations can be made in your head. There is no need to choose transistor sizes exactly according to theory and there is little benefit in tweaking transistor sizes if the design is reasonable.
- Using stage efforts somewhat greater than 4 reduces area and power consumption at a slight cost in speed. Using efforts greater than 6–8 comes at a significant cost in speed.
- Using fewer stages for "less gate delays" does not make a circuit faster. Making gates larger also does not make a circuit faster; it only increases the area and power consumption.
- The delay of a well-designed path is about $\log_4 F$ fanout-of-4 (FO4) inverter delays. Each quadrupling of the load adds about one FO4 inverter delay to the path. Control signals fanning out to a 64-bit datapath therefore incur an amplification delay of about three FO4 inverters.

- The logical effort of each input of a gate increases through no fault of its own as the number of inputs grows. Considering both logical effort and parasitic delay, we find a practical limit of about four series transistors in logic gates and about four inputs to multiplexers. Beyond this fan-in, it is faster to split gates into multiple stages of skinnier gates.
- Inverters or 2-input NAND gates with low logical efforts are best for driving nodes with a large branching effort. Use small gates after the branches to minimize load on the driving gate.
- When a path forks and one leg is more critical than the others, buffer the noncritical legs to reduce the branching effort on the critical path.

4.5.5 Limitations of Logical Effort

OPTIONAL

Logical Effort is based on the linear delay model and the simple premise that making the effort delays of each stage equal minimizes path delay. This simplicity is the method's greatest strength, but also results in a number of limitations:

- Logical Effort does not account for interconnect. The effects of nonnegligible wire capacitance and RC delay will be revisited in Chapter 6. Logical Effort is most applicable to high-speed circuits with regular layouts where routing delay does not dominate. Such structures include adders, multipliers, memories, and other datapaths and arrays.
- Logical Effort explains how to design a critical path for maximum speed, but not how to design an entire circuit for minimum area or power given a fixed speed constraint. This problem is addressed in Section 5.2.2.1.
- Paths with nonuniform branching or reconvergent fanout are difficult to analyze by hand.
- The linear delay model fails to capture the effect of input slope. Fortunately, edge rates tend to be about equal in well-designed circuits with equal effort delay per stage.

4.5.6 Iterative Solutions for Sizing

OPTIONAL

To address the limitations in the previous section, we can write the delay equations for each gate in the system and minimize the latest arrival time. No closed-form solutions exist, but the equations are easy to solve iteratively on a computer and the formulation still gives some insight for the designer. This section examines sizing for minimum delay, while Section 5.2.2.1 examines sizing for minimum energy subject to a delay constraint.

The ith gate is characterized by its logical effort, g_i, parasitic delay, p_i, and drive, x_i. Formally, our goal is to find a nonnegative vector of drives x that minimizes the arrival time of the latest output. This can be done using a commercial optimizer such as MOSEK or, for smaller problems, Microsoft Excel's solver. The arrival time equations are classified as *convex*, which has the pleasant property of having a single optimum; there is no risk of finding a wrong answer. Moreover, they are of a special class of functions called *posynomials*, which allows an especially efficient technique called *geometric programming* to be applied [Fishburn85].