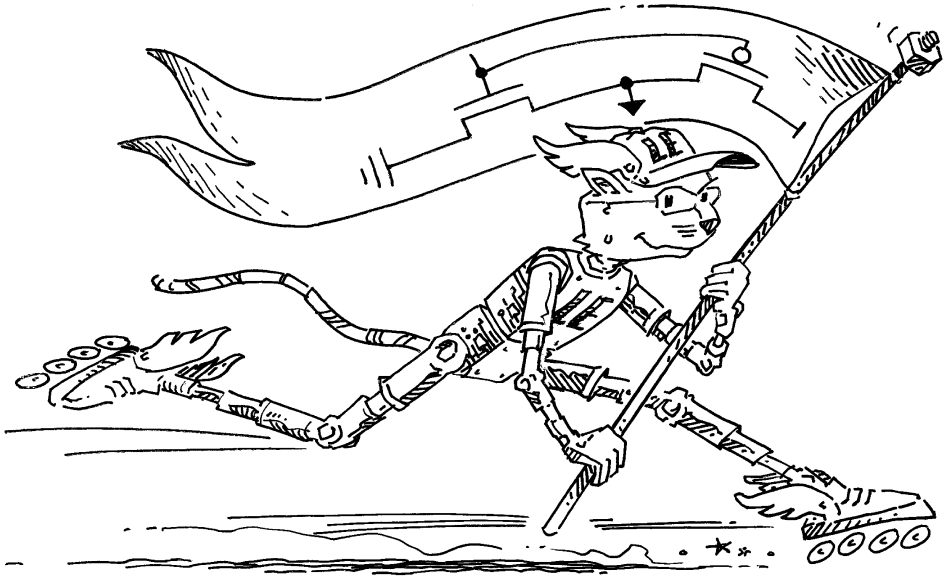

The Method of Logical Effort——1

Designing a circuit to achieve the greatest speed or to meet a delay constraint presents a bewildering array of choices. Which of several circuits that produce the same logic function will be fastest? How large should a logic gate's transistors be to achieve least delay? And how many stages of logic should be used to obtain least delay? Sometimes, adding stages to a path reduces its delay!

The *method of logical effort* is an easy way to estimate delay in a CMOS circuit. We can select the fastest candidate by comparing delay estimates of different logic structures. The method also specifies the proper number of logic stages on a path and the best transistor sizes for the logic gates. Because the method is easy to use, it is ideal for evaluating alternatives in the early stages of a design and provides a good starting point for more intricate optimizations.

This chapter describes the method of logical effort and applies it to simple examples. Chapter 2 explores more complex examples. These two chapters together provide all you need to know to apply the method of logical effort to a wide class of circuits. We devote the remainder of this book to derivations that show why the method of logical effort works, to some detailed optimization

techniques, and to the analysis of special circuits such as domino logic and multiplexers.

1.1 Introduction

To set the context of the problems addressed by logical effort, we begin by reviewing a simple integrated circuit design flow. We will see that topology selection and gate sizing are key steps of the flow. Without a systematic approach, these steps are extremely tedious and time-consuming. Logical effort offers such an approach to these problems.

Figure 1.1 shows a simplified chip design flow illustrating the logic, circuit, and physical design stages. The design starts with a specification, typically in textual form, defining the functionality and performance targets of the chip. Most chips are partitioned into more manageable blocks so that they may be divided among multiple designers and analyzed in pieces by CAD tools. Logic designers write register transfer level (RTL) descriptions of each block in a language like Verilog or VHDL and simulate these models until they are convinced the specification is correct. Based on the complexity of the RTL descriptions, the designers estimate the size of each block and create a floorplan showing relative placement of the blocks. The floorplan allows wire-length estimates and provides goals for the physical design.

Given the RTL and floorplan, circuit design may begin. There are two general styles of circuit design: custom and automatic. *Custom* design trades additional human labor for better performance. In a custom methodology, the circuit designer has flexibility to create cells at a transistor level or choose from a library of predefined cells. The designer must make many decisions: Should I use static CMOS, transmission gate logic, domino circuits, or other circuit families? What circuit topology best implements the functions specified in the RTL? Should I use NAND, NOR, or complex gates? After selecting a topology and drawing the schematics, the designer must choose the size of transistors in each logic gate. A larger gate drives its load more quickly, but presents greater input capacitance to the previous stage and consumes more area and power. When the schematics are complete, functional verification checks that the schematics correctly implement the RTL specification. Finally, timing verification checks that the circuits meet the performance targets. If performance is inadequate, the circuit designer may try to resize gates for improved speed, or may have to

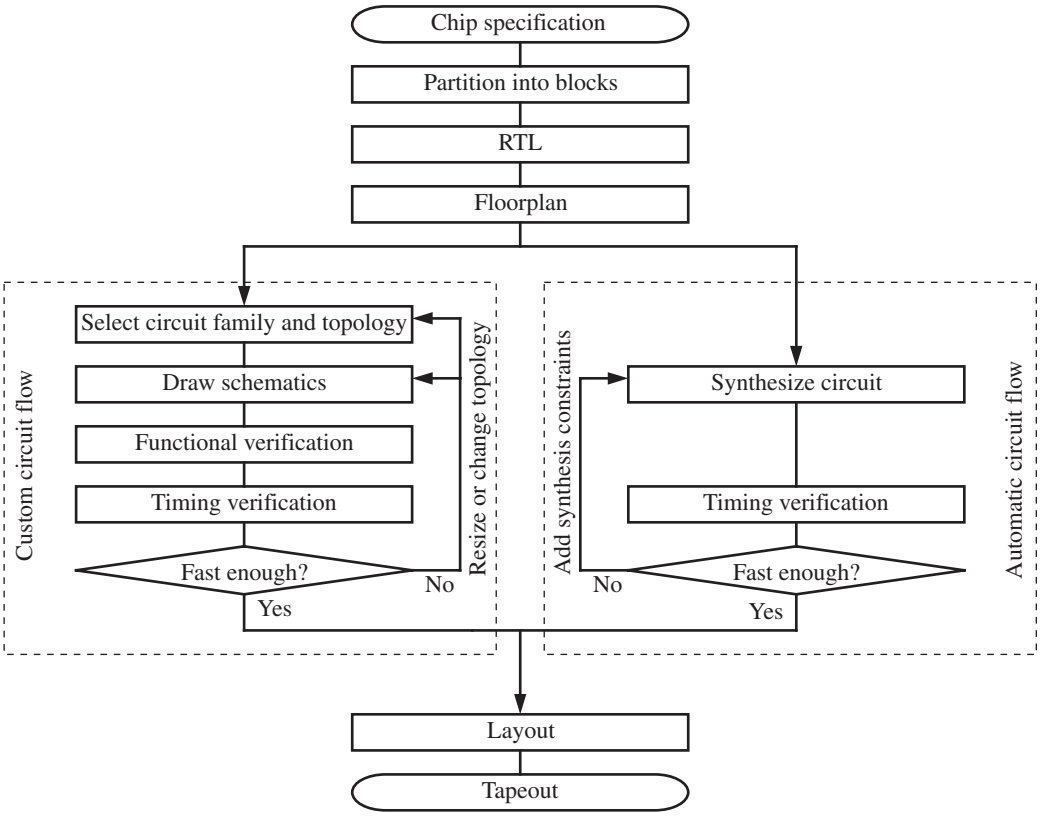

Figure 1.1 — Simplified chip design flow.

change the topology entirely, exploiting parallelism to build faster structures at the expense of more area or switching from static CMOS to faster domino gates.

Automatic circuit design uses synthesis tools to choose circuit topologies and gate sizes. Synthesis takes much less time than manually optimizing paths and drawing schematics, but is generally restricted to a fixed library of static CMOS cells and produces slower circuits than those designed by a skilled engineer. Advances in synthesis and manufacturing technology continue to expand the set of problems that synthesis can acceptably solve, but for the foreseeable future, high-end designs will require at least some custom circuits. Synthesized circuits are normally logically correct by construction, but timing verification is still

necessary. If performance is inadequate, the circuit designer may set directives for the synthesis tool to improve critical paths.

When circuit design is complete, layout may begin. Layout may also be custom or may use automatic place and route tools. Design rule checkers (DRC) and layout versus schematic (LVS) checks are used to verify the layout. Postlayout timing verification ensures the design still meets timing goals after including more accurate capacitance and resistance data extracted from the layout; if the estimates used in circuit design were inaccurate, the circuits may have to be modified again. Finally, the chip is "taped out" and sent for manufacturing.

One of the greatest challenges in this design flow is meeting the timing specifications, a problem known as *timing convergence*. If speed were not a concern, circuit design would be much easier, but if speed were not a concern, the problem could be solved more cost-effectively in software.

Even experienced custom circuit designers often expend a tremendous amount of frustrating effort to meet timing specifications. Without a systematic approach, most of us fall into the "simulate and tweak" trap of making changes in a circuit, throwing it into the simulator, looking at the result, making more changes, and repeating. Because circuit blocks often take half an hour or more in simulation, this process is very time-consuming. Moreover, the designer often tries to speed up a slow gate by increasing its size. This can be counterproductive if the larger gate now imposes greater load on the previous stage, slowing the previous stage more than improving its own delay! Another problem is that without an easy way of estimating delays, the designer who wishes to compare two topologies must draw, size, and simulate a schematic of each. This process takes a great deal of time and discourages such comparisons. The designer soon realizes that a more efficient and systematic approach is needed and over the years develops a personal set of heuristics and mental models to assist with topology selection and sizing.

Users of synthesis tools experience similar frustrations with timing convergence, especially when the specification is near the upper limit of the tool's capability. The synthesis equivalent of "simulate and tweak" is "add constraints and resynthesize"; as constraints fix one timing violation, they often introduce a new violation on another path. Unless the designer looks closely at the output of the synthesis and understands the root cause of the slow paths, adding constraints and resynthesizing may never converge on an acceptable result.

This book is written for those who are concerned about designing fast chips. It offers a systematic approach to topology selection and gate sizing that captures many years of experience and offers a simple language for quantitatively discussing such problems. In order to reason about such questions, we need a simple delay model that's fast and easy to use. The models should be accurate enough that if it predicts circuit *a* is significantly faster than circuit *b*, then circuit *a* really is faster; the absolute delays predicted by the model are not as important because a better simulator or timing analyzer will be used for timing verification. This chapter begins by discussing such a simple model of delay and introduces terms that describe how the complexity of the gate, the load capacitance, and the parasitic capacitance contribute to delay. From this model, we introduce a numeric "path effort" that allows the designer to compare two multistage topologies easily without sizing or simulation. We also describe procedures for choosing the best number of stages of gates and for selecting each gate size to minimize delay. Many examples illustrate these key ideas and show that using fewer stages or larger gates may fail to produce faster circuits.

1.2 —— Delay in a Logic Gate

The method of logical effort is founded on a simple model of the delay through a single MOS logic gate.[1] The model describes delays caused by the capacitive load that the logic gate drives and by the topology of the logic gate. Clearly, as the load increases, the delay increases, but delay also depends on the logic function of the gate. Inverters, the simplest logic gates, drive loads best and are often used as amplifiers to drive large capacitances. Logic gates that compute other functions require more transistors, some of which are connected in series, making them poorer than inverters at driving current. Thus a NAND gate has more delay than an inverter with similar transistor sizes that drives the same load. The method of logical effort quantifies these effects to simplify delay analysis for individual logic gates and multistage logic networks.

1. The term "gate" is ambiguous in integrated circuit design, signifying either a circuit that implements a logic function such as NAND or the gate of a MOS transistor. We hope to avoid confusion by referring to "logic gate" or "transistor gate" unless the meaning is clear from context.

The first step in modeling delays is to isolate the effects of a particular integrated circuit fabrication process by expressing all delays in terms of a basic *delay unit* τ particular to that process.[2] τ is the delay of an inverter driving an identical inverter with no parasitics. Thus we express absolute delay as the product of a unitless delay of the gate d and the delay unit that characterizes a given process:

$$d_{abs} = d\tau \tag{1.1}$$

Unless otherwise indicated, we will measure all times in units of τ. In a typical 0.6μ process τ is about 50 ps. This and other typical process parameters are summarized in Appendix B.

The delay incurred by a logic gate is comprised of two components, a fixed part called the *parasitic delay* p and a part that is proportional to the load on the gate's output, called the *effort delay* or *stage effort* f. (Appendix A lists all of the notation used in this book.) The total delay, measured in units of τ, is the sum of the effort and parasitic delays:

$$d = f + p \tag{1.2}$$

The effort delay depends on the load and on properties of the logic gate driving the load. We introduce two related terms for these effects: the *logical effort* g captures properties of the logic gate, while the *electrical effort* h characterizes the load. The effort delay of the logic gate is the product of these two factors:

$$f = gh \tag{1.3}$$

The logical effort g captures the effect of the logic gate's topology on its ability to produce output current. It is independent of the size of the transistors in the circuit. The electrical effort h describes how the electrical environment of the logic gate affects performance and how the size of the transistors in the gate determines its load-driving capability. The electrical effort is defined by:

$$h = \frac{C_{out}}{C_{in}} \tag{1.4}$$

2. This definition of τ differs from that used by Mead and Conway [7].

Table 1.1 — Logical effort for inputs of static CMOS gates, assuming $\gamma = 2$. γ is the ratio of an inverter's pullup transistor width to pulldown transistor width. Chapter 4 explains how to calculate the logical effort of these and other logic gates.

	Number of inputs					
Gate type	**1**	**2**	**3**	**4**	**5**	*n*
Inverter	1					
NAND		4/3	5/3	6/3	7/3	$(n+2)/3$
NOR		5/3	7/3	9/3	11/3	$(2n+1)/3$
Multiplexer		2	2	2	2	2
XOR (parity)		4	12	32		

where C_{out} is the capacitance that loads the output of the logic gate and C_{in} is the capacitance presented by the input terminal of the logic gate. Electrical effort is also called *fanout* by many CMOS designers. Note that fanout, in this context, depends on the load capacitance, not just the number of gates being driven.

Combining Equations 1.2 and 1.3, we obtain the basic equation that models the delay through a single logic gate, in units of τ:

$$d = gh + p \tag{1.5}$$

This equation shows that logical effort g and electrical effort h both contribute to delay in the same way. This formulation separates τ, g, h, and p, the four contributions to delay. The process parameter τ represents the speed of the basic transistors. The parasitic delay p expresses the intrinsic delay of the gate due to its own internal capacitance, which is largely independent of the size of the transistors in the logic gate. The electrical effort, h, combines the effects of external load, which establishes C_{out}, with the sizes of the transistors in the logic gate, which establish C_{in}. The logical effort g expresses the effects of circuit topology on the delay free of considerations of loading or transistor size. Logical effort is useful because it depends only on circuit topology.

Logical effort values for a few CMOS logic gates are shown in Table 1.1. Logical effort is defined so that an inverter has a logical effort of 1. An inverter driving an exact copy of itself experiences an electrical effort of 1. Therefore, an

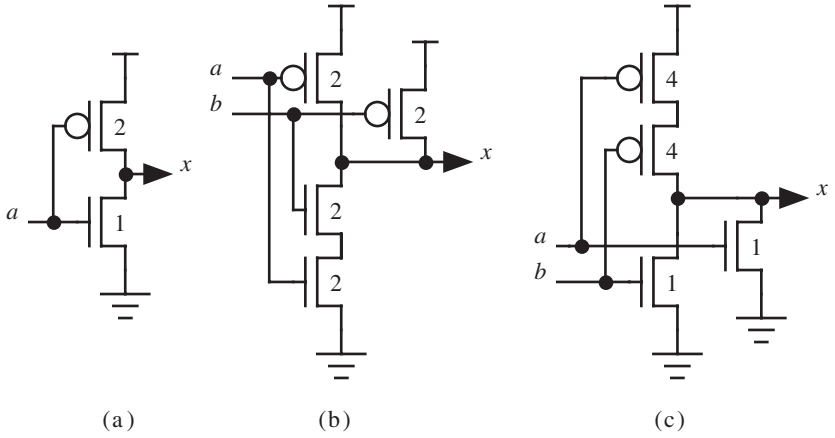

Figure 1.2 — Simple gates: inverter (a), two-input NAND gate (b), and two-input NOR gate (c). The numbers indicate relative transistor widths.

inverter driving an exact copy of itself will have an effort delay of 1, according to Equation 1.3.

The logical effort of a logic gate tells how much worse it is at producing output current than is an inverter, given that each of its inputs may present only the same input capacitance as the inverter. Reduced output current means slower operation, and thus the logical effort number for a logic gate tells how much more slowly it will drive a load than would an inverter. Equivalently, logical effort is how much more input capacitance a gate must present in order to deliver the same output current as an inverter. Figure 1.2 illustrates simple gates with relative transistor widths chosen for roughly equal output currents. The inverter has three units of input capacitance while the NAND has four. Therefore, the NAND gate has a logical effort $g = 4/3$. Similarly, the NOR gate has $g = 5/3$. Chapter 4 estimates the logical effort of other gates, while Chapter 5 shows how to extract logical effort from circuit simulations.

It is interesting but not surprising to note from Table 1.1 that more complex logic functions have larger logical effort. Moreover, the logical effort of most logic gates grows with the number of inputs to the gate. Larger or more complex logic gates will thus exhibit greater delay. As we shall see later, these properties make it worthwhile to contrast different choices of logical structure. Designs that minimize the number of stages of logic will require more inputs for each logic gate and thus have larger logical effort. Designs with fewer inputs and thus

less logical effort per stage may require more stages of logic. In Section 1.4, we will see how the method of logical effort expresses these trade-offs.

The electrical effort h is just a ratio of two capacitances. The load driven by a logic gate is the capacitance of whatever is connected to its output; any such load will slow down the circuit. The input capacitance of the circuit is a measure of the size of its transistors. The input capacitance term appears in the denominator of Equation 1.4 because bigger transistors in a logic gate will drive a given load faster. Usually most of the load on a stage of logic is the capacitance of the input or inputs of the next stage or stages of logic that it drives. Of course, the load also includes the stray capacitance of wires, drain regions of transistors, and so on. We shall see later how to include stray load capacitances in our calculations.

Electrical effort is usually expressed as a ratio of transistor widths rather than actual capacitances. We know that the capacitance of a transistor gate is proportional to its area; if we assume that all transistors have the same minimum length, then the capacitance of a transistor gate is proportional to its width. Because most logic gates drive other logic gates, we can express both C_{in} and C_{out} in terms of transistor widths. If the load capacitance includes stray capacitance due to wiring or external loads, we shall convert this capacitance into an equivalent transistor width. If you prefer, you can think of the unit of capacitance as the capacitance of a transistor gate of minimum length and unit width.

The parasitic delay of a logic gate is fixed, independent of the size of the logic gate and of the load capacitance it drives, because wider transistors providing greater output current have correspondingly greater diffusion capacitance. This delay is a form of overhead that accompanies any gate. The principal contribution to parasitic delay is the capacitance of the source or drain regions of the transistors that drive the gate's output. Table 1.2 presents crude estimates of parasitic delay for a few logic gate types; note that parasitic delays are given as multiples of the parasitic delay of an inverter, denoted as p_{inv}. A typical value for p_{inv} is 1.0 delay units, which is used in most of the examples in this book. p_{inv} is a strong function of process-dependent diffusion capacitances, but 1.0 is representative and is convenient for hand analysis. These estimates omit stray capacitance between series transistors, as will be discussed in more detail in Chapters 3 and 5.

The delay model of a single logic gate, as represented in Equation 1.5, is a simple linear relationship. Figure 1.3 shows this relationship graphically: delay appears as a function of electrical effort for an inverter and for a two-input NAND

Table 1.2 — Estimates of parasitic delay of various logic gate types, assuming simple layout styles. A typical value of p_{inv}, the parasitic delay of an inverter, is 1.0.

Gate type	Parasitic delay
Inverter	p_{inv}
n-input NAND	np_{inv}
n-input NOR	np_{inv}
n-way multiplexer	$2np_{inv}$
XOR, XNOR	$4p_{inv}$

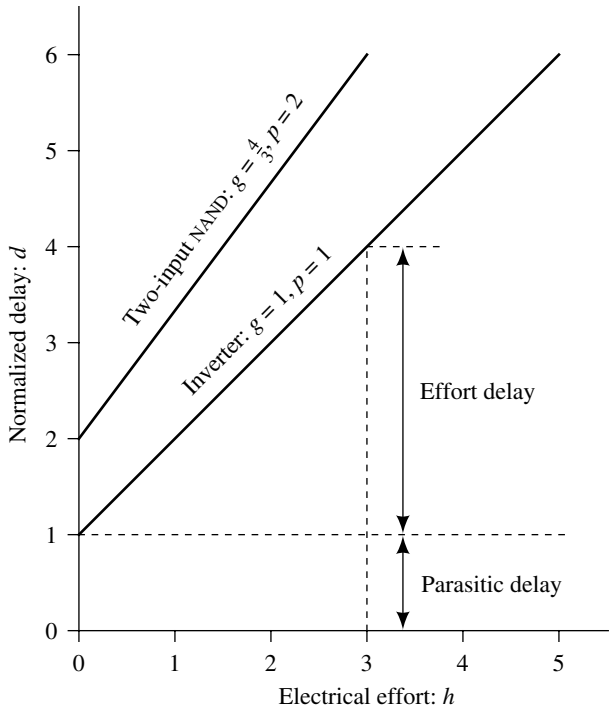

Figure 1.3 — Plots of the delay equation for an inverter and a two-input NAND gate.

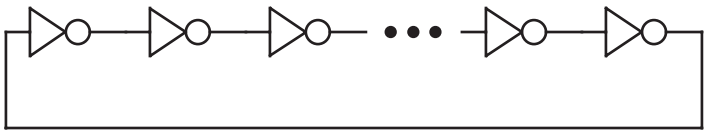

Figure 1.4 — A ring oscillator of N identical inverters.

gate. The slope of each line is the logical effort of the gate; its intercept is the parasitic delay. The graph shows that we can adjust the total delay by adjusting the electrical effort or by choosing a logic gate with a different logical effort. Once we have chosen a gate type, however, the parasitic delay is fixed, and our optimization procedure can do nothing to reduce it.

Example 1.1 Estimate the delay of an inverter driving an identical inverter, as in the ring oscillator shown in Figure 1.4.

Solution Because the inverter's output is connected to the input of an identical inverter, the load capacitance, C_{out}, is the same as the input capacitance. Therefore the electrical effort is $h = C_{out}/C_{in} = 1$. Because the logical effort of an inverter is 1, we have, from Equation 1.5, $d = gh + p = 1 \times 1 + p_{inv} = 2.0$. This result expresses the delay in *delay units*; it can be scaled by τ to obtain the absolute delay, $d_{abs} = 2.0\tau$. In a 0.6μ process with $\tau = 50$ ps, $d_{abs} = 100$ ps.

The ring oscillator shown in Figure 1.4 can be used to measure the value of τ. Because N, the number of stages in the ring, is odd, the circuit is unstable and will oscillate. The delay of each stage of the ring oscillator is expressed by:

$$\frac{1}{2NF} = d\tau = (1 + p_{inv})\tau \tag{1.6}$$

where N is the number of inverters, F is the oscillation frequency, and the 2 appears because a transition must pass twice around the ring to complete a single cycle of the oscillation. If a value for p_{inv} is known, this equation can be used to determine τ from measurements of the frequency of the ring oscillator. Chapter 5 shows a method for measuring both τ and p_{inv}. ■

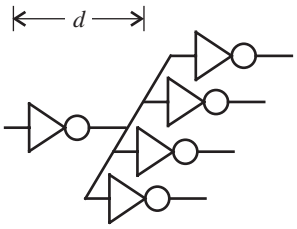

Figure 1.5 — An inverter driving four identical inverters.

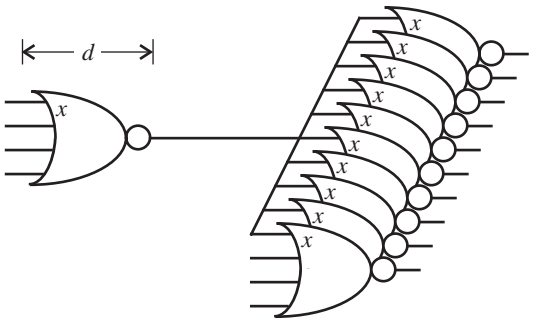

Figure 1.6 — A four-input NOR gate driving 10 identical gates.

Example 1.2 Estimate the delay of a fanout-of-4 (FO4) inverter, as shown in Figure 1.5.

Solution Because each inverter is identical, $C_{out} = 4C_{in}$, so $h = 4$. The logical effort $g = 1$ for an inverter. Thus the FO4 delay, according to Equation 1.5, is $d = gh + p = 1 \times 4 + p_{inv} = 4 + 1 = 5$. It is sometimes convenient to express times in terms of FO4 inverter delays because most designers know the FO4 delay in their process and can use it to estimate the absolute performance of your circuit in their process. ■

Example 1.3 A four-input NOR gate drives 10 identical gates, as shown in Figure 1.6. What is the delay in the driving NOR gate?

Solution If the capacitance of one input of each NOR gate is x, then the driving NOR has $C_{in} = x$ and $C_{out} = 10x$, and thus the electrical effort is $h = 10$. The logical effort of the four-input NOR gate is $9/3 = 3$, obtained from Table 1.1. Thus the delay is $d = gh + p = 3 \times 10 + 4 \times 1$, or 34 delay units. Note that

when the load is large, as in this example, the parasitic delay is insignificant compared to the effort delay. ■

1.3 Multistage Logic Networks

The method of logical effort reveals the best number of stages in a multistage network and how to obtain the least overall delay by balancing the delay among the stages. The notions of logical and electrical effort generalize easily from individual gates to multistage paths.

The logical effort along a path compounds by multiplying the logical efforts of all the logic gates along the path. We use the uppercase symbol G to denote the *path logical effort*, so that it is distinguished from g, the logical effort of a single gate in the path. The subscript i indexes the logic stages along the path.

$$G = \prod g_i \tag{1.7}$$

The electrical effort along a path through a network is simply the ratio of the capacitance that loads the last logic gate in the path to the input capacitance of the first gate in the path. We use an uppercase symbol H to indicate the electrical effort along a path.

$$H = \frac{C_{out}}{C_{in}} \tag{1.8}$$

In this case, C_{in} and C_{out} refer to the input and output capacitances of the path as a whole, as may be inferred from context.

We need to introduce a new kind of effort, named *branching effort,* to account for fanout within a network. So far we have treated fanout as a form of electrical effort: when a logic gate drives several loads, we sum their capacitances, as in Example 1.3, to obtain an electrical effort. Treating fanout as a form of electrical effort is easy when the fanout occurs at the final output of a network. This method is less suitable when the fanout occurs within a logic network because we know that the electrical effort for the network depends only on the ratio of its output capacitance to its input capacitance.

When fanout occurs within a logic network, some of the available drive current is directed along the path we are analyzing, and some is directed off that path. We define the branching effort b at the output of a logic gate to be

$$b = \frac{C_{on-path} + C_{off-path}}{C_{on-path}} = \frac{C_{total}}{C_{useful}} \tag{1.9}$$

where $C_{on-path}$ is the load capacitance along the path we are analyzing and $C_{off-path}$ is the capacitance of connections that lead off the path. Note that if the path does not branch, the branching effort is one. The branching effort along an entire path B is the product of the branching effort at each of the stages along the path.

$$B = \prod b_i \tag{1.10}$$

Armed with definitions of logical, electrical, and branching effort along a path, we can define the *path effort* F. Again, we use an uppercase symbol to distinguish the path effort from the stage effort f associated with a single logic stage. The equation that defines path effort is reminiscent of Equation 1.3, which defines the effort for a single logic gate:

$$F = GBH \tag{1.11}$$

Note that the path branching and electrical efforts are related to the electrical effort of each stage:

$$BH = \frac{C_{out}}{C_{in}} \prod b_i = \prod h_i \tag{1.12}$$

The designer knows C_{in}, C_{out}, and branching efforts b_i from the path specification. Sizing the path consists of choosing appropriate electrical efforts h_i for each stage to match the total BH product.

Although it is not a direct measure of delay along the path, the path effort holds the key to minimizing the delay. Observe that the path effort depends only on the circuit topology and loading and not upon the sizes of the transistors used in logic gates embedded within the network. Moreover, the effort is unchanged if inverters are added to or removed from the path, because the logical effort of an inverter is one. The path effort is related to the minimum achievable delay along the path, and permits us to calculate that delay easily. Only a little more work yields the best number of stages and the proper transistor sizes to realize the minimum delay.

The path delay D is the sum of the delays of each of the stages of logic in the path. As in the expression for delay in a single stage (Equation 1.5), we shall distinguish the *path effort delay* D_F and the *path parasitic delay* P:

$$D = \sum d_i = D_F + P \tag{1.13}$$

The path effort delay is simply

$$D_F = \sum g_i h_i \tag{1.14}$$

and the path parasitic delay is

$$P = \sum p_i \tag{1.15}$$

Optimizing the design of an N-stage logic network proceeds from a very simple principle that we will prove in Chapter 3: *The path delay is least when each stage in the path bears the same stage effort.* This minimum delay is achieved when the stage effort is

$$\hat{f} = g_i h_i = F^{1/N} \tag{1.16}$$

We use a hat over a symbol to indicate an expression that achieves minimum delay.

Combining these equations, we obtain the principal result of the method of logical effort, which is an expression for the minimum delay achievable along a path:

$$\hat{D} = NF^{1/N} + P \tag{1.17}$$

From a simple computation of its logical, branching, and electrical efforts we can obtain an estimate of the minimum delay of a logic network. Observe that when $N = 1$, this equation reduces to Equation 1.5.

To equalize the effort borne by each stage on a path, and therefore achieve the minimum delay along the path, we must choose appropriate transistor sizes for each stage of logic along the path. Equation 1.16 shows that each logic stage should be designed with electrical effort

$$\hat{h}_i = \frac{F^{1/N}}{g_i} \tag{1.18}$$

From this relationship, we can determine the transistor sizes of gates along a path. Start at the end of the path and work backward, applying the capacitance transformation:

$$C_{in_i} = \frac{g_i C_{out_i}}{\hat{f}} \tag{1.19}$$

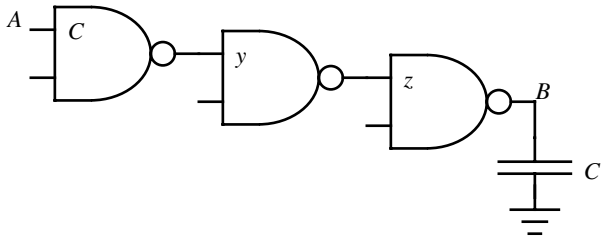

Figure 1.7 — A logic network consisting of three two-input NAND gates.

This determines the input capacitance of each gate, which can then be distributed appropriately among the transistors connected to the input. The mechanics of this process will become clear in the following examples.

EXAMPLE 1.4 Consider the path from A to B involving three two-input NAND gates shown in Figure 1.7. The input capacitance of the first gate is C, and the load capacitance is also C. What is the least delay of this path, and how should the transistors be sized to achieve least delay? (The next example will use the same circuit with a different electrical effort.)

SOLUTION To compute the path effort, we must compute the logical, branching, and electrical efforts along the path. The path logical effort is the product of the logical efforts of the three NAND gates, $G = g_0 g_1 g_2 = 4/3 \times 4/3 \times 4/3 = (4/3)^3 = 2.37$. The branching effort is $B = 1$, because all of the fanouts along the path are one, that is, there is no branching. The electrical effort is $H = C/C = 1$. Hence, the path effort is $F = GBH = 2.37$. Using Equation 1.17, we find the least delay achievable along the path to be $\hat{D} = 3(2.37)^{1/3} + 3(2p_{inv}) = 10.0$ delay units.

This minimum delay can be realized if the transistor sizes in each logic gate are chosen properly. First compute the stage effort $\hat{f} = 2.37^{1/3} = 4/3$. Starting with the output load C, apply the capacitance transformation of Equation 1.19 to compute input capacitance $z = C \times (4/3)/(4/3) = C$. Similarly, $y = z \times (4/3)/(4/3) = z = C$. Hence we find that all three NAND gates should have the same input capacitance, C. In other words, the transistor sizes in the three gates will be the same. This is not a surprising result: all stages have the same load and the same logical effort, and hence bear equal effort, which is the condition for minimizing path delay.

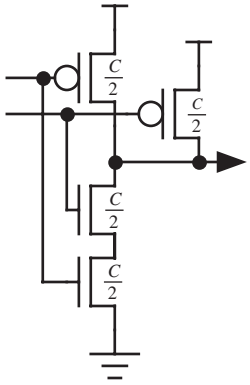

Figure 1.8 — A schematic of a NAND gate from Example 1.4.

A schematic of the NAND gate is shown in Figure 1.8, assuming PMOS transistors have half the mobility of NMOS transistors. Selecting transistor sizes will be discussed further in Chapter 4. Since each input drives both a PMOS and NMOS transistor with capacitance $C/2$, the capacitance of each input is C, as desired. ■

EXAMPLE 1.5 Using the same network as in the previous example, Figure 1.7, find the least delay achievable along the path from A to B when the output capacitance is $8C$.

SOLUTION Using the result from Example 1.4 that $G = (4/3)^3$ and the new electrical effort $H = 8C/C = 8$, we compute $F = GBH = (4/3)^3 \times 8 = 18.96$, so the least path delay is $\hat{D} = 3(18.96)^{1/3} + 3(2p_{inv}) = 14.0$ delay units. Observe that although the electrical effort in this example is eight times the electrical effort in the earlier example, the delay is increased by only 40%.

Now let us compute the transistor sizes that achieve minimum delay. The stage effort $\hat{f} = 18.96^{1/3} = 8/3$. Starting with the output load $8C$, apply the capacitance transformation of Equation 1.19 to compute input capacitance $z = 8C \times (4/3)/(8/3) = 4C$. Similarly, $y = z \times (4/3)/(8/3) = z/2 = 2C$. To verify the calculation, calculate the capacitance of the first gate $y \times (4/3)/(8/3) = y/2 = C$, matching the design specification. Each successive logic gate has twice the input capacitance of its predecessor. This is achieved by making the transistors in a gate twice as wide as the corresponding

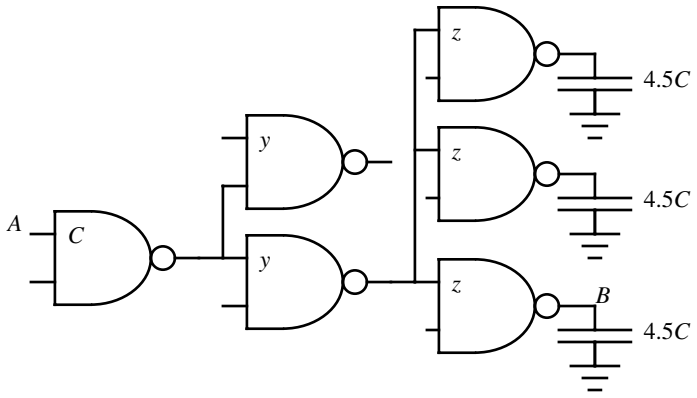

Figure 1.9 — A multistage logic network with internal fanout.

transistors in its predecessor. The wider transistors in successive stages are better able to drive current into the larger loads. ■

Example 1.6 Optimize the circuit in Figure 1.9 to obtain the least delay along the path from A to B when the electrical effort of the path is 4.5.

Solution The path logical effort is $G = (4/3)^3$. The branching effort at the output of the first stage is $(y + y)/y = 2$, and at the output of the second stage it is $(z + z + z)/z = 3$. The path branching effort is therefore $B = 2 \times 3 = 6$. The electrical effort along the path is specified to be $H = 4.5$. Thus $F = GBH = 64$, and $\hat{D} = 3(64)^{1/3} + 3(2p_{inv}) = 18.0$ delay units.

To achieve this minimum delay, we must equalize the effort in each stage. Since the path effort is 64, the stage effort should be $(64)^{1/3} = 4$. Starting from the output, $z = 4.5C \times (4/3)/4 = 1.5C$. The second stage drives three copies of the third stage, so $y = 3z \times (4/3)/4 = z = 1.5C$. We can check the math by finding the size of the first stage $2y \times (4/3)/4 = (2/3)y = C$, as given in the design spec. ■

Example 1.7 Size the circuit in Figure 1.10 for minimum delay. Suppose the load is 20 microns of gate capacitance and that the inverter has 10 microns of gate capacitance.

Solution Assuming minimum-length transistors, gate capacitance is proportional to gate width. Hence, it is convenient to express capacitance in terms of microns of gate width, as given in this problem.

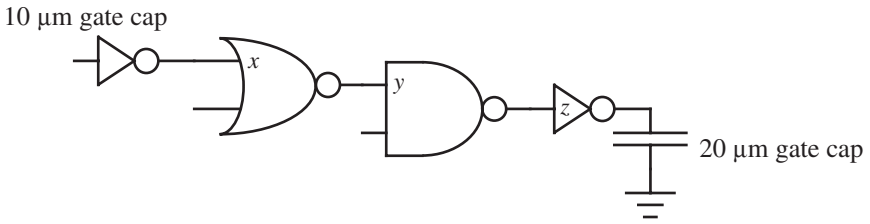

Figure 1.10 — A multistage logic network with a variety of gates.

The path has logical effort $G = 1 \times (5/3) \times (4/3) \times 1 = 20/9$. The electrical effort is $H = 20/10 = 2$, and the branching effort is 1. Thus, $F = GBH = 40/9$, and $\hat{f} = (40/9)^{1/4} = 1.45$.

Start from the output and work backward to compute sizes: $z = 20 \times 1/1.45 = 14$, $y = 14 \times (4/3)/1.45 = 13$, and $x = 13 \times (5/3)/1.45 = 15$. These input gate widths are divided among the transistors in each gate. Notice that the inverters are assigned larger electrical efforts than the more complex gates because they are better at driving loads. Also note that these calculations do not have to be very precise. We will see in Section 3.6 that sizing a gate too large or too small by a factor of 1.5 still results in circuits within 5% of minimum delay. Therefore, it is easy to use "back-of-the-envelope" hand calculations to find gate sizes to one or two significant figures.

Note that the parasitic delay does not enter into the procedure for calculating transistor sizes to obtain minimum delay. Because the parasitic delay is fixed, independent of the size of the logic gate, adjustments to the size of logic gates cannot alter parasitic delay. In fact, we can ignore parasitic delay entirely unless we want to obtain an accurate estimate of the time required for a signal to propagate through a logic network, or if we are comparing two logic networks that contain different types of logic gates or different numbers of stages and therefore exhibit different parasitic delays. ■

Example 1.8 Consider three alternative circuits for driving a load 25 times the input capacitance of the circuit. The first design uses one inverter, the second uses three inverters in series, and the third uses five in series. All three designs compute the same logic function. Which is best, and what is the minimum delay?

Solution In all three cases, the path logical effort is 1, the branching effort is 1, and the electrical effort is 25. Equation 1.17 gives the path delay $D = N(25)^{1/N} + Np_{inv}$ where $N = 1$, 3, or 5. For $N = 1$, we have $\hat{D} = 26$ delay units; for $N = 3$, $\hat{D} = 11.8$; and for $N = 5$, $\hat{D} = 14.5$. The best choice is $N = 3$. In this design, each stage will bear an effort of $(25)^{1/3} = 2.9$, so each inverter will be 2.9 times larger than its predecessor. This is the familiar geometric progression of sizes that is found in many textbooks. ■

This example shows that the fastest speed obtainable depends on the number of stages in the circuit. Since the path delay varies markedly for different values of N, it is clear we need a method for choosing N to yield the least delay; this is the topic of the next section.

1.4 Choosing the Best Number of Stages

The delay equations of logical effort, such as Equation 1.17, can be solved to determine the number of stages, $\hat{N}$, that achieves the minimum delay. Although we will defer the solution technique until Chapter 3, Table 1.3 presents some results. The table shows, for example, that a single stage is fastest only if the path effort F is 5.83 or less. If the path effort lies between 5.83 and 22.3, a two-stage design is best. If it lies between 22.3 and 82.2, three stages are best. The table confirms that the right number of stages to use in Example 1.8, which has $F = 25$, is three. As the effort gets very large, the stage effort approaches 3.59.

If we use Table 1.3 to select the number of stages that gives the least delay, we may find that we must add stages to a network. We can always add an even number of stages by attaching pairs of inverters to the end of the path. Because we can't add an odd number of inverters without changing the logic function of the network, we may have to settle for a somewhat slower design or alter the logic network to accommodate an inverted signal. If a path uses a number of stages that is not quite optimal, the overall delay is usually not increased very much; what is disastrous is a design with half or twice the best number of stages.

The table is accurate only when we are considering increasing or decreasing the number of stages in a path by adding or removing inverters, because the table assumes that stages being added or removed have a parasitic delay equal to that of an inverter. Chapter 3 explains how other similar tables can be produced. When we are comparing logic networks that use different logic gate types or different

Table 1.3 — Best number of stages to use for various path efforts. For example, for path efforts between 3920 and 14200, seven stages should be used; the stage effort will lie in the range 3.3–3.9 delay units. The table assumes $p_{inv} = 1.0$.

Path effort F	Best number of stages, $\hat{N}$	Minimum delay $\hat{D}$	Stage effort, f, range
0		1.0	
	1		0–5.8
5.83		6.8	
	2		2.4–4.7
22.3		11.4	
	3		2.8–4.4
82.2		16.0	
	4		3.0–4.2
300		20.7	
	5		3.1–4.1
1090		25.3	
	6		3.2–4.0
3920		29.8	
	7		3.3–3.9
14200		34.4	
	8		3.3–3.9
51000		39.0	
	9		3.3–3.9
184000		43.6	
	10		3.4–3.8
661000		48.2	
	11		3.4–3.8
2380000		52.8	
	12		3.4–3.8
8560000		57.4	

numbers of stages of logic, it is necessary to evaluate the delay equations to determine which design is best.

Example 1.9 A string of inverters in a 0.6μ process drives a signal that goes off-chip through a pad. The capacitance of the pad and its load is 40 *pF,* which is equivalent to about 20,000 microns of gate capacitance. Assuming the load on the input should be that of an inverter with 7.2 microns of input capacitance, what is the fastest inverter string?

Solution As in Example 1.8, the logical and branching efforts are both 1, but the electrical effort is $20{,}000/7.2 = 2777$. Table 1.3 specifies a six-stage design.

The stage effort will be $\hat{f} = (2777)^{1/6} = 3.75$. Thus the input capacitance of each inverter in the string will be 3.75 times that of its predecessor. The path delay will be $\hat{D} = 6 \times 3.75 + 6 \times p_{inv} = 28.5$ delay units. This corresponds to an absolute delay of $28.5\tau = 1.43$ ns, assuming $\tau = 50$ ps. ■

This example finds the best ratio of the sizes of succeeding stages to be 3.75. Many texts teach us to use a ratio of $e = 2.718$, but the reasoning behind the smaller value fails to account for parasitic delay. As the parasitic delay increases, the size ratio that achieves least delay rises above e, and the best number of stages to use decreases. Chapter 3 explores these issues further and presents a formula for the best stage effort.

In general, the best stage effort $\hat{f}$ is between 3 and 4. Targeting a stage effort of 4 is convenient during design and gives delays within 1% of minimum delay for typical parasitics. Thus, the number of stages $\hat{N}$ is about $\log_4 F$. We will find that stage efforts between 2 and 8 give delays within 35% of minimum and efforts between 2.4 and 6 give delays within 15% of minimum. Therefore, choosing the right stage effort is not critical.

We will also see in Chapter 3 that an easy way to estimate the delay of a path is to approximate the delay of a stage with effort of 4 as that of an FO4 inverter. We found in Example 1.2 that an FO4 inverter has a delay of 5 units. Therefore, the delay of a circuit with path effort F is about $5 \log_4 F$, or about $\log_4 F$ FO4 delays. This is somewhat optimistic because it neglects the larger parasitic delay of complex gates.

1.5 Summary of the Method

The method of logical effort is a design procedure for achieving the least delay along a path of a logic network. It combines into one calculation the effort required to drive large electrical loads and to perform logic functions. The principle expressions of logical effort are summarized in Table 1.4. The procedure is:

1. Compute the path effort $F = GBH$ along the path of the network you are analyzing. The path logical effort G is the product of the logical efforts of the logic gates along the path; use Table 1.1 to obtain the logical effort of each individual logic gate. The branching effort B is the product of the branching effort at each stage along the path. The electrical effort H is the ratio of the

Table 1.4 — Summary of terms and equations for concepts in the method of logical effort.

Term	Stage expression	Path expression
Logical effort	g (Table 1.1)	$G = \prod g_i$
Electrical effort	$h = C_{out}/C_{in}$	$H = C_{out-path}/C_{in-path}$
Branching effort	—	$B = \prod b_i$
Effort	$f = gh$	$F = GBH = \prod f_i$
Effort delay	f	$D_F = \sum f_i$ minimized when $f_i = F^{1/\hat{N}}$
Number of stages	1	N (Table 1.3)
Parasitic delay	p (Table 1.2)	$P = \sum p_i$
Delay	$d = f + p$	$D = D_F + P$

capacitance loading the last stage of the network to the input capacitance of the first stage of the network.

2. Use Table 1.3 or estimate $\hat{N} \approx \log_4 F$ to find out how many stages $\hat{N}$ will yield the least delay.
3. Estimate the minimum delay, $\hat{D} = \hat{N}F^{1/\hat{N}} + \sum p_i$, using values of parasitic delay obtained from Table 1.2. If you are comparing different architectural approaches to a design problem, you may choose to stop the analysis here.
4. Add or remove stages if necessary until N, the number of stages in your circuit, is approximately $\hat{N}$.
5. Compute the effort to be borne by each stage: $\hat{f} = F^{1/N}$.
6. Starting at the last logic stage in the path, work backward to compute transistor sizes for each of the logic gates by applying the equation $C_{in} = (g_i/\hat{f})C_{out}$ for each stage. The value of C_{in} for a stage becomes C_{out} for the previous stage, perhaps modified to account for branching effort.

This design procedure finds the circuit with the least delay along a particular path, without regard to area, power, or other limitations that may be as important as delay. In some cases, compromises will be necessary to obtain practical designs. For example, if this procedure is used to design drivers for a high-capacitance bus, the drivers may be too big to be practical. You may compromise by using a larger stage delay than the design procedure calls for, or even

by making the delay in the last stage much greater than in the other stages; both of these approaches reduce the size of the final driver at the expense of delay.

The method of logical effort achieves an *approximate* optimum. Because it ignores a number of second-order effects, such as stray capacitances between series transistors within logic gates, a circuit designed with the procedure given above can sometimes be improved by careful simulation with a circuit simulator and subsequent adjustment of transistor sizes. However, in our experience the method of logical effort alone obtains designs that are within 10% of the minimum.

Another limitation of logical effort is the fact that circuits with complex branches or interconnect have no closed-form best design. Chapters 9 and 10 address these issues and provide approximations useful when gate or wire loads dominate, but in some cases, iteration is still necessary.

One of the strengths of the method of logical effort is that it combines into one framework the effects on performance of capacitive load, of complexity of the logic function being computed, and of the number of stages in the network. For example, if you redesign a logic network to use high fan-in logic gates in order to reduce the number of stages, the logical effort increases, thus blunting the improvement. Although many designers recognize that large capacitive loads must be driven with strings of drivers that increase in size geometrically, they are not sure what happens when logic is mixed in, as occurs often in tristate drivers. The method of logical effort addresses all of these design problems.

1.6 —— A Look Ahead

The information presented in this chapter is sufficient to attack almost any design. The next chapter applies the method to a variety of circuits of practical importance. Chapter 3 exposes the model behind the method and derives the equations presented in this chapter. Chapter 4 shows how to compute the logical effort of a logic gate and exhibits a catalog of logic gate types. Chapter 5 describes how to measure various parameters required by the method, such as p_{inv} and τ. The remaining chapters explore refinements to the method and more intricate design problems. Chapters 6 and 7 describe how to unbalance or skew a gate to favor a particular input or transition at the expense of the others. Chapter 8 applies logical effort to other circuit families, including pseudo-NMOS, domino, and transmission gates. Chapters 9 and 10 tackle the problems of cir-

cuits that fork and branch in irregular fashions. Chapter 11 uses logical effort to gain insights on wide structures including many-input gates, decoders, and multiplexers. The conclusion in Chapter 12 summarizes the method of logical effort and many insights provided by the method. It gives a design procedure to apply logical effort and compares the procedure with other approaches to path design. Finally, it reviews some of the limitations of logical effort that are important to the designer. Even if you skip the middle chapters on a first reading, we still recommend you glance at the conclusion.

You may also wish to refer to the appendices from time to time. We recognize that the notation of logical effort can be confusing at first, so Appendix A contains a complete list of all the symbols with definitions. Appendix B summarizes nominal process parameters for the 0.6μ process used in examples throughout the book and Appendix C contains solutions to the odd-numbered exercises.

1.7 —— Exercises

The bracketed numbers to the left of each exercise indicate the degree of difficulty for each. Please see About the Exercises in the Preface for a ratings guide.

[20] **1-1** Consider the circuits shown in Figure 1.11. Both have a fanout of 6, that is, they must drive a load six times the capacitance of each of the inputs. What is the path effort of each design? Which will be fastest? Compute the sizes x and y of the logic gates required to achieve least delay.

[20] **1-2** Design the fastest circuit that computes the NAND of four inputs with a fanout of 6. Consider a four-input NAND gate by itself, a four-input NAND gate followed by two additional inverters, and a tree formed by 2 two-input NAND gates whose outputs are connected to a two-input NOR gate followed by an

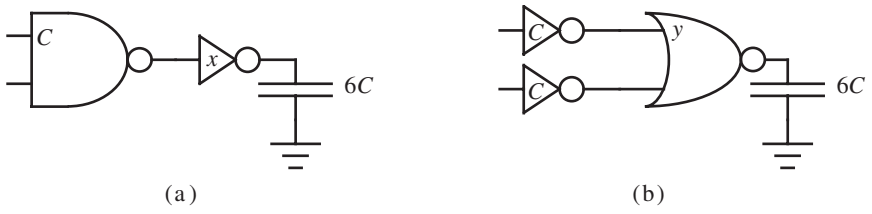

Figure 1.11 — Two circuits for computing the AND function of two inputs.

inverter. Estimate the shortest delay achievable for each circuit. If the fanout were larger, would other circuits be better?

[10] **1-3** A three-stage logic path is designed so that the effort borne by each stage is 10, 9, and 7 delay units, respectively. Can this design be improved? Why? What is the best number of stages for this path? What changes do you recommend to the existing design?

[10] **1-4** A clock driver must drive 500 minimum-size inverters. If its input must be a single minimum-size inverter, how many stages of amplification should be used? If the input to the clock driver comes from outside the integrated circuit via an input pad, could fewer stages be used? Why?

[15] **1-5** A particular system design of interest will have eight levels of logic between latches. Assuming that the most complex circuits involve four-input NAND gates with fanouts of three in all eight levels of logic and that latching overhead is negligible, estimate the minimum clock period.

[20] **1-6** A long metal wire carries a signal from one part of a chip to another. Only a single unit load may be imposed on the signal source. At its destination the signal must drive 20 unit loads. The distributed wire capacitance is equivalent to 100 unit loads; assume the wire has no resistance. Design a suitable amplifier. You may invert the signal if necessary. Should the amplifier be placed at the beginning, middle, or end of the wire?